

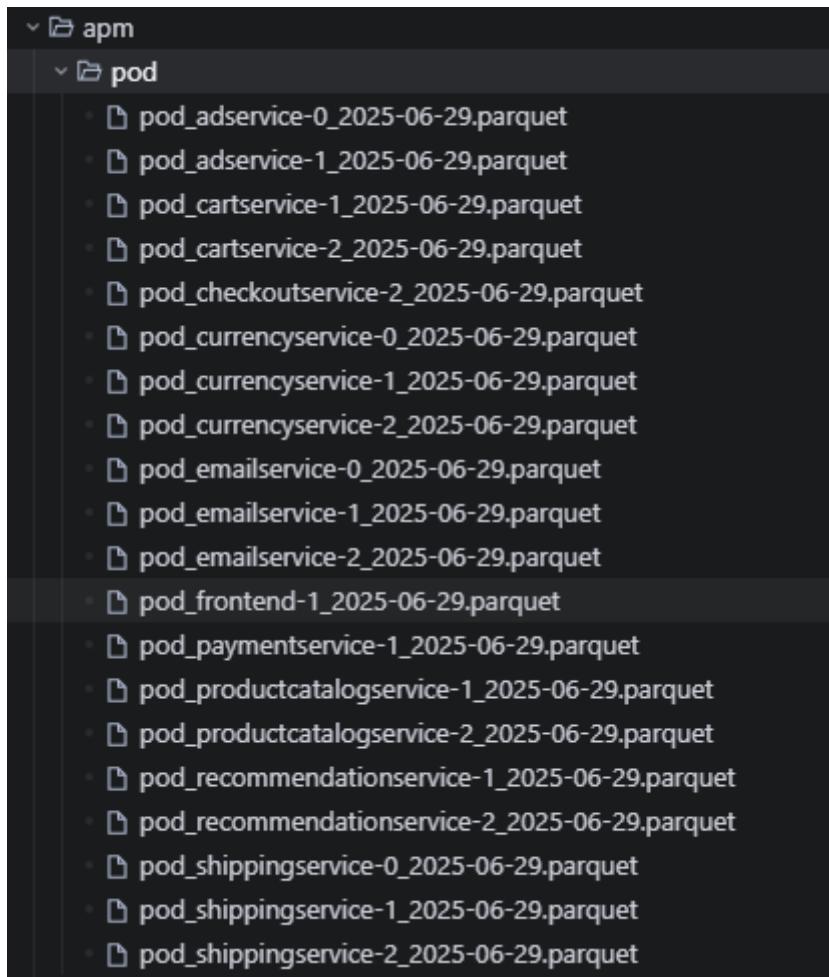
11.10 多智能体代码实现 副本

Metric Refinement (MicroRCA-Agent的方法)

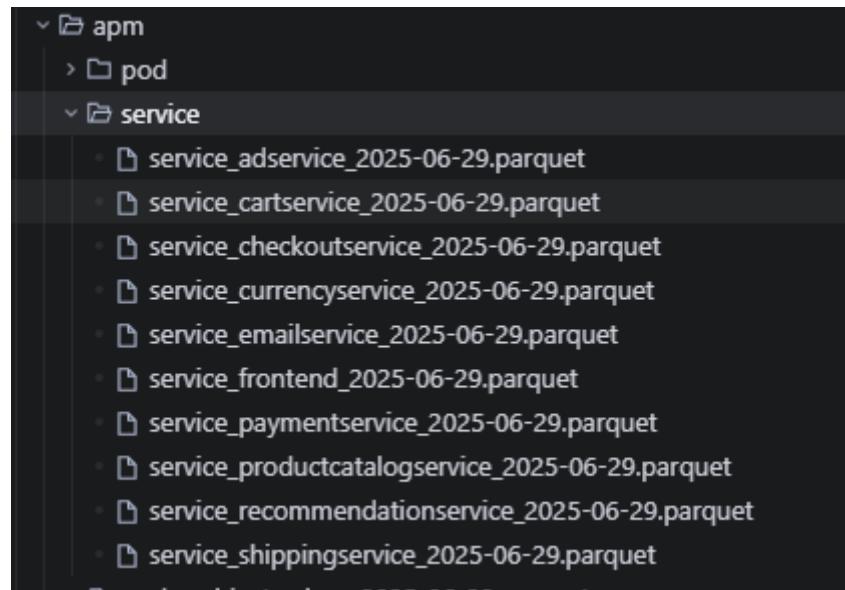
metric目录结构：

apm/ #保存的是应用程序性能监控相关的指标数据。

pod/ #以 pod 为单位聚合的指标数据，以 pod_adservice-0_2025-06-29.parquet 为例，这个文件包含了名为 adservice-0 在6月29日一整天的所有 APM 指标，其中作者认为的关键指标有 key_metrics = ['client_error_ratio', 'error_ratio', 'request', 'response', 'rrt', 'server_error_ratio', 'timeout']。

