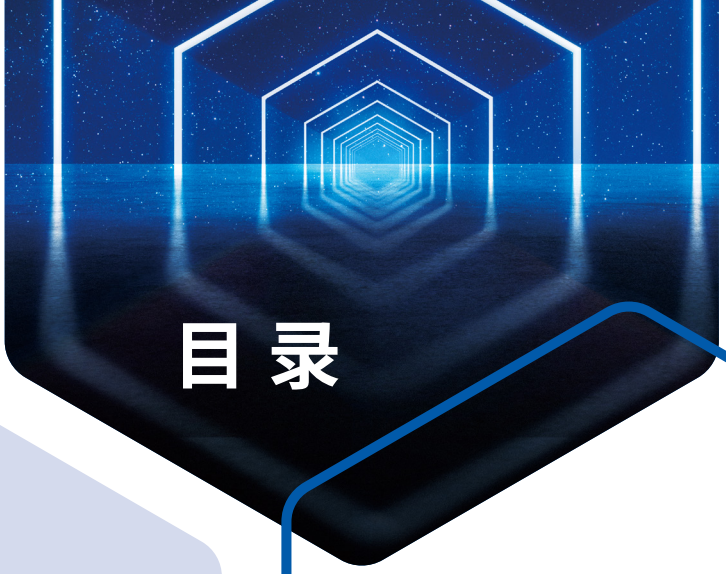
openEuler 23.09

技术白皮书



目录

CONTENTS

概述 01

欧拉开源操作系统（openEuler, 简称“欧拉”）从服务器操作系统正式升级为面向数字基础设施的操作系统，支持服务器、云计算、边缘计算、嵌入式等应用场景，支持多样性计算，致力于提供安全、稳定、易用的操作系统。

欧拉开源社区通过开放的社区形式与全球的开发者共同构建一个开放、多元和架构包容的软件生态体系，孵化支持多种处理器架构、覆盖数字基础设施全场景，推动企业数字基础设施软硬件、应用生态繁荣发展。

2019 年 12 月 31 日，面向多样性计算的操作系统开源社区 openEuler 正式成立。

2020 年 3 月 30 日，openEuler 20.03 LTS（Long Term Support，简写为 LTS，中文为长生命周期支持）版本正式发布，为 Linux 世界带来一个全新的具备独立技术演进能力的 Linux 发行版。

2020 年 9 月 30 日，首个 openEuler 20.09 创新版发布，该版本是 openEuler 社区中的多个企业、团队、独立开发者协同开发的成果，在 openEuler 社区的发展进程中具有里程碑式的意义，也是中国开源历史上的标志性事件。

2021 年 3 月 31 日，发布 openEuler 21.03 内核创新版，该版本将内核升级到 5.10, 还在内核方向实现内核热升级、内存分级扩展等多个创新特性，加速提升多核性能，构筑千核运算能力。

2021 年 9 月 30 日，全新 openEuler 21.09 创新版如期而至，这是欧拉全新发布后的第一个社区版本，实现了全场景支持。增强服务器和云计算的特性，发布面向云原生的业务混部 CPU 调度算法、容器化操作系统 KubeOS 等关键技术；同时发布边缘和嵌入式版本。

2022 年 3 月 30 日，基于统一的 5.10 内核，发布面向服务器、云计算、边缘计算、嵌入式的全场景 openEuler 22.03 LTS 版本，聚焦算力释放，持续提升资源利用率，打造全场景协同的数字基础设施操作系统。

2022 年 9 月 30 日，发布 openEuler 22.09 创新版本，持续补齐全场景的支持。

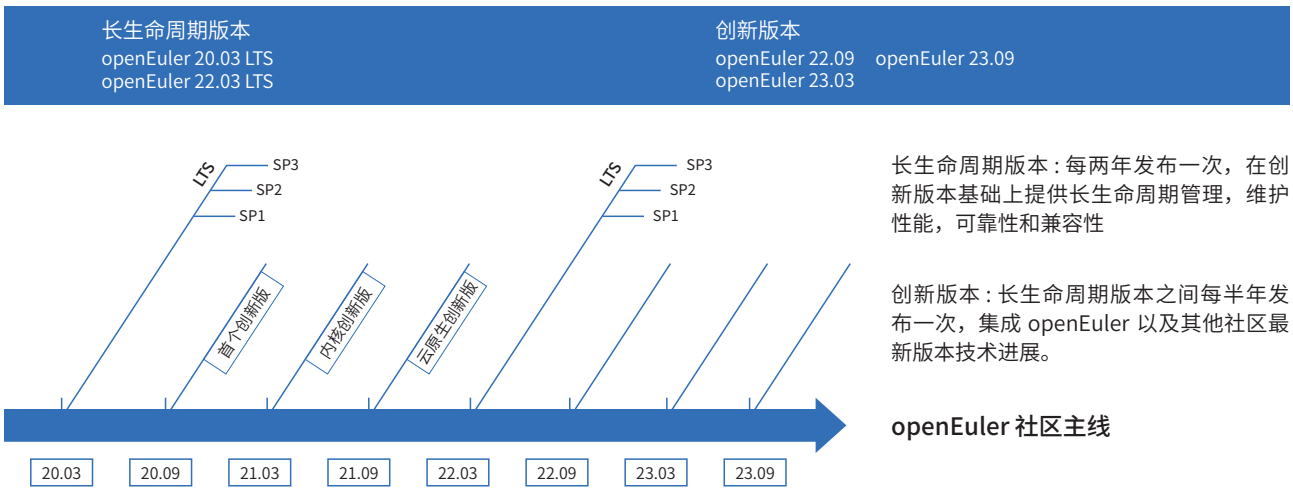
2022 年 12 月 30 日，发布 openEuler 22.03 LTS SP1 版本，打造最佳迁移工具实现业务无感迁移，性能持续领先。

