智能座舱端云一体性能与稳定性平台 (Polaris 1.0) 系统设计文档

版本信息

序号	版本	修订内容	状态	修订人	日期
1	0.1	First draft		操权力	2025/12/9

文档目的

本文档旨在全面定义 智能座舱端云一体性能与稳定性平台 (代号 Polaris 1.0) 的系统架构、功能需求及实施路径。本文档将服务于以下核心场景：

- 管理层决策：清晰阐述项目背景、痛点、ROI（投入产出比）及资源需求，作为立项审批与资源调度的依据。
- 跨部门协同：作为座舱平台部与车云平台部沟通数据协议、接口规范及边界划分的“蓝本”，确保端云技术方案的一致性。
- 工程落地指导：作为项目启动后的核心输入，指导研发团队进行端侧 Agent 开发、埋点设计及测试验收。

背景与问题定义

背景

当前智能座舱的数据建设存在数据维度失衡与底层感知缺失的问题，具体表现在以下三个方面：

- 应用质量量化手段缺失：目前虽已具备应用层的业务埋点能力（如 PV/UV、页面点击流），能支撑产品运营分析；但对于应用技术质量（如 Crash率、ANR率、错误日志）及 核心性能指标（如启动耗时、页面响应延迟）尚缺乏系统性的监控与度量手段，导致软件交付质量缺乏客观数据支撑。
- 平台侧缺乏云端可观测性：作为座舱底座的平台研发部门，目前缺乏专属的云端观测平台。对于线上车辆的系统级健康度（如 SystemServer 重启、关键服务存活、资源水位），研发团队缺乏实时获取线上运行时状态的能力，往往只能在故障发生后进行被动回溯。
- 系统稳定性保障体系亟待构建：随着智能座舱软件规模与复杂度的提升，单纯依赖线下测试已难以覆盖所有边缘场景。为了保障用户体验，亟需构建一套严谨的、标准化的端云一体性能与稳定性监控平台，实现对线上真实运行质量的精准监测与闭环管理。

当前痛点

痛点	描述	业务影响
跨端故障排查成本较高	当前缺乏跨端（Android-Linux-MCU）的自动化关联数据， 面对复杂的跨域交互问题， 排查过程往往需要人工拼接多端日志。	**研发效率受限**：故障定位往往需要多方协同与多次排查， 拉长了问题的解决周期。
性能量化数据覆盖不足	现有的性能评估主要依赖线下测试或有限样本， 缺乏全量用户场景下的启动速度、流畅度等自动化量化数据。	**版本评价受限**： 难以精确捕捉版本迭代中的细微性能波动， 线上实际体验的评估数据不够丰满。
偶发异常现场回溯困难	对于线上偶发的非必现问题，目前主要依赖事后尝试复现， 缺乏异常发生瞬间的自动“快照”捕获机制。	**闭环周期较长**：部分偶发性稳定性问题（如随机黑屏、 卡顿）因缺乏现场数据支持，难以快速彻底根除。
资源效能优化缺乏支撑	缺乏进程级的 CPU、内存、IO 历史趋势画像， 在进行精细化资源管控时缺乏足够的数据颗粒度。	**成本优化受限**： 硬件资源规划倾向于保守策略以保障稳定性，BOM 成本的进一步精细化挖掘存在困难。

目标与范围

项目目标

本项目旨在基于 “端侧深度探针 + 云端聚合分析 + 全链路追踪” 的技术理念，构建 Polaris 1.0 端云一体化平台，实现以下三个核心目标：

1. 全链路可观测：打破 Android、Linux Host、MCU 的数据孤岛，建立统一的 全局事件标准 (Global Event ID)，将分散在不同系统的故障与状态数据聚合至同一平台，实现跨端调用的追踪，为后续的可视化链路分析奠定数据基础。

2. 故障现场自动聚合与关联：突破现有“日志碎片化”及“事后拉取不全”的局限。建立 “事件驱动”的现场快照机制，在异常发生瞬间，自动聚合与该事件强相关的全维度上下文信息（如 Trace、系统 Log、进程状态等）并生成 完整的故障证据包。这不仅实现了 Event 与 Log 的精准索引，更确保了现场信息的完整性，彻底解决因关键日志缺失导致无法定位的难题。
3. 数据驱动治理：建立系统级的性能与稳定性基线（Baseline），通过量化数据驱动版本质量验收与硬件资源优化，将质量管理从“定性”转向“定量”。

核心 KPI 指标

维度	指标名称	目标值 (示例)	说明
质量	**严重故障主动发现率**	**> 90%**	在用户报修前，通过平台主动捕获并预警系统级崩溃与卡顿。
效率	**日志精准命中率**	**100%**	每一个上报的严重异常事件，都能直接下载到对应的、正确的 Log 文件，无需人工筛选。
复现	**致命问题现场捕获率**	**> 80%**	针对 Crash/Watchdog 等致命问题，确保有对应的 Trace/Log 可供分析。
成本	**资源优化场景产出**	**TOP 5/季度**	每季度识别并输出 5 个高资源消耗（CPU/内存）场景。