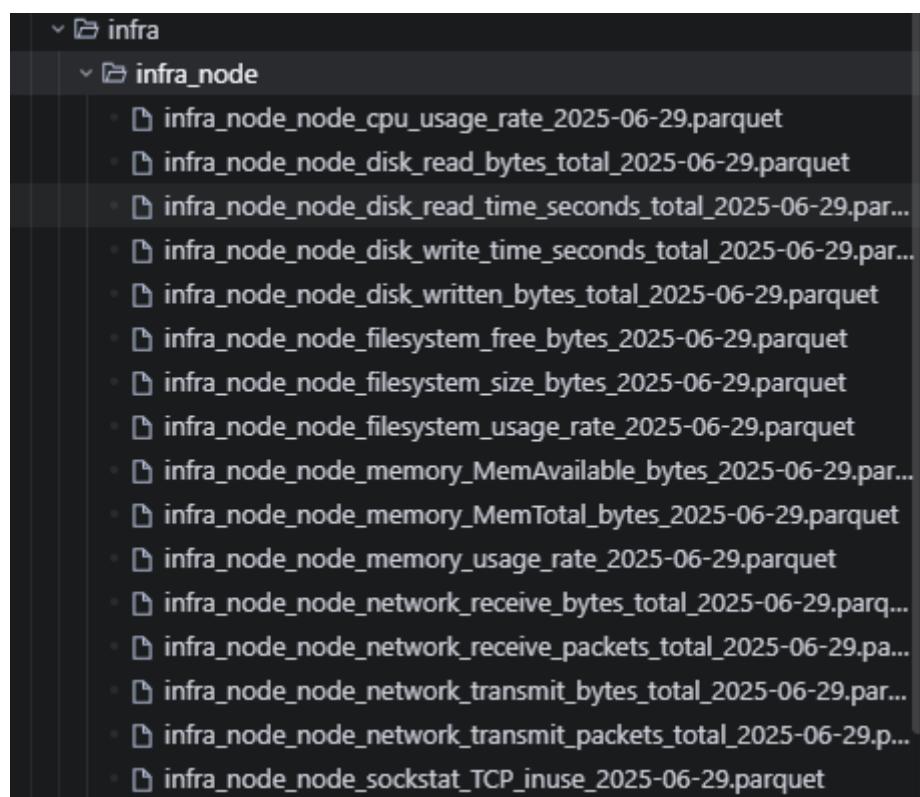


service/ #以 service 为单位聚合的指标数据，以 service_adservice_2025-06-29 为例，这个文件包含了名为 adservice 在6月29日一整天的聚合指标，代表了 adservice 的总体性能。