2023 年 3 月 30 日，发布 openEuler 23.03 内核创新版本，采用 Linux Kernel 6.1 内核，为未来 openEuler 长生命周期版本采用 6.x 内核提前进行技术探索，方便开发者进行硬件适配、基础技术创新及上层应用创新。

2023 年 6 月 30 日，发布 openEuler 22.03 LTS SP2 版本，场景化竞争力特性增强，性能持续领先。

2023 年 9 月 30 日，发布 openEuler 23.09 创新版本，是基于 6.4 内核的创新版本（参见版本生命周期），提供更多新特性和功能，给开发者和用户带来全新的体验，服务更多的领域和更多的用户。

openEuler 版本管理

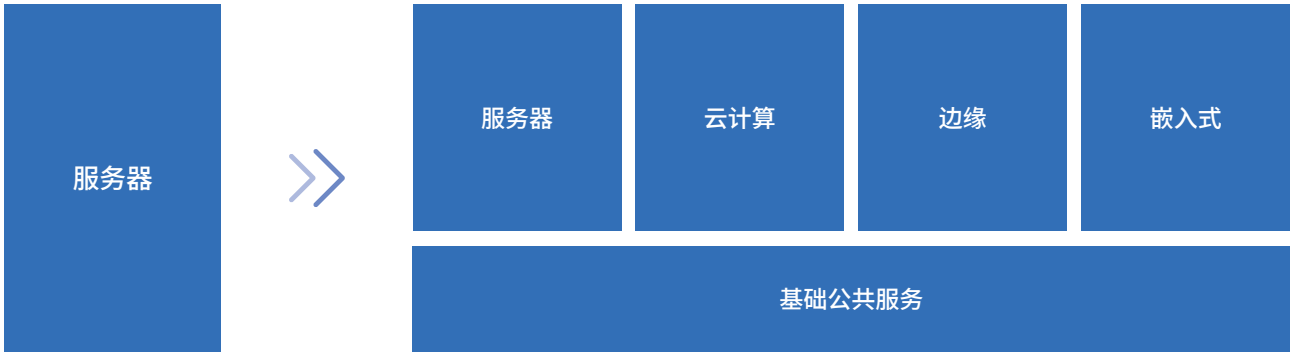


openEuler 作为一个操作系统发行版平台，每两年推出一个 LTS 版本。该版本为企业级用户提供一个安全稳定可靠的操作系统。

openEuler 也是一个技术孵化器。通过每半年发布一次的创新版，快速集成 openEuler 以及其他社区的最新技术成果，将社区验证成熟的特性逐步回合到发行版中。这些新特性以单个开源项目的方式存在于社区，方便开发者获得源代码，也方便其他开源社区使用。

社区中的最新技术成果持续合入发行版，发行版通过用户反馈反哺技术，激发社区创新活力，从而不断孵化新技术。发行版平台和技术孵化器互相促进、互相推动、牵引版本持续演进。

openEuler 覆盖全场景的创新平台



openEuler 已支持 X86、ARM、SW64、RISC-V、LoongArch 多处理器架构，逐步扩展 PowerPC 等更多芯片架构支持，持续完善多样性算力生态体验。

openEuler 社区面向场景化的 SIG 不断组建，推动 openEuler 应用边界从最初的服务器场景，逐步拓展到云计算、边缘计算、嵌入式等更多场景。openEuler 正成为覆盖数字基础设施全场景的操作系统。

openEuler 希望与广大生态伙伴、用户、开发者一起，通过联合创新、社区共建，不断增强场景化能力，最终实现统一操作系统支持多设备，应用一次开发覆盖全场景。

openEuler 开放透明的开源软件供应链管理

开源操作系统的构建过程，也是供应链聚合优化的过程。拥有可靠开源软件供应链，是大规模商用操作系统的基础。openEuler 从用户场景出发，回溯梳理相应的软件依赖关系，理清所有软件包的上游社区地址，源码和上游对应验证。完成构建验证、分发、实现生命周期管理。开源软件的构建、运行依赖关系、上游社区，三者之前形成闭环且完整透明的软件供应链管理。

02 平台架构

系统框架

openEuler 是覆盖全场景的创新平台，在引领内核创新，夯实云化基座的基础上，面向计算架构互联总线、存储介质发展新趋势，创新分布式、实时加速引擎和基础服务，结合边缘、嵌入式领域竞争力探索，打造全场景协同的面向数字基础设施的开源操作系统。

openEuler 23.09 发布面向服务器、云原生、边缘和嵌入式场景的全场景操作系统版本，统一基于 Linux Kernel 6.4 构建，对外接口遵循 POSIX 标准，具备天然协同基础。同时 openEuler 23.09 版本集成分布式软总线、KubeEdge+ 边云协同框架等能力，进一步提升数字基础设施协同能力，构建万物互联的基础。

