

镜像站云原生 FS 与可观测性实践

尚子焜 @ SEU Mirrors

东南大学

2026-08-19

Intro

- 始于 2023 年的一次讨论
- 东南大学网信中心、SiliCOM 协会、Linux@SEU 支持创办
- 网信华为云本地机房 VM*2, 10Gbps 校园网内网出口
- 1T 本地 SSD + 38T/50T (raw) 浪潮 AS13000 对象存储
- 主要基于成熟的 tunasync 方案，部分特殊服务反代（pypi 等）
- 公开项目 @GitHub



东南大学开源镜像站 / SEU Mirrors^β

首页

新闻公告

同步状态



稳定、可靠的开源镜像服务

基于「快照」的原子化同步，无中间状态，始终完整。

Cloud-native (?) Filesystems

- 受制于网信云平台原因，无法拿到裸块存储
- 支持的网络存储协议较为有限： CIFS/NFS/S3

产品系列		AS13000G7-H (IDFS)
产品型号		AS13000G7-HS12/HS36/HS60, AS13000G7-HN24, AS13000G7-224
集群能力	存储协议	块/文件/对象/大数据
	集群节点规模	可随着业务需求横向扩展，最大10240节点
	集群容量规模	EB级
	数据冗余	2~8个副本， 纠删码(K+M、K+M:B；其中K取值范围为[2, 32]、M取值范围为[1, 4]、当K>=22时，M不能设置为4、当M=4时，要求K<=21)， 最大纠删比：32+2。 可根据数据不同的保护级别在同一系统内同时设置数据副本模式和纠删码模式
硬件规格	CPU	C86架构多核处理器
	内存	RDIMM、NVDIMM
	主机接口类型	1Gb、10Gb、25Gb、100Gb以太网网络接口
	磁盘类型	NVMe SSD、SATA SSD、SATA HDD、SAS HDD
	连接协议	支持POSIX、NFS、CIFS/SMB、FTP、S3、HDFS、iSCSI、NVMe-oF、RoCE v2、XBD、CSI、Cinder等访问协议
软件规格	高级软件功能	对象重删、压缩、数据分层、远程复制、对象多站点、智能磁盘预测、自动精简、配额管理、QoS、克隆、快照、增强ACL、多租户、WORM、融合互通、软拷贝、回收站、内核客户端、小文件聚合、对象聚合、元数据检索、亚健康检测、健康巡检、数据加密、多版本管理、报表、审计、对象数据生命周期、一键部署
	兼容平台/组件	兼容POSIX标准接口，Windows、Linux、UNIX、MacOS、VmwareESX等操作系统，KVM、vSphere、Xen、Hyper-V等虚拟化平台，兼容OpenStack云平台；兼容GB/T28181、RTSP、PSIA、H5、IDK、ISSC-SDK、ONVIF、私有协议等视频协议

Take 1: NFS + Btrfs

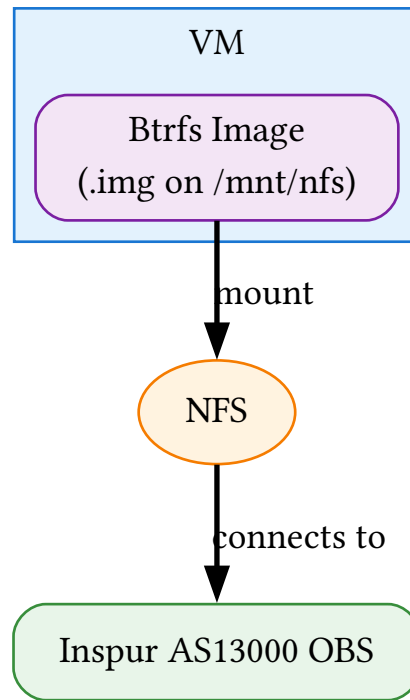
在裸 NFS 上创建 Btrfs disk image (.img)

优点:

- 文件系统级 CoW (Copy-on-Write)
- 原生支持多 volume (卷) 备份
- tunasync 后端支持较完善

缺点:

太慢!