infra/ #保存的是基础设施相关的监控指标数据。

infra_node/ 以虚拟机节点 (Node) 为单位聚合的指标数据，反映了整个节点的资源使用情况



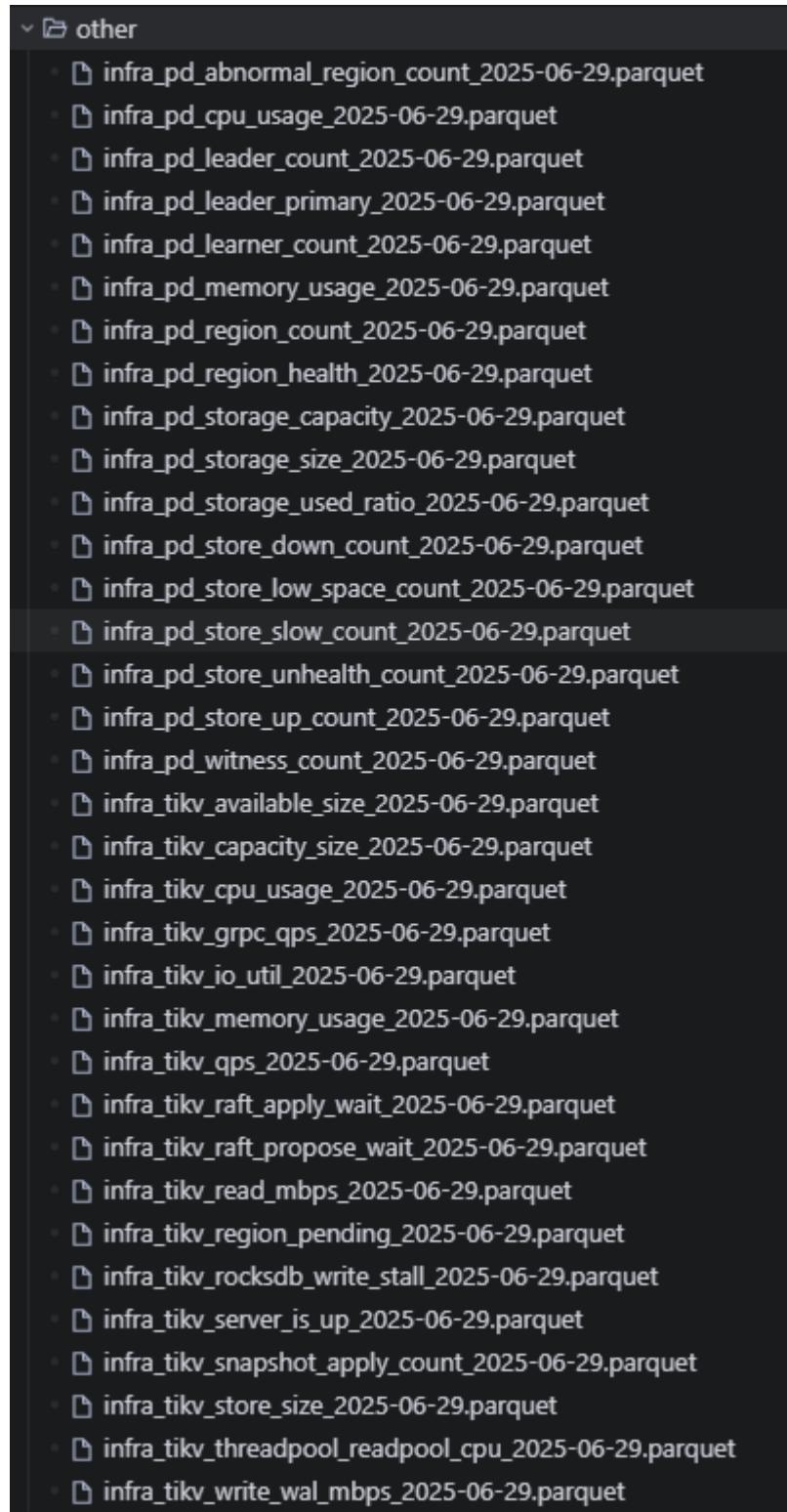
infra_pod/ #以 Pod 为单位聚合的基础设施指标。它反映了每个 Pod 消耗的底层资源。

└ infra
└ infra_node
└ infra_pod
└ infra_pod_pod_cpu_usage_2025-06-29.parquet
└ infra_pod_pod_fs_reads_bytes_2025-06-29.parquet
└ infra_pod_pod_fs_writes_bytes_2025-06-29.parquet
└ infra_pod_pod_memory_working_set_bytes_2025-06-29.parquet
└ infra_pod_pod_network_receive_bytes_2025-06-29.parquet
└ infra_pod_pod_network_receive_packets_2025-06-29.parquet
└ infra_pod_pod_network_transmit_bytes_2025-06-29.parquet
└ infra_pod_pod_network_transmit_packets_2025-06-29.parquet
└ infra_pod_pod_processes_2025-06-29.parquet

infra_tidb/ #专门针对 TiDB 数据库集群的基础设施和性能指标。TiDB 是一个分布式数据库，有自己独特的监控指标体系。

└ infra_tidb
└ infra_tidb_block_cache_size_2025-06-29.parquet
└ infra_tidb_connection_count_2025-06-29.parquet
└ infra_tidb_cpu_usage_2025-06-29.parquet
└ infra_tidb_duration_95th_2025-06-29.parquet
└ infra_tidb_duration_99th_2025-06-29.parquet
└ infra_tidb_duration_avg_2025-06-29.parquet
└ infra_tidb_failed_query_ops_2025-06-29.parquet
└ infra_tidb_memory_usage_2025-06-29.parquet
└ infra_tidb_qps_2025-06-29.parquet
└ infra_tidb_server_is_up_2025-06-29.parquet
└ infra_tidb_slow_query_2025-06-29.parquet
└ infra_tidb_top_sql_cpu_2025-06-29.parquet
└ infra_tidb_transaction_retry_2025-06-29.parquet
└ infra_tidb_uptime_2025-06-29.parquet