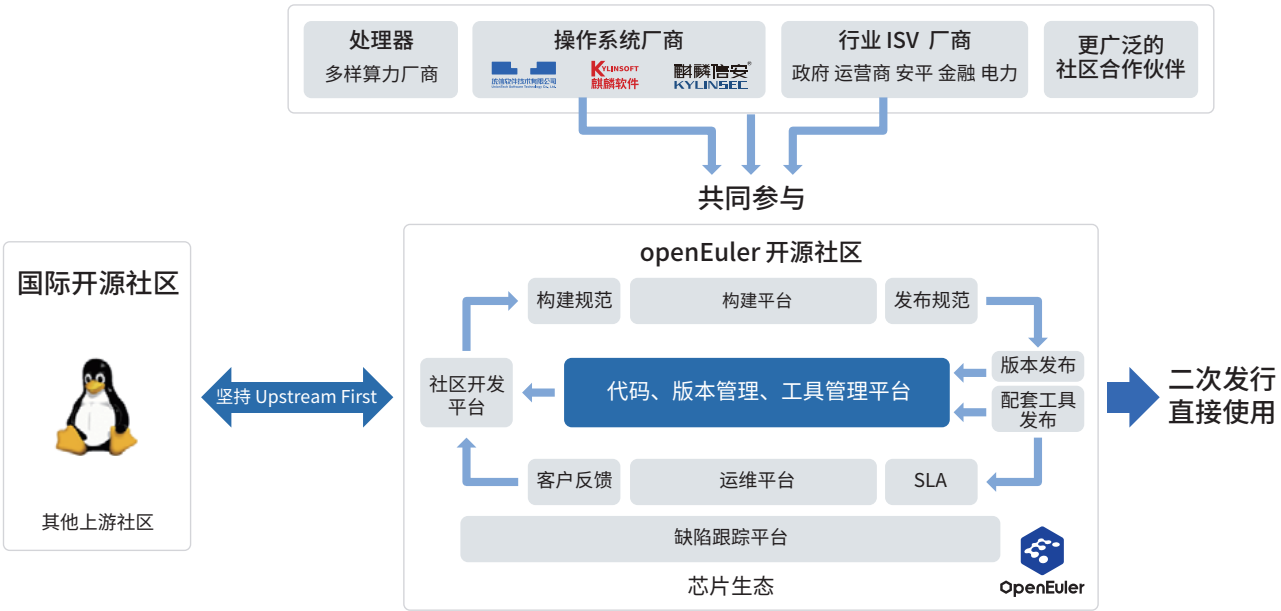
面向未来，社区将持续创新、社区共建、繁荣生态，夯实数字基座。

繁荣社区生态

- 友好桌面环境：UKUI、DDE 、Xfce、Kiran-desktop、GNOME 桌面环境，丰富社区桌面环境生态。
- 欧拉 DevKit：支持操作系统迁移、兼容性评估、简化安全配置 secPaver 等更多开发工具。

平台框架

openEuler 社区与上下游生态建立连接，构建多样性的社区合作伙伴和协作模式，共同推进版本演进。



硬件支持

openEuler 社区当前已与多个设备厂商建立丰富的南向生态，Intel、AMD 等主流芯片厂商的加入和参与，openEuler 全版本支持 X86、ARM、申威、龙芯、RISC-V 五种架构，并支持多款 CPU 芯片，包括龙芯 3 号、兆芯开先 / 开胜系列、Intel IceLake/ Sapphire Rapids、AMD EPYC Milan /Genoa 等芯片系列，支持多个硬件厂商发布的多款整机型号、板卡型号，支持网卡、RAID、FC、GPU&AI、DPU、SSD、安全卡七种类型的板卡，具备良好的兼容性。

支持的 CPU 架构如下：

硬件类型	X86	ARM
CPU	Intel、AMD、兆芯、海光	鲲鹏、飞腾

支持的整机如下：

硬件类型	X86	ARM
整机	Intel：超聚变 AMD：超微 海光：中科可控	鲲鹏：泰山 飞腾：青松、宝德
	兆芯：兆芯	

支持的板卡类型如下：

硬件类型	X86	ARM
网卡	华为、Mellanox、Intel	华为、Mellanox、Intel
Raid	Avago	Avago
FC	Marvell、Emulex	Marvell、Emulex
GPU & AI	Nvidia	Nvidia
SSD	华为	华为

全版本支持的硬件型号可在兼容性网站查询：https://www.openeuler.org/zh/compatibility/。

03 运行环境

服务器

若需要在物理机环境上安装 openEuler 操作系统，则物理机硬件需要满足以下兼容性和最小硬件要求。

硬件兼容支持请查看 openEuler 兼容性列表：https://openeuler.org/zh/compatibility/。

部件名称	最小硬件要求
架构	ARM64、x86_64
内存	为了获得更好的体验，建议不小于 4GB
硬盘	为了获得更好的体验，建议不小于 20GB

虚拟机

openEuler 安装时，应注意虚拟机的兼容性问题，当前已测试可以兼容的虚拟机及组件列表如下：

- centos-6 qemu 6.2.0-79.oe2309 libvirt 6.2.0-59.oe2309 virt-manager 4.1.0.2-oe2309
- centos-7 qemu 6.2.0-79.oe2309 libvirt 6.2.0-59.oe2309 virt-manager 4.1.0.2-oe2309
- centos-8 qemu 6.2.0-79.oe2309 libvirt 6.2.0-59.oe2309 virt-manager 4.1.0.2-oe2309
- windows2016 qemu 6.2.0-79.oe2309 libvirt 6.2.0-59.oe2309 virt-manager 4.1.0.2-oe2309 qemu 8.1.0
- windows2019 qemu 6.2.0-79.oe2309 libvirt 6.2.0-59.oe2309 virt-manager 4.1.0.2-oe2309 qemu 8.1.0