Take 1: NFS + Btrfs

Cloud-native (?) Filesystems

BlockSize: 1.0 MiB, BigFileSize: 1.0 GiB, SmallFileSize: 128 KiB,
SmallFileCount: 100, NumThreads: 8

ITEM	VALUE	COST	
Write big file	378.94 MiB/s	21.62 s/file	
Read big file	368.60 MiB/s	22.22 s/file	
Write small file	146.9 files/s	54.44 ms/file	<- - raw NFS
Read small file	2181.8 files/s	3.67 ms/file	
Stat file	7702.0 files/s	1.04 ms/file	
Write big file	220.76 MiB/s	37.11 s/file	
Read big file	208.79 MiB/s	39.24 s/file	
Write small file	1148.0 files/s	6.97 ms/file	<- - Btrfs
Read small file	1256.3 files/s	6.37 ms/file	
Stat file	346667.5 files/s	0.02 ms/file	

Profiling

- Btrfs 元数据 + 小文件 (checksum 等) 并发读写速度几乎不可接受
- 如何量化? naive bpftrace:

```
#!/usr/bin/env bpftrace
```

```
BEGIN
```

```
{  
    printf("Tracing open file latency... Hit Ctrl-C to end.\n");  
}
```

```
tracepoint:syscalls:sys_enter_openat
```

```
{  
    if (strncmp(str(args->filename), "/mnt/btrfs/serve/archlinux", 32) == 0) {  
        printf("%s ", str(args->filename));  
        @start[tid] = nsecs;  
    }  
}
```

```
tracepoint:syscalls:sys_exit_sendfile64
/@start[tid]/
{
    $delta = (nsecs - @start[tid]) / 1000;
    printf("(%d bytes): %d ms\n", args->ret, $delta / 1000);
    @usec = hist($delta);
    @usec_stats = stats($delta);
    delete(@start[tid]);
}

END
{
    clear(@start);
}
```

Thread 20882 send 133681 bytes file 0 ms

Thread 20887 send 2579310 bytes file 0 ms

Thread 20889 send 900312 bytes file 9 ms

Thread 20887 send 900312 bytes file 6 ms

Take 2: cache come to rescue!

Any problem in computer science can be solved with **another level of indirection**.

— Roger Needham

- 镜像站特性无法避免大量随机小文件读写
- 考虑增加新的缓存层

内核树内的文件系统缓存机制

- FS-Cache: 主要面向网络存储层的读缓存, 不完全适合套娃架构
- bcache: 磁盘块缓存, 维护状况较差、文档缺失
- dm-cache: **DM/LVM** 逻辑盘缓存, 维护状况稳定

Take 2: cache come to rescue!

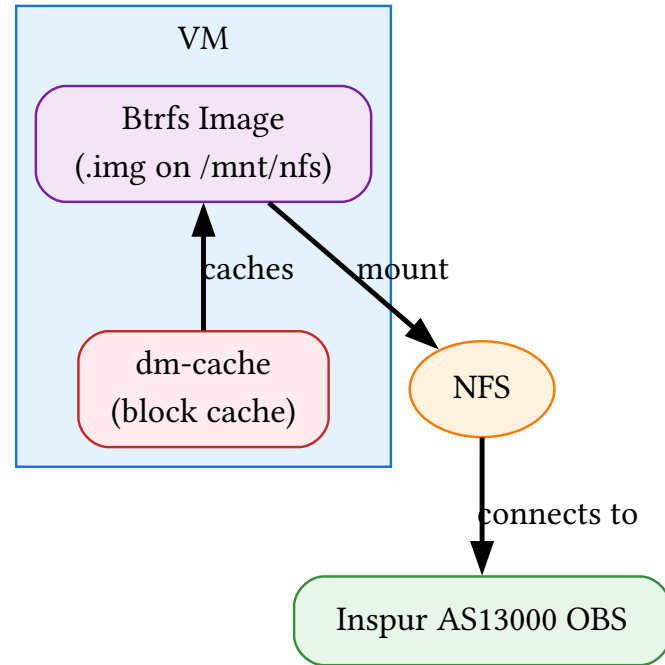
在 Btrfs 上配置 dm-cache 缓存

优点:

- 不影响已有文件系统配置, 可以保留数据
- 有成熟适配解决方案
- 能有效提升小文件读写速度

缺点:

- DM 子系统创建/修改配置逻辑复杂
- 任何缓存都依然无法解决 NFS 的协议缺陷及 Btrfs 元数据写放大的问题



Take 2: cache come to rescue!