other/ #专门针对 TiDB 的两大核心组件：PD (Placement Driver) 和 TiKV 的性能指标数据。PD (Placement Driver) 是 TiDB 集群的元数据管理中心，负责存储数据路由信息 (Region 在哪个 TiKV 节点上)、调度和负载均衡。它就像是整个集群的大脑。TiKV 是 TiDB 的分布式键值存储引擎，负责真正地存储数据。数据被切分成多个 Region，并以 Raft 协议保证多副本之间的数据一致性。它就是集群的“肌肉”，负责数据的读写和持久化。



1. 分析普通微服务

用到文件有 `apm/pod/*`。按 `Service → Pod → Metrics → {normal_period, fault_period}` 结构组织分析结果，作者认为的关键指标如下。

代码块

```
1 key_metrics = ['client_error_ratio', 'error_ratio', 'request',
2                 'response', 'rrt', 'server_error_ratio', 'timeout']
```

1.1.根据异常描述找到异常起始时间与结束时间，将其时间段定义为 fault_period。然后找到故障前后的正常时间段，具体为上一个故障结束后10分钟到当前故障开始，以及当前故障结束后10分钟到下一个故障开始，将其定义为 normal_period。

1.2.对异常时间段与正常时间段的数据进行统计。收集该时间段的数据，遍历每个 pod，在每个 pod 下遍历每个关键指标，去除该指标中最小两个和最大两个（仅针对正常时间段的数据），生成描述性统计信息，有样本数量、平均值、标准差、最小值、最大值、第一四分位数、中位数、第三四分位数、P95、P99、非零比例。

2.分析TiDB服务

用到的文件有 infra_pd_*, infra_tikv_*, infra_tidb_*。按 Service → Metric → {normal_periods, fault_period} 结构组织分析结果。作者认为的关键指标有这么一些，每一个指标对应一个parquet文件。

代码块

```
1  'tidb-tidb': [
2      'failed_query_ops',    # 失败请求数 - 错误率指标
3      'duration_99th',     # 99分位请求延迟 - 关键性能指标
4      'connection_count',   # 连接数 - 负载指标
5      'server_is_up',       # 服务存活节点数 - 可用性指标
6      'cpu_usage',          # CPU使用率 - 资源饱和度
7      'memory_usage'        # 内存使用量 - 资源使用
8  ],
9  'tidb-pd': [
10     'store_up_count',    # 健康Store数量 - 集群健康度
11     'store_down_count',  # Down Store数量 - 故障指标
12     'store_unhealth_count', # Unhealth Store数量 - 异常指标
13     'storage_used_ratio', # 已用容量比 - 容量指标
14     'cpu_usage',          # CPU使用率 - 资源使用
15     'memory_usage'        # 内存使用量 - 资源使用
16  ],
17  'tidb-tikv': [
18      'cpu_usage',          # CPU使用率 - 资源使用
19      'memory_usage',       # 内存使用量 - 资源使用
20      'server_is_up',       # 服务存活节点数 - 可用性
21      'available_size',     # 可用存储容量 - 容量预警
22      'raft_propose_wait',  # RaftPropose等待延迟P99 - 性能指标
23      'raft_apply_wait',    # RaftApply等待延迟P99 - 性能指标
24      'rocksdb_write_stall' # RocksDB写阻塞次数 - 关键异常指标
25  ]
```

2.1.分析策略不变，首先找到 fault_period 与 normal_period，然后生成指标的统计信息。

3. 创建合并的Service+TiDB prompt并调用LLM（第1次调用）

合并普通微服务与TiDB服务的统计信息，过滤掉一些异常期间表现正常的指标，过滤机制如下。

代码块

```
1 # 计算对称比率
2 p50_symmetric_ratio = abs(fault_p50 - normal_p50) / (
3     (fault_p50 + normal_p50) / 2 + 1e-9
4 )
5 p99_symmetric_ratio = abs(fault_p99 - normal_p99) / (
6     (fault_p99 + normal_p99) / 2 + 1e-9
7 )
8 # 过滤阈值（建议 0.05 根据需求调整，过大将导致过多信息被视为正常）
9 if p50_symmetric_ratio < 0.05 and p99_symmetric_ratio < 0.05:
10     continue
```