部件名称	最小虚拟化空间要求
架构	ARM64、x86_64
CPU	2 个 CPU
内存	为了获得更好的体验，建议不小于 4GB
硬盘	为了获得更好的体验，建议不小于 20GB

边缘设备

若需要在边缘设备环境上安装 openEuler 操作系统，则边缘设备硬件需要满足以下兼容性和最小硬件要求。

部件名称	最小硬件要求
架构	ARM64、x86_64
内存	为了获得更好的体验，建议不小于 4GB
硬盘	为了获得更好的体验，建议不小于 20GB

嵌入式

若需要在嵌入式环境上安装 openEuler 操作系统，则嵌入式硬件需要满足以下兼容性和最小硬件要求。

部件名称	最小硬件要求
架构	ARM64、ARM32、x86_64
内存	为了获得更好的体验，建议不小于 512MB
硬盘	为了获得更好的体验，建议不小于 256MB

04 场景创新

异构通用内存管理框架（GMEM）特性

在后摩尔时代，GPU、TPU 和 FPGA 等专用异构加速器设备正不断涌现，它们与 CPU 类似，需要将数据放在本地内存（例如 LPDDR 或 HBM）中以提高计算速度。加速器厂商们也不可避免地需要开发复杂的内存管理系统。

现行加速器内存管理方案存在诸多缺陷：

- CPU 侧内存管理与加速器侧分离，数据显式搬移，加速器内存管理的易用性和性能难以平衡。
- 大模型场景下加速器设备 HBM 内存（Hign BandWidth Memory）严重不足，现有的手动 swap 方案性能损耗大且通用性差。
- 搜推、大数据场景存在大量无效数据搬移，缺少高效内存池化方案。

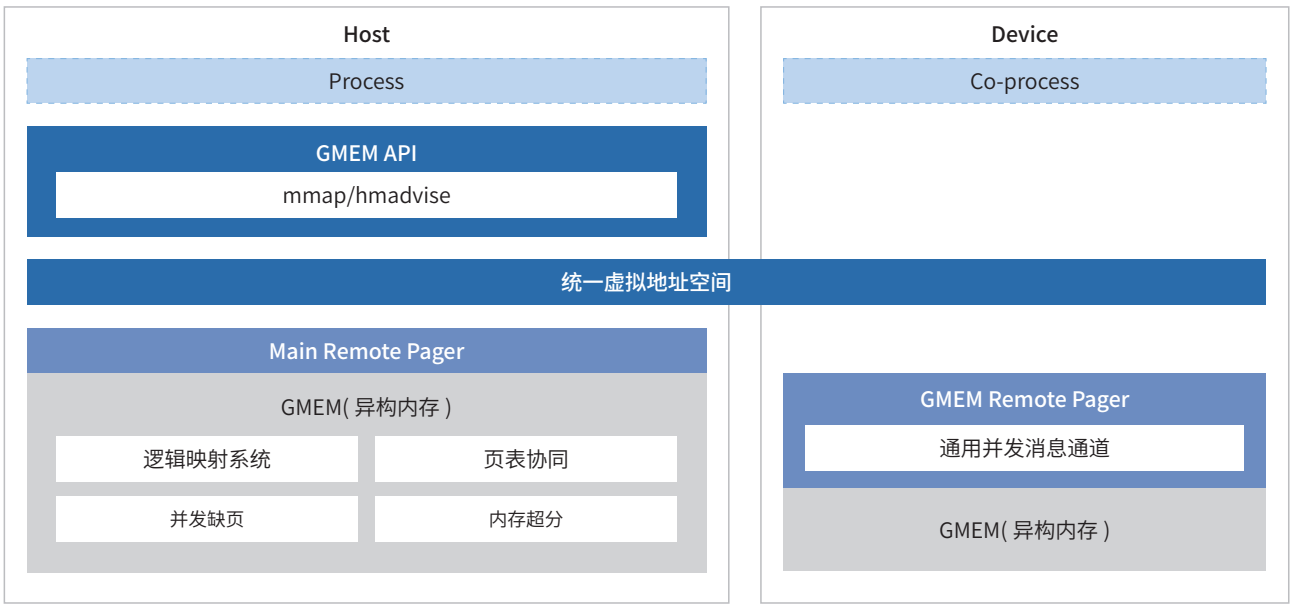
Linux 现有的 HMM 框架，编程复杂度高且依赖人工调优，性能和可移植性差，引发 OS 社区反弹，最终导致 HMM 方案搁浅。异构加速器领域亟需高效的统一内存管理机制。

异构通用内存管理框架 GMEM （Generalized Memory Management），提供了异构内存互联的中心化管理机制，且 GMEM API 与 Linux 原生内存管理 API 保持统一，易用性强，性能与可移植性好。

加速器使用 GMEM API 将内存接入统一地址空间后，可自动获得 GMEM 面向异构内存编程优化的能力。与此同时，加速器驱动无需重复实现内存管理框架，大幅降低开发维护带来的成本。

开发者使用一套统一申请、释放的 API，即可完成异构内存编程，无需处理内存搬移等细节。在加速器 HBM 内存不足时，GMEM 可将 CPU 内存作为加速器缓存，透明地超分 HBM，无需应用手动 swap。GMEM 提供高效免搬移的内存池化方案，当内存池以共享方式接入后，可解决数据反复搬移的痛点。