项目范围

范围内

1. **端侧全栈感知体系**：
   - **Android 深度探针**：构建系统级监控服务 `PolarisAgentService` ，实现对应用生命周期、核心服务状态、底层资源（LMK/IO/Binder）的**全维度深度监听**。
   - **Linux/MCU 异构覆盖**：建设 Linux Host 侧的**系统健康守护进程**，负责关键服务（Service）存活检测与系统指标采集；适配 MCU 遥测协议，实现异构芯片间的故障透传。
   - **边缘智能处理**：在端侧实现数据的**预处理与清洗**，包含事件聚合、流控防爆、日志现场的智能截取与压缩，减轻车云带宽压力。
   - **标准化基础设施**：建立《全局事件注册表》及自动化工具链，统一多端的数据定义与协议标准。
2. **云端分析能力需求**:
   - **元数据管理能力**：要求云端支持同步《全局事件注册表》，实现对上报事件的自动化解析、分类与标签化管理。
   - **自动化关联引擎**：要求云端具备**“事件-日志”自动匹配能力**，将结构化的 Event 数据与非结构化的 Log 文件（基于索引）在存储层自动关联，形成完整的故障证据包。
   - **趋势与模式识别**：要求云端支持基于时间窗口的聚合计算，能够识别异常爆发（Spike）趋势及性能指标（CPU/内存）的长期演进趋势。
3. **可视化与运营平台**
   - **数字化质量驾驶舱**：建设多维度的质量仪表盘（Dashboard），支持按版本、车型、时间段下钻分析千车故障率、性能基线达标率。
   - **智能排查工作台**：提供“一站式”问题分析界面，支持通过 EventID/TraceID 检索故障，直接浏览关联的日志、堆栈及设备状态，支持远程诊断指令的下发与结果展示。

范围外

1. **可视化的全链路拓扑分析**：1.0 阶段聚焦于跨端链路数据的 标准化采集与逻辑串联，优先夯实数据底座能力；全链路图形化的调用链拓扑展示规划在后续版本迭代中实现。
2. **业务代码修复**：Polaris 平台负责精准“定位”并“指派”问题，**不负责** 具体业务 APP 内部的代码逻辑修复。
3. **交互体验设计**：本项目专注于性能数据的量化，**不包含** HMI 界面（UI/UE）的主观交互设计与优化。

业务流程与核心场景

角色定义

角色	职责描述	关注点
研发工程师	接收告警，分析堆栈与日志，修复 Bug；针对疑难客诉问题， 远程下发特定诊断指令	故障堆栈的完整性，日志关联的准确性， 是否需要补充更多运行时信息。
质量工程师	配置告警阈值，监控线上大盘水位，识别版本质量风险	故障率趋势是否劣化，性能指标是否符合预期， 流量消耗是否异常。
产品经理	查看应用活跃度与性能体验趋势	核心功能的响应速度趋势，用户使用过程中的卡顿频率。

核心作业流程图

1. 故障主动发现闭环流程

> *描述从异常发生到研发接入的处理路径*

- **捕获 (Capture)**: Polaris Agent 监听到异常（如 ANR），记录运行时状态，抓取 Trace/Logcat，并生成唯一 EventID。
- **处理 (Process)**: 端侧进行流量控制检查，通过 `logf` 索引将 Event 与 Log 文件进行逻辑组合。
- **上报 (Report)**: Event 数据实时上报，大文件 Log 在 WiFi/空闲时段异步上传（支持云端按需拉取）。
- **通知 (Notify)**: 云端检测到异常数据超过阈值（例如某版本 Crash 率上升），向 **责任模块负责人** 发送通知。
- **分析 (Analyze)**: 研发工程师查看通知，进入平台查看关联的上下文数据，确认问题根因并修复。

2. 疑难问题排查流程

> *描述针对复杂客诉或非必现问题的处理路径*

- **检索**: 研发工程师在平台输入车辆 VIN 码或 EventID 检索相关记录。
- **查看**: 系统展示该事件的发生时间、设备信息、以及**已自动关联**的 Log 文件下载链接。
- **诊断**: 针对区域技术支持无法处理的复杂客诉，若现有日志不足以定位，**研发工程师** 通过控制台下发 `Shell` 诊断指令，端侧执行后回传结果，以获取更深度的运行时信息。

典型用户故事

场景一：风险预警

> **背景**: 某车型灰度推送 v1.5 OTA 版本。
> **事件**: 上线 24 小时内，Polaris 平台监测到 `GVM_SYS_STORAGE_LOW` （磁盘空间不足）事件在特定批次车辆上的上报量呈**异常上升趋势**。
> **行动**:
>
> 1. 平台自动触发 **风险预警**，即时通知研发负责人。
> 2. 研发工程师通过平台获取存储分布数据，精准定位到某应用私有目录占用空间急剧膨胀。
> 3. **分析**: 结合自动关联采样的 Log，确认该应用在特定异常分支下陷入**数据库高频重复写入**死循环。
>   **结果**: 研发团队在磁盘被完全耗尽导致系统挂死（System Hang）前，紧急输出修复补丁并推送 OTA，成功拦截了批量重大事故。