[Unit]

Description=Setup dm-cache for mirrors on boot

Documentation=<https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/cache.html>

After=local-fs.target mnt-nfs.mount

After=dev-mapper-dm_cache-meta.device dev-mapper-dm_cache-cache.device

[Service]

Type=oneshot

RemainAfterExit=yes

ExecStart=/sbin/losetup /dev/loop0 /mnt/nfs/btrfs.img

ExecStart=/bin/bash -c '/sbin/dmsetup create mirrors --table "0 \$(/sbin/blockdev --getsz /mnt/nfs/btrfs.img) cache /dev/dm_cache/meta /dev/dm_cache/cache /dev/loop0 512 1 writethrough default 0" --readahead "256"'

ExecStart=/usr/bin/mount -o autodefrag "/dev/mapper/mirrors" /mnt/btrfs

ExecStop=/usr/bin/umount --all-targets "/mnt/btrfs"

ExecStop=/sbin/dmsetup resume "mirrors"

ExecStop=/sbin/blockdev --flushbufs "/dev/mapper/mirrors"

ExecStop=/sbin/dmsetup remove "mirrors"

ExecStop=/sbin/losetup -d /dev/loop0

[Install]

WantedBy=multi-user.target

What if we -NFS +S3?

- tunasync 架构要求底层 **POSIX** 存储，支持 S3 需要大改代码
- 市面上的 S3 FUSE mount 实现大多性能较差，不适合生产

JuiceFS: 开源云原生分布式文件系统

优点

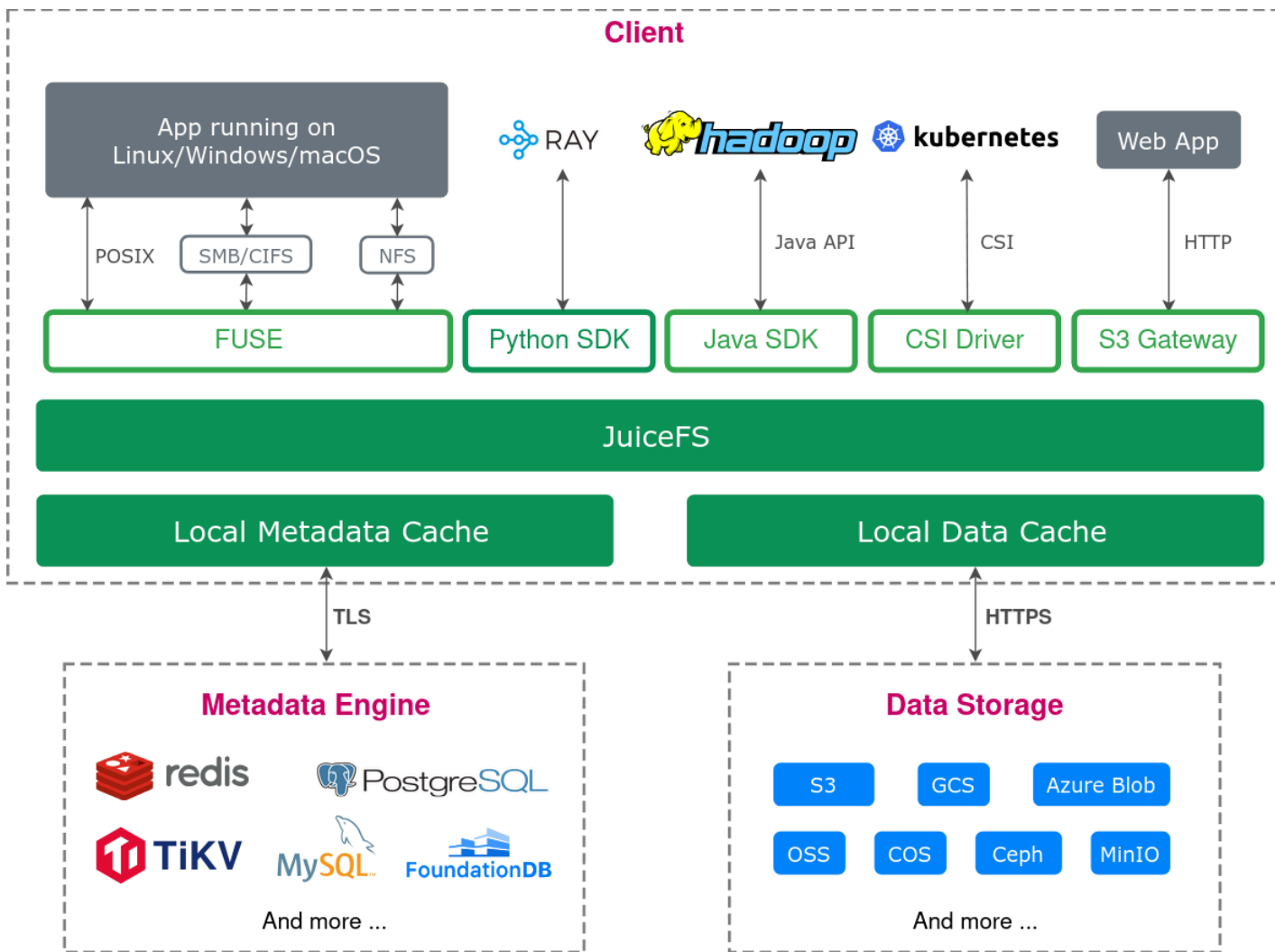
- 存储分离设计：
裸数据 - **OBS**；元数据 - **DB**
- 支持原生缓存、**FUSE mount**、**CoW**
- 支持大部分主流数据库
(PG、Redis)