功能描述



GMEM 革新了 Linux 内核中的内存管理架构，其中逻辑映射系统屏蔽了 CPU 和加速器地址访问差异，remote_pager 内存消息交互框架提供了设备接入抽象层。在统一的地址空间下，GMEM 可以在数据需要被访问或换页时，自动地迁移数据到 OS 或加速器端。

异构内存特性

为了结合加速器算力与 CPU 通用算力，实现统一的内存管理和透明内存访问，GMEM 设计了统一虚拟内存地址空间机制，将原本的 OS 与加速器并行的两套地址空间合并为统一虚拟地址空间。

GMEM 建立了一套新的逻辑页表去维护这个统一虚拟地址空间，通过利用逻辑页表的信息，可以维护不同处理器、不同微架构间多份页表的一致性。基于逻辑页表的访存一致性机制，内存访问时，通过内核缺页流程即可将待访问内存在主机与加速器进行搬移。在实际使用时，加速器可在内存不足时可以借用主机内存，同时回收加速器内的冷内存，达到内存超分的效果，突破模型参数受限于加速器内存的限制，实现低成本的大模型训练。

通过在内核中提供 GMEM 高层 API，允许加速器驱动通过注册 GMEM 规范所定义的 MMU 函数直接获取内存管理功能，建立逻辑页表并进行内存超分。逻辑页表将内存管理的高层逻辑与 CPU 的硬件相关层解耦，从而抽象出能让各类加速器复用的高层内存管理逻辑。加速器只需要注册底层函数，不再需要实现任何统一地址空间协同的高层逻辑。

Remote Pager 内存消息交互框架

Remote Pager 作为 OS 内核外延的内存管理框架，设计并实现了主机和加速器设备之间协作的消息通道、进程管理、内存交换和内存预取等模块，由独立驱动 remote_pager.ko 使能。通过 Remote Pager 抽象层可以让第三方加速器很容易的接入 GMEM 系统，简化设备适配难度。

用户 API

用户可以直接使用 OS 的 mmap 分配统一虚拟内存，GMEM 在 mmap 系统调用中新增分配统一虚拟内存的标志（MMAP_PEER_SHARED）。

同时 libgmem 用户态库提供了内存预取语义 hmadvise 接口，协助用户优化加速器内存访问效率（参考 https://gitee.com/openeuler/libgmem）。

约束限制

- 目前仅支持 2M 大页，所以 host OS 以及 NPU 卡内 OS 的透明大页需要默认开启。
- 通过 MAP_PEER_SHARED 申请的异构内存目前不支持 fork 时继承。

综上，GMEM 使用方法可参考以下链接：

https://gitee.com/openeuler/docs/tree/master/docs/zh/docs/GMEM

应用场景

异构统一内存编程

在面向异构内存编程时，使用 GMEM 可分配 CPU 和加速器之间的统一虚拟内存，CPU 内存与加速器内存可共享一个指针，显著降低了异构编程复杂度。当前基于 NPU 试点，驱动仅需百行修改即可接入 GMEM，替换原有约 4000 行内存管理框架代码。

加速器内存自动超分

使用 GMEM 接口分配内存时，将不受加速器的物理内存容量所限制，应用可以透明地超分内存（当前上限为 CPU 的 DRAM 容量）。GMEM 将较冷的设备内存页换出到 CPU 内存上，拓展了应用处理的问题规模，实现高性能、低门槛训推。通过 GMEM 提供的极简异构内存管理框架，在超大模型训练中，GMEM 性能领先 NVIDIA-UVM。随着内存使用量增长，领先比例不断提升，在超分两倍以上时可领先 NVIDIA-UVM 60% 以上。（数据基于 NPU-Ascend910 与 GPU-A100 硬件，在相同 HBM 内存条件下测试。）

开源大模型原生支持（LLaMA 和 ChatGLM）

虽然大模型日渐火爆，但普通用户使用大模型还存在一定的门槛。llama.cpp 和 chatglm-cpp 是基于 C/C++ 实现的模型推理框架，通过模型量化等手段，支持用户可以在 CPU 机器上完成开源大模型的部署和使用。