场景二：稳定性治理

> **背景**: 某应用发布 v2.0 灰度版本。
> **事件**: 灰度发布期间，平台监测到应用出现**偶发性** `GVM_APP_ANR` （无响应）告警，且线下测试难以复现。
> **行动**:
>
> 1. 研发工程师点击告警详情，查看聚合后的故障样本。
> 2. 系统已通过 `logf` 字段自动关联了故障时刻的 `traces.txt` 以及系统侧 `perflog` (性能日志)。
> 3. **分析**: 工程师通过 Trace 文件发现应用主线程阻塞在 Binder IPC 调用中；进一步联合分析 `perflog` ，定位到是对端 Service 在高并发场景下因锁竞争导致处理耗时过长，拖累了客户端。
>   **结果**: 确认根因为**服务端卡顿**。研发工程师针对服务端逻辑进行异步化优化，彻底解决了这一隐蔽的跨进程阻塞问题。

场景三：性能监控

> **背景**: 某版本上线后，产品经理关注核心应用在复杂交互场景下的滑动流畅度。
> **事件**: Polaris 仪表盘显示 `GVM_APP_JANK` (掉帧/卡顿) 指标在特定列表滑动场景下出现劣化趋势。
> **行动**:
>
> 1. 系统展示了掉帧率与主线程负载的关联曲线。
> 2. **发现**: 在卡顿发生的时间段内，主线程 MessageQueue 待处理消息数量显著激增。
> 3. **分析**: 研发工程师通过分析采集到的 `Looper` 统计数据，发现是一次性加载过多列表项导致并在主线程频繁 Post UI 刷新消息，引发**主线程消息队列积压**，从而阻塞了渲染信号（Vsync）的处理。
>   **结果**: 研发工程师引入消息合并与节流机制（Throttling），消除了主线程拥堵，恢复了滑动流畅性。

场景四：远程指令下发

> **背景**: 用户反馈方控按键（下一曲）失效，或错误地控制了不显示在屏幕上的后台音乐应用，常规 Logcat 无法体现系统内部的分发逻辑。
> **行动**:

> 1. 研发工程师怀疑是 MediaSession 焦点抢占或状态同步异常。
> 2. 工程师通过 Polaris 控制台，向目标车辆下发 dumpsys media_session 指令。
> 3. 分析: 回传的诊断结果显示，Media button session 仍被后台应用 com.reachauto.clouddesk 占用（尽管其状态为 active=false），导致按键事件未正确分发给前台亮屏的 com.tencent.wecarflow。
>
> 结果: 确认根因是后台应用未正确释放焦点，研发工程师将 Bug 准确指派给相关应用团队，无需现场抓包。

需求拆解

本章节将 Polaris 1.0 平台的核心需求拆解为四大关键能力域。这些能力定义了系统的边界与核心价值，是后续详细功能设计的基础。
本章节采用能力域（Capability Domain）拆解方法，以系统应具备的核心能力为中心，而非具体功能或实现方式。每一能力域仅定义目标、适用范围与责任边界，不涉及接口设计、数据结构或技术选型细节。具体功能点将在后续《功能性需求》中展开，质量与约束要求将在《非功能性需求》中统一定义。

稳定性全栈感知能力

目标：构建覆盖 Android 应用层、系统框架层以及 Linux Host/MCU 异构计算单元的异常捕获体系，实现全栈、全维度的故障感知与现场数据留存。

能力名称	能力描述与目标	适用范围	责任边界
应用层稳定性监控 (App Layer Stability)	**描述**：具备对 Android **应用层 (APK)** 致命异常的实时监测能力。涵盖 Java Crash、App Native Crash (JNI)、ANR 及 App OOM； 在异常触发时同步执行现场冻结与堆栈抓取。 **目标**：确保应用级崩溃捕获率 > 98%，异常现场数据完整性 100%。	Android Framework Third-party Apps System Apps (Launcher等)	**负责**：捕获应用堆栈、页面栈及进程状态； **不负责**： 分析应用内部具体的业务逻辑错误。
系统框架稳定性监控 (System Framework Stability)	**描述**：具备对 **Android 核心服务**及**关键守护进程**的存活状态监测能力； 识别系统级资源耗尽风险（如 Binder 耗尽、句柄泄漏、LMK）。 **目标**：准确识别 SystemServer 死锁 (Watchdog)、 核心服务崩溃、系统异常重启及严重资源拥堵事件。	SystemServer Binder Driver Native Daemons (SurfaceFlinger等)	**负责**： 识别导致系统不稳定的服务异常和资源瓶颈 **不负责**：介入 Linux Kernel 内部调度机制的调试。
异构运行环境监控 (Heterogeneous Env Monitoring)	**描述**：具备对 **Linux Host (PVM)** 及 **MCU** 运行状态的独立监测能力。通过 Native Daemon 标准化采集 Linux 侧服务状态、系统重启事件以及 MCU 侧的心跳与硬件故障码。 **目标**： 实现对底层虚拟化环境与硬件外设健康状况的统一视图监控。	Linux Host (PVM) MCU Hypervisor	**负责**：异构数据的标准化接入、协议对齐及状态监测； **不负责**： 异构系统内部具体业务逻辑的监控实现。