缺点

- 使用 S3 作为块存储时，
无法快速准确地统计
实际用量
- 原生元数据备份功能
无法保证多线程一致性

Take 3: JuiceFS

Cloud-native (?) Filesystems



Take 3: JuiceFS

BlockSize: 1.0 MiB, BigFileSize: 1.0 GiB, SmallFileSize: 128 KiB,
SmallFileCount: 100, NumThreads: 8

ITEM	VALUE	COST	
Write big file	378.94 MiB/s	21.62 s/file	
Read big file	368.60 MiB/s	22.22 s/file	
Write small file	146.9 files/s	54.44 ms/file	<-- raw NFS
Read small file	2181.8 files/s	3.67 ms/file	
Stat file	7702.0 files/s	1.04 ms/file	
Write big file	348.95 MiB/s	23.48 s/file	
Read big file	324.52 MiB/s	25.24 s/file	
Write small file	417.6 files/s	19.16 ms/file	<-- JuiceFS w/ PG
Read small file	5964.7 files/s	1.34 ms/file	
Stat file	17426.9 files/s	0.46 ms/file	

事故时间线

- 1 19:54 juicefs dump --threads 10 导出日常备份
- +0 12:23 **PG WAL** 写满磁盘 → 错误地移除 WAL 并重置数据库
- +0 13:02 juicefs load 恢复备份 → jfs_node inode 唯一约束冲突
- +0 13:43 阅读源码, 人工修正 jfs_counter.next{Inode,Chunk} 相关元数据
- +0 14:51 修复后执行 fsck --path / --repair 通过, 无报错后临时上线
- +0 22:53 日志出现大量 fail to read sliceId, 确认数据完整性损坏
- +1 03:29 执行 fsck --path, 期望完整检查修复块存储数据
- +1 14:19 确认原生 fsck 实现有严重的性能问题, 决定修改源码
- +1 21:23 修改后脚本删除 ~300 万损坏 Inode, 服务重新上线
- +2 22:58 juicefs gc --delete 回收 1/3 无效存储, 重启同步

根因分析

1. 备份不一致

juicefs dump 在关系型 SQL 引擎下多线程运行无法保证事务一致性

- 1). 不同线程并行读写不同时间点的元数据
- 2). 同时执行了 dump 命令进行备份
- 3). 导出的备份包含交叉的 inode/chunk 引用、计数，导致恢复失败
- 4). 由于在备份的同时仍有大量数据写入，导致数据完整性损坏

- 更多细节@余光的部落格

2. fsck 行为不符合预期

- 文档状态检查的行为描述模糊
- fsck --path / --repair 仅检查部分元数据属性
- fsck 尽管检查了数据块完整性，但由于频繁请求对象存储和元数据路径，有严重的性能瓶颈

故障指标

- 总数据规模：~**20TB**，Inode **1100 万**
- 损坏范围：~**300 万 Inode** (文件/目录)
- 影响范围： **1/3 对象** 存储
- 恢复耗时： **1.5 天**

教训

- **永远不要依赖** 上层应用作为主要灾备手段
- **优先保证** 底层引擎 (**DB**) 高可用！
- Talk is cheap. ~~Show me~~ **Read the code** 而不是依赖常识和假设
- **可观测性建设** 能更早/好地发现问题

Observability

How can we do better?