功能描述

llama.cpp 支持多个英文开源大模型的部署，如 LLaMa/LLaMa2/Vicuna 等。

chatglm-cpp 支持多个中文开源大模型的部署，如 ChatGLM-6B/ChatGLM2-6B/Baichuan-13B 等。

用户可通过 llama.cpp 和 chatglm-cpp 选择适合自己的开源大模型进行部署。

其主要特性如下：

- 基于 ggml 的 C/C++ 实现。
- 通过 int4/int8 量化、优化的 KV 缓存和并行计算等多种方式加速内存高效 CPU 推理。

使用方法请参考 openEuler 支持部署大模型。

应用场景

用户可通过 llama.cpp/chatglm-cpp 在 CPU 机器上部署大模型，并体验智能问答、AI 对话等功能。

05 内核创新

openEuler 6.4 内核中的新特性 / 合入的关键继承特性

openEuler 23.09 基于 Linux Kernel 6.4 内核构建，在此基础上，回合了 openEuler 社区低版本的有益特性及社区创新特性。

- 潮汐 affinity 调度特性：感知业务负载动态调整业务 CPU 亲和性，当业务负载低时使用 prefered cpus 处理，增强资源的局部性；当业务负载高时，突破 preferred cpus 范围限制，通过增加 CPU 核的供给提高业务的 QoS。
- CPU QoS 优先级负载均衡特性：在离线混部 CPU QoS 隔离增强，支持多核 CPU QoS 负载均衡，进一步降低离线业务 QoS 干扰。
- SMT 驱离优先级反转特性：解决混部 SMT 驱离特性的优先级反转问题，减少离线任务对在线任务 QoS 的影响。
- 混部多优先级：允许 cgroup 配置 -2~2 的 cpu.qos_level，即多个优先级，使用 qos_level_weight 设置不同优先级权重，按照 CPU 的使用比例进行资源的划分。并提供唤醒抢占能力。在提高机器利用率的同时，保证高优和延迟敏感的在线业务不受离线业务的影响。
- 可编程调度：基于 eBPF 的可编程调度框架，支持内核调度器动态扩展调度策略，以满足不同负载的性能需求，具备以下特点：

  （1）标签管理机制：开放对任务和任务组进行标签标记的能力，用户和内核子系统可通过接口对特定工作负载进行标记，调度器通过标签可以感知特定工作负载的任务。

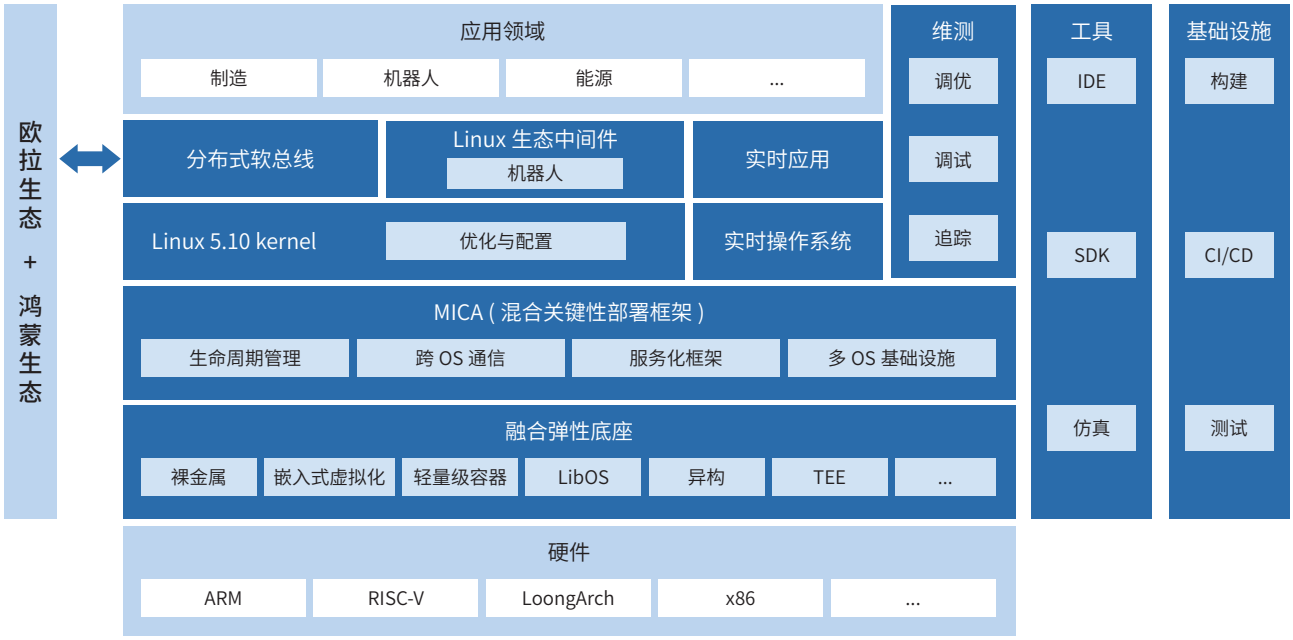
  （2）抢占、选核、选任务等功能点的策略扩展：可编程调度框架支持 CFS 调度类抢占、选核、选任务等功能的策略扩展，提供精心设计的扩展点和丰富的辅助方法，帮助用户简单，高效的扩展策略。
- Numa Aware spinlock：基于 MCS 自旋锁在锁传递算法上针对多 NUMA 系统优化，通过优先在本 NUMA 节点内传递，能大量减少跨 NUMA 的 Cache 同步和乒乓，从而提升锁的整体吞吐量，提升业务性能。
- 支持 TCP 压缩：大数据等场景节点间数据传输量大，网络传输是性能瓶颈。在 TCP 层对指定端口的数据进行压缩后再传输，收包侧把数据解压后再传给用户态，从而提升分布式场景节点间数据传输的效率。
- 热补丁：内核热补丁主要针对内核的函数实现的 bug 进行免重启修复，原理主要在于如何完成动态函数替换，openEuler 上的 livepatch 与 Linux 主线上的实现略有不同，采用直接修改指令的方法，而非主线基于 ftrace 实现，在运行时直接跳转至新函数，无需经过查找中转，效率较高。
- Sharepool 共享内存：Sharepool 共享内存是一种在多个进程之间共享数据的技术。它允许多个进程访问同一块内存区域，从而实现数据共享。在共享内存中，多个进程可以同时读写同一块内存区域，这样可以避免进程之间频繁地进行数据拷贝，提高了数据访问的效率。同时，共享内存还可以减少进程之间的通信开销，提高了系统的整体性能。
- Memcg 异步回收：Memcg 是一种内核机制，用于限制和管理进程组的内存使用量。当一个进程组使用的内存超过了 Memcg 的限制时，Memcg 会触发内存回收，以确保系统的稳定性和可靠性。Memcg 异步回收是一种优化机制，它可以在系统负载较低的时候，异步地回收 Memcg 中的内存，以避免在系统负载高峰期间出现内存回收的延迟和性能问题。这种机制可以提高系统的稳定性和可靠性，同时也可以提高系统的性能和响应速度。
- 支持 filescgroup：Cgroup files 子系统提供对系统中一组进程打开的文件数量（即句柄数）进行分组管理，相比于已有的 rlimit 方法，能更好的实现文件句柄数的资源控制（资源申请及释放、资源使用动态调整、实现分组控制等），并为资源管理提供方便调用的接口，实现避免某个进程打开过多文件造成整个系统资源不足无法正常工作。
- cgroupv1 使能 cgroup writeback：cgroup writeback 用于控制和管理文件系统缓存的写回行为，提供了一种灵活的方式来管理文件系统缓存的写回行为，以满足不同应用场景下的需求。它可以帮助优化系统的 IO 性能，并提供更好的资源控制和管理能力。主要功能包括：缓存写回控制、IO 优先级控制、写回策略调整等。
- 支持核挂死检测特性：解决 PMU 停止计数导致 hardlockup 无法检测系统卡死的问题，利用核间 CPU 挂死检测机制，让每个 CPU 检测相邻 CPU 是否挂死，保障系统在部分 CPU 关中断挂死场景下能够自愈。