性能与资源度量能力

目标：建立可量化的性能基线，从“主观体验”转向“客观数据”，实现对计算资源（CPU/Mem/IO）的精细化审计。

能力名称	能力描述与目标	适用范围	责任边界
交互体验量化	**描述**：具备对用户核心交互路径（冷/热启动、页面滑动、点击响应）的耗时与流畅度监测能力。 **目标**：量化 App 启动速度与掉帧率（Jank）， 支持版本间性能对比。	Top 核心应用, Launcher, SystemUI	负责采集关键节点的耗时数据；不负责 UI 渲染流程的优化。
资源水位画像	**描述**：具备对进程级资源消耗（CPU使用率、内存占用、IO吞吐量）的周期性采样与超限识别能力。 **目标**：识别“资源刺客”与异常泄漏，绘制 24h 资源趋势图。	所有运行状态下的进程	负责资源数据的统计与归因； 不负责系统资源调度策略。
异常爆发检测	**描述**：具备对特定异常事件（如连续 Crash、持续高负载）的频率统计与突变识别能力。 **目标**：防止单点故障引发的“告警风暴”， 并在端侧进行初步降噪。	全局事件流	负责端侧的流控与阈值判断。

现场还原与协同能力

目标：解决“有报警无日志”的痛点，构建端云协同的自动化取证与远程诊断通道。

能力名称	能力描述与目标	适用范围	责任边界
标准化事件协议体系	**描述**：基于《全局事件注册表》构建统一的事件定义、序列化与解析能力。 **目标**： 确保端侧上报数据与云端解析引擎的语义一致性， 支持协议动态扩展。	端侧 Agent, 车云 SDK, 云端解析服务	负责协议的定义与维护工具链；不限制业务 Payload 的具体内容。
智能现场快照	**描述**：具备“事件驱动”的自动化日志聚合能力， 在异常发生瞬间关联并打包 Trace、Logcat 及系统状态信息。 **目标**：实现 Event 与 Log 文件的 1:1 精准索引。	本地日志系统, 文件系统	负责日志的定位、截取与压缩； 不负责日志内容的语义分析。
远程诊断执行	**描述**：具备安全可控的云端指令接收与本地执行能力， 支持下发 Shell 脚本或调试命令。 **目标**： 在不打扰用户的前提下获取更深度的运行时信息。	Shell 环境, Debug 接口	负责指令通道的建立与执行结果回传； 严禁执行未授权的高危写操作， 不支持批量执行、默认灰度单车、 需显式授权、强审计。

数据智能与运营能力

目标：将海量原始数据转化为可行动的洞察（Actionable Insights），支撑研发与质量团队的决策。

能力名称	能力描述与目标	适用范围	责任边界
端云数据自动关联	**描述**：具备在海量存储中， 根据索引自动将结构化事件与非结构化日志文件进行绑定的能力。 **目标**：消除人工查找日志的成本。	云端存储层	负责数据的逻辑关联与存储生命周期管理。
实时风险预警	**描述**：具备基于时间窗口的流式计算能力， 识别线上故障的爆发趋势并触发告警。 **目标**：实现故障感知。	计算引擎	负责告警策略的计算与推送； 不负责告警后的工单流转。
多维质量可视化	**描述**：具备多维度（版本/车型/时间/地区） 的数据聚合与图表展示能力。 **目标**：提供从“宏观大盘”到“微观个案”的钻取分析视图。	数据仓库, 可视化前端	负责数据的可视化呈现。