- 保存所有服务的错误日志
- 设计更完善的记录指标
- 及时监测前两者推送异常预警

On what?

节点 (Node)

CPU, Memory, Disk, Net ...

服务 (Service)

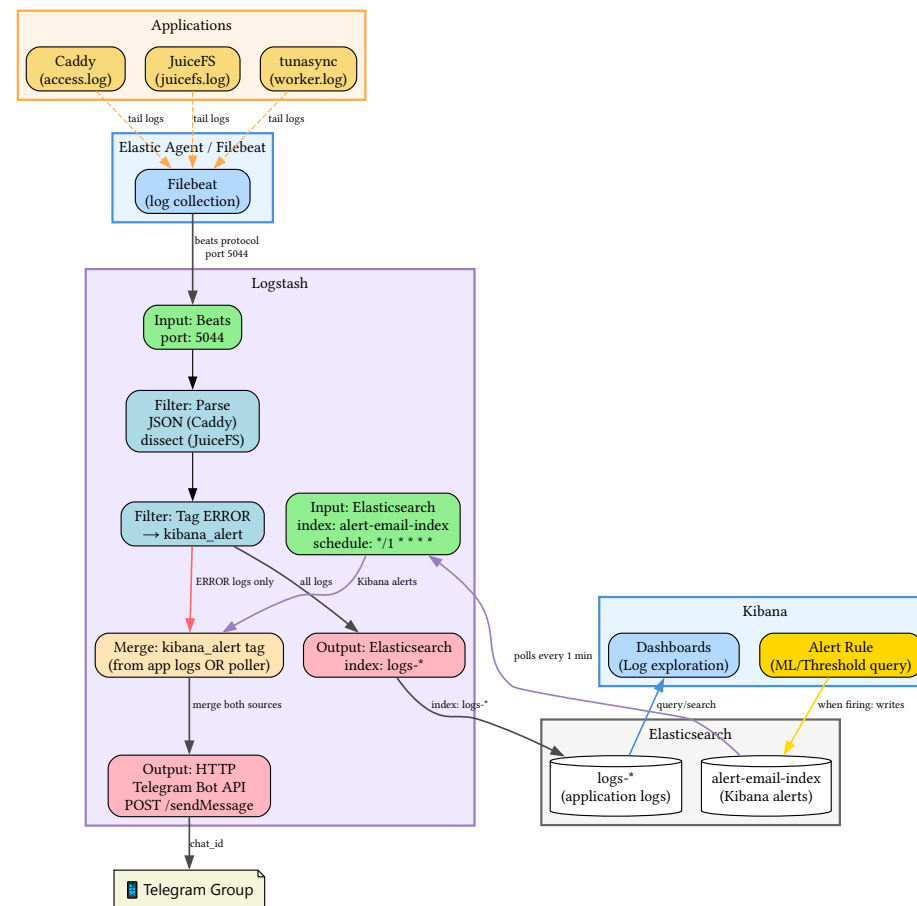
Caddy, tunasync, JuiceFS, Postgres ...

Elastic Kit

架构组件

- **Elastic Agent:** 系统指标采集 (CPU/内存/磁盘/网络/进程)
- **Filebeat:** 应用日志采集 (Caddy/JuiceFS/tunasync)
- **Logstash:** 日志处理 + 告警推送 (Telegram)
- **Elasticsearch:** 日志数据库
- **Kibana:** 可视化 + 告警规则

Cursed pipeline



Bonus: tunasync 监控

需求：期望定时监测 tunasync 镜像的同步状态，实现超时报警

要求监控应用必须：

- 能正确解析时间格式/时间戳
- 能正确处理计数

显然，ELK 自带的轻量 HTTP 监控器无法满足我们的需求；
必须使用 **Synthetic monitor**：

... 而其本质是一个无头
Playwright/Chromium。

```
journey("tunasync status check", ({ page,
params }) => {
  monitor.use({
    id: "tunasync-monitor",
    schedule: 240,
  });
  step("check outdated mirrors", async () =>
  {
    const resp = await page.goto(params.url);
    expect(resp.status()).toBe(200);

    const json = await resp.json();
    // monitor failed logic ...
  });
});
```