06 特性增强

嵌入式

openEuler 发布面向嵌入式领域的版本 openEuler 23.09 Embedded（基于 5.10 内核），提供更加丰富的嵌入式软件包构建能力，支持实时 / 非实时平面混合关键部署，并集成分布式软总线。openEuler Embedded 围绕工业和机器人领域持续深耕，通过行业项目垂直打通，不断完善和丰富嵌入式技术栈和生态。openEuler 23.09 Embedded 支持嵌入式虚拟化弹性底座，提供 Jailhouse 虚拟化方案、openAMP 轻量化混合部署方案，用户可以根据自己的使用场景选择最优的部署方案。同时支持 ROS humble 版本，集成 ros-core、rosbase、SLAM 等核心软件包，满足 ROS2 运行时要求。未来 openEuler Embedded 将协同 openEuler 社区生态伙伴、用户、开发者，逐步扩展支持 RISC-V、龙芯等芯片架构，丰富工业中间件、ROS 中间件、仿真系统等能力，打造嵌入式领域操作系统解决方案。

功能描述



南向生态

openEuler Embedded Linux 当前主要支持 ARM64、x86-64 两种芯片架构，支持 RK3568、Hi3093、树莓派 4B、x86-64 工控机等具体硬件。23.09 版本新增支持 RK3399、RK3588 芯片。初步支持了 ARM32、RISC-V 两种架构通过 QEMU 仿真来体现。未来还将支持龙芯、飞腾等芯片。

嵌入式弹性虚拟化底座

openEuler Embedded 的融合弹性底座是为了在多核片上系统（SoC,System On Chip）上实现多个操作系统共同运行的一系列技术的集合，包含了裸金属、嵌入式虚拟化、轻量级容器、LibOS、可信执行环境（TEE）、异构部署等多种实现形态。不同的形态有各自的特点，例如裸金属可以得到最佳的性能、嵌入式虚拟化可以实现更好的隔离与保护、轻量级容器则有更好的易用性与灵活性等等。

- 裸金属：基于 openAMP 实现裸金属混合部署方案，支持外设分区管理，性能最好，但隔离性和灵活性较差。目前支持 UniProton/Zephyr/RT-Thread 和 openEuler 嵌入式 Linux 混合部署。
- 分区虚拟化：基于 Jailhouse 实现工业级硬件分区虚拟化方案，性能和隔离性较好，但灵活性较差。目前支持 FreeRTOS 和 openEuler 嵌入式 Linux 混合部署。
- 实时虚拟化：openEuler 社区自研虚拟化 ZVM，兼顾性能、隔离性和灵活性，综合最优。目前支持 Zephyr 和 Linux 混合部署。

混合关键性部署框架

openEuler Embedded 的混合关键性部署框架构建在融合弹性底座之上，通过一套统一的框架屏蔽下层融合弹性底座形态的不同，从而实现 Linux 和其他 OS 运行时便捷地混合部署。依托硬件上的多核能力使得通用的 Linux 和专用的实时操作系统有效互补，从而达到全系统兼具两者的特点，并能够灵活开发、灵活部署。

混合关键性部署框架的组成主要有四大部分：生命周期管理、跨 OS 通信、服务化框架和多 OS 基础设施。生命周期管理主要负责从 OS（Client OS）的加载、启动、暂停、结束等工作；跨 OS 通信为不同 OS 之间提供一套基于共享内存的高效通信机制；服务化框架是在跨 OS 通信基础之上便于不同 OS 提供各自擅长服务的框架，例如 Linux 提供通用的文件系统、网络服务，实时操作系统提供实时控制、实时计算等服务；多 OS 基础设施是从工程角度为把不同 OS 从工程上有机融合在一起的一系列机制，包括资源表达与分配，统一构建等功能。