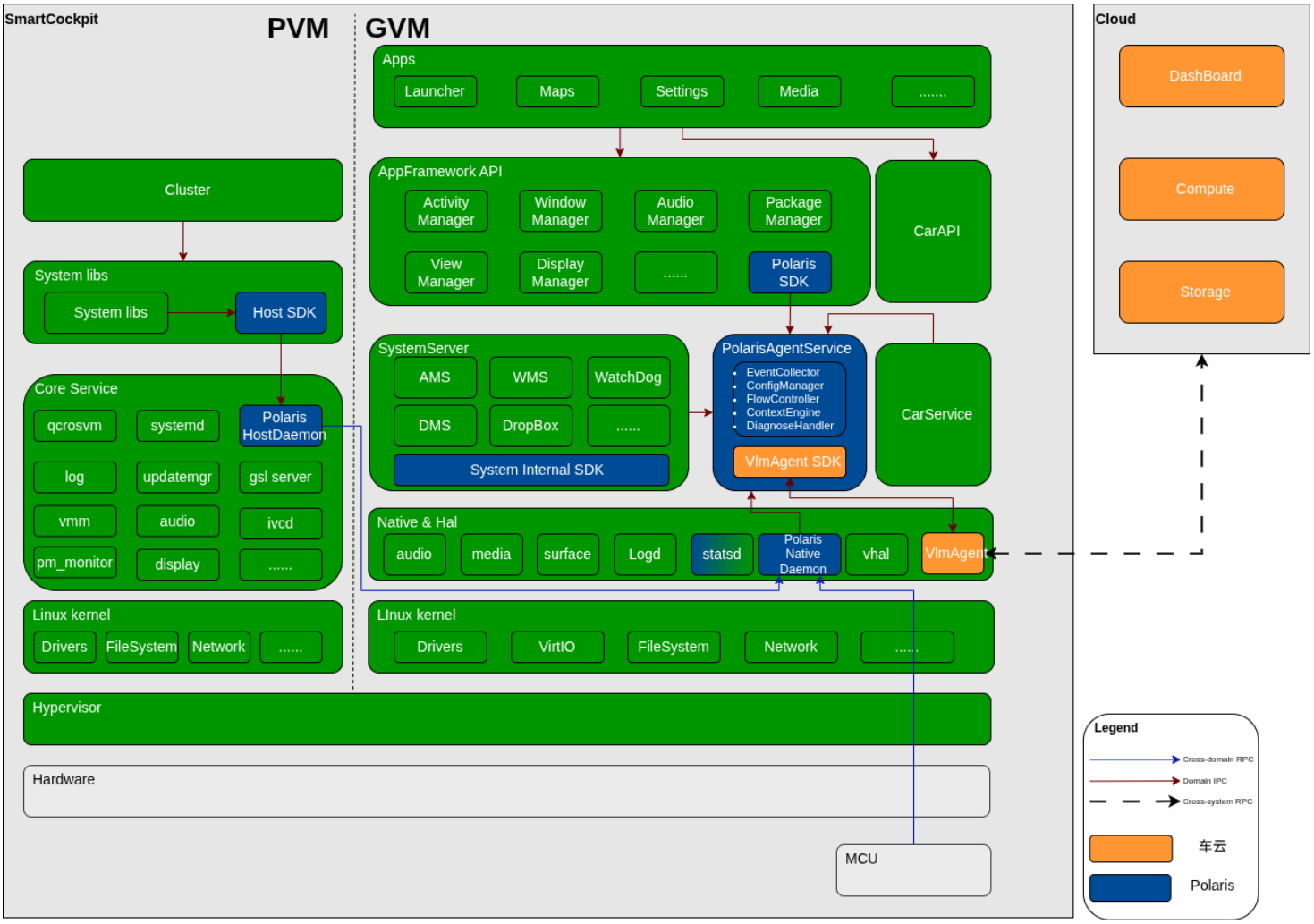
系统总体方案

总体设计概述

Polaris 1.0 基于 **Hypervisor 虚拟化架构** 设计，旨在构建跨越 **GVM (Guest VM - Android)**、**PVM (Primary VM - Linux)** 及 **MCU** 的端云一体化全栈监控系统。

系统采用 **分层架构** 与 **模块化服务设计**。在控制面上，通过 **注册表驱动（Registry-Driven）** 机制实现业务埋点定义与底层采集逻辑的解耦；在数据面上，通过 **双守护进程（Dual Daemon）** 机制打通异构芯片与系统的通信壁垒。系统将计算能力前置至端侧，通过 `PolarisAgentService` 实现数据的实时清洗、流控与现场关联，仅将高价值的结构化数据与诊断日志同步至云端。

系统总体架构图



架构分层详解

业务应用与接口层

本层负责定义数据采集的标准接口，通过自动代码生成技术屏蔽底层通信细节：

- **Polaris SDK**: 面向上层业务应用（如 Launcher, Maps）。该组件由《全局事件注册表》编译生成，提供强类型的事件对象封装与校验逻辑，负责将业务数据序列化并传递给 Framework 层。
- **System Internal SDK**: 面向 SystemServer 内部服务（如 AMS, WMS）。与 Polaris SDK 同源生成，专门用于系统关键服务内部的插桩（Instrumentation），以捕获服务级异常与状态变更。
- **Host SDK**: 面向 PVM 侧的 Linux 应用程序（如 Cluster HMI）、系统核心服务，提供 C++ 标准上报接口，负责将 PVM 侧业务数据发送至 Host Daemon。

框架传输与核心服务层

本层位于 Android GVM，是数据汇聚、策略执行与处理的核心区域：

- **Polaris SDK (Framework API)**: 部署于 `AppFramework API` 层。作为系统级的传输接口实现，它承接来自上层业务的调用请求，并维护与 *PolarisAgentService的 IPC 通信链路，确保数据的可靠投递。
- **PolarisAgentService**: 常驻系统服务，内部包含五个核心功能模块：
  - **EventCollector**: **统一接入模块**。作为 Binder 服务端接收 Polaris SDK 请求；同时作为 LocalSocket 客户端，在服务启动时主动连接 Native Daemon 建立长连接通道，并通过后台线程实时拉取 Native 侧上报的事件流。
  - **ConfigManager**: **配置管理模块**。负责加载本地注册表文件的配置，解析采样率、阈值及采集开关策略。
  - **FlowController**: **流量控制模块**。对输入事件进行频率限制，防止异常爆发导致系统资源耗尽。
  - **ContextEngine**: **现场聚合模块**。在事件通过流控后，负责生成唯一 EventID，挂载系统时间戳，并根据事件类型关联 Logcat、Trace 文件及进程快照，生成索引信息。

- **DiagnoseHandler**: **诊断执行模块**。负责校验并执行来自云端的诊断指令（Shell Command），并管理执行结果的回传。

原生与异构跨域层

本层负责 Android Runtime 之外的底层环境监控及跨虚拟机通信：

- **Polaris Native Daemon (GVM)**:
- **本地采集**: 负责监控 Native 进程崩溃（Tombstone）、系统资源、及 HAL 层状态。
- **跨域网关**: 作为 GVM 侧的通信端点，维护与 PVM/MCU 的连接，接收跨域透传的数据。
- **Polaris Host Daemon (PVM)**:
- **宿主监控**: 负责监控 PVM 侧的 `systemd` 服务状态、关键驱动状态及虚拟机管理服务（qcrosvm/VMM）。