架构组件

- **Prometheus**: 时序数据库
- **Exporter**: 采集各类指标
- **Grafana**: 可视化 + 告警规则

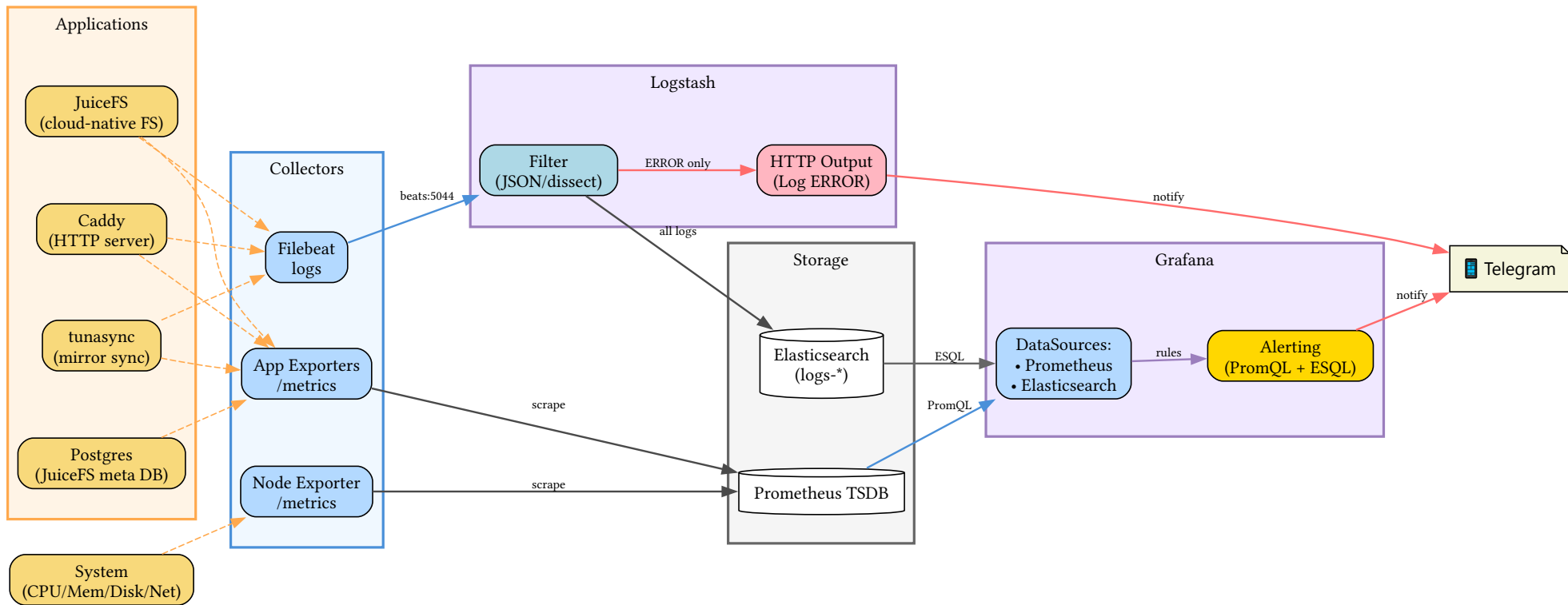
采集指标

- **node_exporter** (9101): 主机指标
- **tunasync** (9101): 镜像同步状态
- **juicefs** (9101): 文件系统指标
- **caddy** (9101): HTTP 服务指标
- **postgres** (9187): 数据库指标

日志怎么办?

- Prometheus 本质是时序的
- Elasticsearch 本质是文本的
- Grafana v13.0: ES|QL query!
- 坚持使用成熟的 ES 存储日志

```
FROM `logs-generic-default`  
| WHERE event.dataset ==  
"caddy.access"  
| STATS count = COUNT() BY  
`request.headers.X-Forwarded-For`  
| SORT count DESC  
| LIMIT 10
```