然后将合并结果嵌入在 prompt 中调用大模型描述观察到的现象。

代码块

```
1 prompt =
2 """
3 请根据提供的APM（应用性能监控）指标数据和TiDB分布式数据库指标数据，描述所有服务在正常期间
4 和故障期间的业务服务性能表现差异现象。
5 .....
6 请基于APM业务监控数据和TiDB数据库监控数据客观描述观察到的现象，控制在2000字以内，为后续综
7 合分析提供简洁有效的现象总结。
8 """
```

4. 分析Node级别数据

用到的文件有 infra_pod/*, infra_node/*。

对于 infra_pod/* 文件的指标分析，按 Pod → Metrics → {normal_period, fault_period} 结构组织分析结果，作者认为的关键指标如下，每一个指标对应一个parquet文件。

代码块

```
1 target_metrics = [
2     'pod_cpu_usage', 'pod_fs_reads_bytes', 'pod_fs_writes_bytes',
3     'pod_memory_working_set_bytes', 'pod_network_receive_bytes',
4     'pod_network_receive_packets', 'pod_network_transmit_bytes',
5     'pod_network_transmit_packets', 'pod_processes'
6 ]
```

对于 infra_node/* 文件的指标分析，按 Node → Metrics → {normal_period, fault_period} 结构组织分析结果，作者认为的关键指标如下，每一个指标对应一个parquet文件。

代码块

```
1 target_metrics = ['node_cpu_usage_rate',
2                     'node_disk_read_bytes_total',
3                     'node_disk_read_time_seconds_total',
4                     'node_disk_write_time_seconds_total',
5                     'node_disk_written_bytes_total',
6                     'node_filesystem_free_bytes',
7                     'node_filesystem_usage_rate',
8                     'node_filesystem_usage_rate',
9                     'node_memory_MemAvailable_bytes',
10                    'node_memory_MemTotal_bytes',
11                    'node_memory_usage_rate',
12                    'node_network_receive_bytes_total',
13                    'node_network_receive_packets_total',
14                    'node_network_transmit_bytes_total',
15                    'node_network_transmit_packets_total',
16                    'node_sockstat_TCP_inuse', ]
```

4.1. 分析策略不变，同样是先找到 fault_period 与 normal_period，然后生成指标的描述性信息。但是在分析 infra_pod/* 文件的指标数据时，作者用到了一个过滤机制。

代码块

```
1 if normal_desc is not None and fault_desc is not None:#过滤掉变化倍数在 0.95 到
2     1.05 之间的指标
3     normal_mean = normal_desc['mean']
4     fault_mean = fault_desc['mean']
5     epsilon = 1e-9 # 极小数，防止除零
6     ratio = (fault_mean + epsilon) / (normal_mean + epsilon)
7
8     if 0.95 <= ratio <= 1.05:
9         print(f"    指标 {metric_name} 变化倍数 {ratio:.2f} 在 0.95~1.05 之间，跳
  过保存")
10        continue
```

5. 创建包含Service+TiDB分析结果的Node prompt并调用LLM（第2次调用）

获取当天 pod 部署信息，{node_name: [pod1, pod2, ...]}。合并上述分析结果，将 pod 详细指标合并在该 node 下，过滤掉一些异常期间表现正常的指标，过滤机制与 3. 一致。

代码块

```
1 请基于提供的Service级别分析结果、Pod部署信息和基础设施监控指标数据，进行全局的现象总结分析。  
2 ## Service级别分析结果回顾  
3 {service_analysis_result}  
4 .....  
5 请基于Service分析、Pod部署信息和基础设施监控数据，提供全局的综合现象总结，控制在2000字以内。  
6 """
```

Metric Refinement（我的想法）

分层思想

作者没有用到 `apm/service/*` 下的文件，这个目录存放的是以 Service 为单位聚合的指标数据。

作者在分析微服务指标数据时直接分析 `apm/pod/*` 下的所有文件，这种做法虽然合理但是计算量大，存储每个 pod 的指标分析结果导致 token 增加。可以先分析 `apm/service/*` 下的 service 文件，发现某个 service 异常再找到对应的 `apm/pod/*` 文件，分析该 service 下的 pod，减少计算量和 token 消耗。

代码块

```
1 service → pod → {pod_metrics → {normal_period, fault_period}}
```

精细化调用agent

将所有统计信息合并在一起输入给大模型会产生极大的上下文窗口，可以在每一次生成指标的统计信息时调用一次agent，以确认该指标是否偏离了正常值，需不需要保留。