传输通道与云平台

- **VlmAgent**: **统一传输网关**。作为端侧唯一的数据出口，负责接收来自 `PolarisAgentService` 的结构化事件,以及日志文件（按需拉取），执行断点续传、数据压缩与网络流量调度。
- **Cloud Platform**: 负责数据的计算、存储与可视化。

核心设计原则

1. 进程级隔离与服务化：PolarisAgentService 设计为独立系统进程，而非 SystemServer 的内部线程。这种设计带来了两大优势：
   - 稳定性：监控服务的异常（如 OOM）不会导致系统核心服务（SystemServer）崩溃，反之亦然。
   - 高性能：独立的进程空间避免了与 AMS/WMS 争抢主线程资源，确保了监控逻辑的独立调度。
2. 系统核心即客户端：确立 SystemServer 在监控体系中的 Client 身份。AMS、WMS 等核心服务通过 System Internal SDK，以跨进程调用（IPC）的方式向 Polaris 上报数据。这种“旁路监控”模式确保了对系统原有逻辑的最小侵入。
3. 异构接入抽象化：针对 MCU 等异构单元，系统采用 "HAL 驱动适配 + Daemon 统一采集" 的接入原则。不强依赖特定的物理连接方式（如直连或透传），而是通过 Native 层的适配层（Adapter/HAL）屏蔽硬件连接差异，确保架构在不同车型硬件拓扑下的通用性与兼容性。

功能性需求

稳定性全栈感知能力

应用层稳定性监控

FR-STAB-001 应用 Java 崩溃 (Java Crash) 捕获

属性	内容
优先级	P0
前置条件	1.系统层已部署全局监控探针。 2. 监控功能的配置开关处于开启状态。
输入	**触发源**: 应用运行环境（Android Runtime）抛出的**未捕获异常信号**（Uncaught Exception）。 **数据**: 1. 异常堆栈信息（StackTrace）。 2. 异常类型与描述消息（Exception Message）。
处理逻辑	1. **异常拦截**: 在应用进程因异常即将终止前，拦截异常信号，挂起当前线程以确保有足够的 CPU 时间片执行数据采集。 2. **上下文捕获**: 提取崩溃时刻的运行时环境信息，包括但不限于: - 进程名称、线程名称及 ID。 - 应用的前后台状态。 - 当前 Activity 页面栈信息（用于还原用户路径）。 3. **流控策略**: 执行本地频次控制策略。检查该进程在设定时间窗口（如 10 分钟）内的崩溃次数，若超限则降级处理（仅记录统计计数， 不抓取详细堆栈），防止日志写入引发 IO 阻塞。 4. **透传退出**:

属性	内容
	数据采集完成后，**必须**将异常信号交还给系统默认处理程序，确保应用能够按照 Android 系统规范正常退出， 防止应用界面假死或进程僵滞。
输出	1. **结构化事件**：生成包含完整上下文信息的 `GVM_APP_CRASH` 事件对象。 2. **本地日志**：在本地持久化存储区保留一份异常日志备份（作为兜底）。 3. 结构化事件以及本地日志存储目录参考《Polaris 1.0 全局事件ID与注册表规范》

FR-STAB-002 应用无响应 (ANR) 捕获

属性	内容
优先级	P0
前置条件	1. 系统层已部署全局监控探针。 2. 监控功能的配置开关处于开启状态。
输入	**触发源**：系统框架层（Framework）识别到的**应用无响应信号**（AppNotResponding）。 **数据**： 1. 目标应用进程标识（PID/ProcessName）。 2. 系统生成的**堆栈跟踪文件**（Trace File，通常位于 `/data/anr/` 目录）。
处理逻辑	1. **信号识别**： 实时接收系统 ActivityManagerService 发出的 ANR 通知。 2. **目标过滤**： 根据配置白名单判断是否采集该进程，过滤非关注应用的 ANR 事件。 3. **堆栈截取**： 读取系统生成的 Trace 文件，根据目标 PID **精准截取**该进程及其子线程的堆栈片段（需剔除文件中的其他无关进程数据， 以减少数据体积）。 4. **快照关联**： 获取 ANR 发生时刻的系统负载信息（Load Avg / CPU Usage / IO Wait）并与堆栈信息打包。 5. **流控策略**： 执行本地频次控制策略。检查该进程在设定时间窗口（如 10 分钟）内的 ANR 次数，若超限则仅记录计数，不再执行堆栈截取操作。
输出	1. **结构化事件**：生成包含 Trace 附件索引（Reference）的 `GVM_APP_ANR` 事件对象。 2. **本地日志**：在本地持久化存储区生成关联的证据包（包含截取的 Trace 片段与系统负载快照）。 3. 结构化事件以及本地日志存储目录参考《Polaris 1.0 全局事件ID与注册表规范》