Bonus: tunasync 监控

需求：期望定时监测 tunasync 镜像的同步状态，实现超时报警

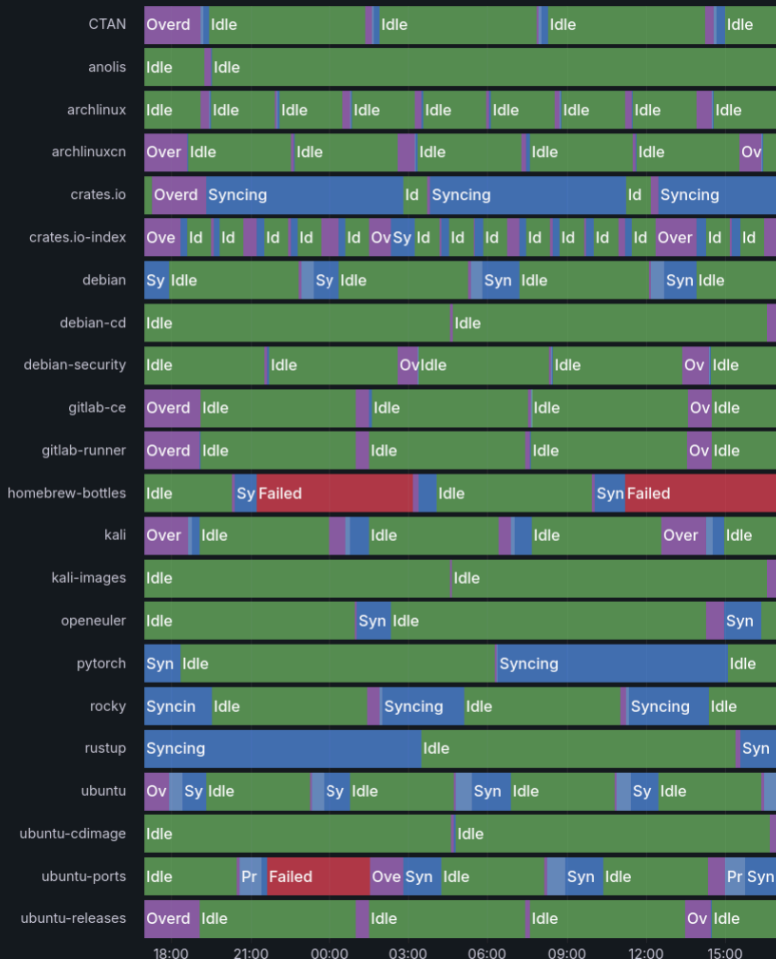
实现：直接在 tunasync-manager 应用测添加监控指标

```
type prometheusCollector struct {  
    registry *prometheus.Registry  
    mu       sync.RWMutex  
    manager  *Manager  
  
    // Worker metrics  
    workerRegistered      prometheus.Gauge  
    workerLastOnlineTime  *prometheus.GaugeVec  
    // Mirror job metrics  
    mirrorJobStatus       *prometheus.GaugeVec  
    mirrorLastUpdate      *prometheus.GaugeVec  
  
    mirrorLastStarted      *prometheus.GaugeVec  
    mirrorLastEnded        *prometheus.GaugeVec  
    mirrorNextSchedule     *prometheus.GaugeVec  
    mirrorIsMaster         *prometheus.GaugeVec  
    mirrorSizeBytes        *prometheus.GaugeVec  
    mirrorSyncDuration     *prometheus.GaugeVec  
  
    // Aggregated metrics  
    totalMirrors      prometheus.Gauge  
    mirrorsByStatus  *prometheus.GaugeVec  
}
```

Bonus++: 镜像过期检测

需求: 如何发现「已结束但下次调度已过期」的迟到任务?

```
# Overdue (7) ONLY applies when job_status is success (2)
- record: tunasync_mirror_job_status_eff
  expr: |
    ((7 *
      (tunasync_mirror_last_ended_timestamp >= bool
tunasync_mirror_last_started_timestamp
      and on(mirror_name, worker_id)
        (tunasync_mirror_next_schedule_timestamp < time()
          and tunasync_mirror_next_schedule_timestamp > 0)))
      and on(mirror_name, worker_id)
        (tunasync_mirror_job_status == 2))
    or on(mirror_name, worker_id)
    max by(mirror_name, worker_id, instance, job)(tunasync_mirror_job_status)
```



The Reality

- Logs + Metrics aren't enough: 透过现象看不到本质
- LLM 的能力在于对较固定场景的初步分析
- 排障脚本 → Webhook → Skills, 本质是相似的流程/经验总结
- 服务端难以完全信任 Agent 进行自主决策

Use-cases

- 让 AI 做繁琐的小事: 压缩总结日志、找报错
- 语义的、友好的、更方便的用户界面
- TimesFM: 更好的异常检测?
- AgentSight: 部署只读的 Agent, 使用 SELinux 等机制管控权限?

Thanks for watching :<3