FR-STAB-003 应用 Native 库崩溃 (App JNI Crash) 捕获

属性	内容
优先级	P0
前置条件	1. 系统层已部署全局监控探针（Native Daemon）。 2. 监控功能的配置开关处于开启状态。
输入	**触发源**： 应用进程（APP）加载的 JNI 动态库触发致命信号（SIGSEGV/SIGABRT）。 **数据**： 1. 系统生成的 Tombstone 崩溃文件（通常位于 `/data/tombstones/` ）。 2. 进程退出信号（Signal Code）。
处理逻辑	1. **监听与解析**： 实时监听系统 Tombstone 文件的生成事件，读取文件头部信息。 2. **身份识别**： 检查崩溃进程的 UID 或进程名称。若属于**非系统核心进程**（即普通 App），则执行应用级采集逻辑；若为系统服务则忽略 （交由系统框架监控处理）。 3. **指纹去重**： 基于“应用名称 + 崩溃堆栈关键帧”生成唯一指纹，在端侧聚合重复的崩溃事件，防止日志风暴。 4. **事件生成**： 将非结构化的 Tombstone 数据转换为标准化的事件对象。

属性	内容
输出	1. **结构化事件**：生成 `GVM_APP_NATIVE_CRASH` 事件对象。 2. **本地日志**：建立事件 ID 与原始 Tombstone 文件的索引关联。 3. 结构化事件以及本地日志存储目录参考《Polaris 1.0 全局事件ID与注册表规范》

FR-STAB-004 应用 OOM (App OOM) 事件监控

属性	内容
优先级	P0
前置条件	1. 系统层已部署全局监控探针。 2. 监控功能的配置开关处于开启状态。 3. 具备获取应用进程退出详细原因的能力（如 ApplicationExitInfo 或类似机制）。
输入	**触发源**： 应用进程**意外终止信号**。 **数据**： 1. 进程退出原因描述（Exit Reason，需区分系统回收/异常崩溃）。 2. 进程终止前的内存使用统计数据（如 PSS/RSS/VSS）。
处理逻辑	1. **原因甄别**： 在进程退出后，识别退出原因。准确区分是**系统低内存查杀 (LMK)**（通常表现为 `REASON_LOW_MEMORY` ）还是**Java 堆内存耗尽**（通常表现为 `OutOfMemoryError` 导致的 Crash）引发的异常。 2. **内存快照回溯**： 尝试关联该进程在终止前最近一次采集的内存统计数据（如 PSS/RSS），以辅助判断是否存在内存泄漏。 3. **风暴抑制**： 针对前台应用因 OOM 导致的反复重启进行检测。若同一应用在短时间内（如 5 分钟）连续触发 OOM，则实施指数退避策略，减少上报频次。 4. **事件生成**： 组装 OOM 事件负载，标记明确的 OOM 类型（System LMK / Java OOM）。
输出	1. **结构化事件**：生成包含内存快照信息的 `GVM_APP_OOM` 事件对象。 2. **本地日志**：记录关联的系统内存水位信息（MemInfo）。 3. 结构化事件以及本地日志存储目录参考《Polaris 1.0 全局事件ID与注册表规范》

系统框架稳定性监控

FR-STAB-005 SystemServer Watchdog (死锁) 监控

属性	内容
优先级	P0
前置条件	1. 监控探针已植入系统看门狗（Watchdog）模块或具备监听能力。 2. 监控功能的配置开关处于开启状态。
输入	**触发源**： 系统关键锁或核心线程（如 UI Thread, IoThread）**等待超时信号**（通常阈值为 60秒）。 **数据**： 1. 阻塞线程的完整堆栈信息（Stack Traces）。 2. 持锁状态与锁竞争信息（Lock Contention）。
处理逻辑	1. **重启前拦截**： 在系统触发看门狗复位（Soft Reboot / Restart）流程前，优先执行监控逻辑，确保有短暂的时间窗口进行数据转存。 2. **现场固化**： 立即将当前的系统全量线程堆栈（Traces.txt）复制或转存至持久化存储区，**防止系统重启过程清理现场文件**，导致关键证据丢失。 3. **异常标记**： 在磁盘特定位置写入“非正常重启”标志位（Flag），以便系统下次启动时进行归因统计，区分正常关机与异常重启。 4. **事件上报**： 尝试通过 Native 通道（因为 Java 层可能已挂死）发送死锁事件。

属性	内容
输出	1. **结构化事件**：生成包含死锁堆栈索引的 `GVM_SYS_WATCHDOG` 事件对象。 2. **本地日志**：在持久化目录保留死锁现场的 Trace 文件。 3. 结构化事件以及本地日志存储目录参考《Polaris 1.0 全局事件ID与注册表规范》

FR-STAB-006 Android 系统异常重启 (System Restart) 监控

属性	内容
优先级	P0
前置条件	1. 系统完成启动初始化流程（Boot Completed）。 2. 具备读取系统启动属性（Boot Reason）及持久化存储的权限。
输入	**触发源**： 系统**启动完成广播** (Boot Completed) 或同等时机的初始化信号。 **数据**： 1. 系统启动原因属性（如 `sys.boot.reason` 或 `ro.boot.bootreason` ）。 2. 持久化存储中的**历史状态标记**（包含上一次启动时间戳、Watchdog/Crash 遗留的异常标志位）。
处理逻辑	1. **原因推断**： 对比本次启动原因与上一次运行状态进行逻辑仲裁： - **已知异常**：若存在 Watchdog 或 Core Crash 遗留的标记，判定为对应的系统级故障重启。 - **内核恐慌**：若启动属性标识为 Kernel Panic 或 WDT（硬件看门狗），判定为内核级重启。 - **正常重启**：若标识为用户主动关机、OTA 升级或常规电源管理操作，判定为正常重启。 - **掉电/未知**：若无任何异常标记且非正常重启，判定为异常掉电或未知原因重启。 2. **时长计算**： 基于上一次记录的启动时间戳，计算上一次系统正常运行的时长（Uptime），用于评估系统平均无故障时间（MTBF）。 3. **状态重置**： 分析完成后，清除历史异常标记，更新本次启动时间戳，为下一次监控周期做准备。
输出	1. **结构化事件**：生成包含重启原因分类（Category）及运行时长（Duration）的 `GVM_SYS_RESTART` 事件对象。 2. 结构化事件以及本地日志存储目录参考《Polaris 1.0 全局事件ID与注册表规范》

FR-STAB-007 核心服务崩溃 (Core Service Crash) 监控

属性	内容
优先级	P0
前置条件	1. 监控进程具备监听系统服务管理器（ServiceManager）或 init 进程状态的能力。 2. 核心进程白名单配置已加载。
输入	**触发源**： 1. Native 守护进程崩溃产生的 Tombstone 文件。 2. ServiceManager 发出的 `DeathRecipient` 通知。 3. init 进程发出的 `SIGCHLD` 信号。 **数据**： 1. 崩溃进程名称（Process Name）及 PID。 2. 进程退出状态码或终止信号。
处理逻辑	1. **核心识别**： 匹配崩溃进程名称是否在**核心白名单**中（如 `surfaceflinger` ， `audioserver` ， `netd` ， `lmkd` ）。若不在白名单，则视为普通 Native Crash 处理（参考 FR-STAB-003）。 2. **多源仲裁**： 优先使用 Tombstone 信息（包含详细堆栈），若未生成 Tombstone（如被系统强杀或 Watchdog 处决），则使用 ServiceManager 通知作为补充来源。 3. **等级判定**： 根据服务重要性标记故障等级（例如 SurfaceFlinger 崩溃标记为“致命”，会导致屏幕黑屏或系统软重启）。 4. **事件生成**： 组装核心服务崩溃事件，记录服务名称、崩溃时间及退出原因。

属性	内容
输出	1. **结构化事件**：生成包含服务名及影响等级的 `GVM_CORE_CRASH` 事件对象。 2. **本地日志**：关联该时间点附近的系统日志（Logcat）与崩溃堆栈。 3. 结构化事件以及本地日志存储目录参考《Polaris 1.0 全局事件ID与注册表规范》

FR-STAB-008 系统低内存 (LMK) 事件监控

属性	内容
优先级	P0
前置条件	1. 系统启用 Low Memory Killer 机制（如 Userspace LMKD）。 2. 监控组件具备接收系统内存管理模块通知的权限。
输入	**触发源**： 系统内存管理守护进程（lmkd）执行的**进程查杀动作**。 **数据**： 1. 目标进程信息（PID、UID、Process Name）。 2. 查杀时的决策依据（OOM Score Adj）。 3. 触发查杀时的系统内存压力状态（Memory Pressure State / PSI）。
处理逻辑	1. **动作捕获**： 实时感知 LMK 的查杀行为。**推荐方案**：采用源码插桩（Instrumentation）方式，在 `lmkd` 执行 kill 操作的原子逻辑处植入通知钩子，以获取零延迟、高精度的上下文信息；（备选方案：监听 EventLog 中的 `lmk_kill` 标签）。 2. **水位快照**： 同步记录系统当前的内存水位详情（MemTotal, MemFree, SwapUsed, Cached），用于后续分析是物理内存耗尽还是虚拟内存（Swap）耗尽。 3. **聚合去噪**： 执行时间窗口聚合策略。由于内存压力常导致短时间内连续查杀多个进程，需将同一压力波峰内（如 1 秒）的一组查杀事件聚合，避免产生告警风暴。 4. **严重性判定**： 识别被杀进程的类型。若被杀进程为前台可见应用或关键服务，标记为“高影响”事件。
输出	1. **结构化事件**：生成包含被杀进程列表及内存水位的 `GVM_SYS_LMK` 事件对象。 2. **本地日志**：保留查杀时刻的 `meminfo` 快照。 3. 结构化事件以及本地日志存储目录参考《Polaris 1.0 全局事件ID与注册表规范》

FR-STAB-009 Binder 通信异常监控

FR-STAB-010 文件句柄 (FD) 泄漏监控

异构运行环境监控 (Heterogeneous Env Monitoring)

FR-STAB-012 Linux Host (PVM) 重启与状态监控

FR-STAB-013 MCU 故障码与心跳监控

FR-STAB-014 异构关键进程 (PVM Critical Process) 稳定性监控

非功能需求

端云交互协议设计

安全与隐私

风险 & 限制 & 依赖

实施计划

阶段划分

资源需求计划

附录