混合关键性部署框架当前能力：

- 支持裸金属模式下 openEuler Embedded Linux 和 RTOS（Zephyr/UniProton）的生命周期管理、跨 OS 通信。
- 支持分区虚拟化模式下 openEuler Embedded Linux 和 RTOS（FreeRTOS）的生命周期管理、跨 OS 通信。
- 支持裸金属模式下在 openEuler Embedded Linux 侧通过 gdb 调试 RTOS（UniProton）。

北向生态

- 北向软件包支持：350+ 嵌入式领域常用软件包的构建。
- ROS 运行时：支持 ROS2 humble 版本，集成 ros-core、ros-base、SLAM 等核心包，并提供 ROS SDK，简化嵌入式 ROS 开发。
- 软实时内核：基于 Linux 5.10 内核提供软实时能力，软实时中断响应时延微秒级。
- 分布式软总线基础能力：集成 OpenHarmony 的分布式软总线和 hichain 点对点认证模块，实现欧拉嵌入式设备之间互联互通、欧拉嵌入式设备和 OpenHarmony 设备之间互联互通。

硬实时系统（UniProton）

UniProton 是一款实时操作系统，具备极致的低时延和灵活的混合关键性部署特性，可以适用于工业控制场景，既支持微控制器 MCU，也支持算力强的多核 CPU。目前关键能力如下：

- 支持 Cortex-M、ARM64、X86_64 架构，支持 M4、RK3568、X86_64、Hi3093、树莓派 4B 芯片和单板。
- 支持树莓派 4B、Hi3093、X86_64 设备上通过裸金属模式和 openEuler Embedded Linux 混合部署。
- 支持通过 gdb 在 openEuler Embedded Linux 侧远程调试。
- 支持 360+ POSIX 接口，支持文件系统、设备管理、shell 控制台。

应用场景

嵌入式系统可广泛应用于工业控制、机器人控制、电力控制、航空航天、汽车及医疗等领域。

GCC for openEuler

GCC for openEuler 基线版本已经从 GCC 10.3 升级到 GCC 12.3 版本，支持自动反馈优化、软硬件协同、内存优化、SVE 向量化、矢量化数学库等特性。

1. GCC 版本升级到 12.3，默认语言标准从 C14/C++14 升级到 C17/C++17 标准，支持 Armv9-a 架构，X86 的 AVX512 FP16 等更多硬件架构特性。

	GCC 10.3.0	GCC 11.3.0	GCC 12.3.0
发布时间	2021/4/8	2022/4/21	2023/5/8
C 标准	默认 c17 支持 c2x	默认 c17 支持 c2x	默认 c17 支持 c2x
C++ 标准	默认 c++14 支持 c++17 实验性 C++2a 改进 支持部分 C++20	默认 c++17 实验性 C++2a 改进 支持部分 C++20	默认 c++17 实验性 C++2a 改进 支持部分 C++20
架构新特性	armv8.6-a (bfloat16 extension/ Matrix Multiply extension) SVE2 Cortex-A77 Cortex-A76AE Cortex-A65 Cortex-A65AE Cortex-A34	armv8.6-a, +bf16, +i8mm armv8.8-r Cortex-A78 Cortex-A78AE Cortex-A78C Cortex-X1	armv8.7-a, +ls64 atomic load and store armv8.8-a, +mop, accelerate memory operations armv9-a Ampere-1 Cortex-A710 Cortex-X2 AVX512-FP16 SSE2-FP16

2. 支持结构体优化，指令选择优化等，充分使能 ARM 架构的硬件特性，运行效率更高，在 SPEC CPU 2017 等基准测试中性能大幅优于上游社区的 GCC 10.3 版本。
3. 支持自动反馈优化特性，实现应用层 MySQL 数据库等场景性能大幅提升。

功能描述

- 支 ARM 架构下 SVE 矢量化优化，在支持 SVE 指令的机器上启用此优化后能够提升程序运行的性能。
- 支持内存布局优化，通过重新排布结构体成员的位置，使得频繁访问的结构体成员放置于连续的内存空间上，提升 Cache 的命中率，提升程序运行的性能。
- 支持冗余成员消除优化，消除结构体中从不读取的结构体成员，同时删除冗余的写语句，缩小结构体占用内存大小，降低内存带宽压力，提升性能。
- 支持数组比较优化，实现数组元素并行比较，提高执行效率。
- 支持 ARM 架构下指令优化，增强 ccmp 指令适用场景，简化指令流水。
- 支持自动反馈优化，使用 perf 收集程序运行信息并解析，完成编译阶段和二进制阶段反馈优化，提升 MySQL 数据库等主流应用场景的性能。

应用场景

通用计算领域，运行 SPECCPU 2017 测试，相比于上游社区的 GCC 10.3 版本可获得 20% 左右的性能收益。

其他场景领域，使能自动反馈优化后，MySQL 性能提升 15% 以上；使能内核反馈优化后，实现 Unixbench 性能提升 3% 以上。

Kmesh 项目

随着 AI、直播等大应用的发展及传统应用云化改造的深入，数据中心集群规模越来越大、应用类型也越来越丰富，如何实现集群内服务间的高效互通、满足应用 SLA 诉求已成为数据中心面临的关键问题，对云基础设施提出了很高的要求。

基于 K8S 的云基础设施能够帮助应用实现敏捷的部署管理，但在应用流量编排方面有所欠缺，服务网格的出现很好的弥补了 K8S 流量编排的缺陷，与 K8S 互补，真正实现敏捷的云应用开发运维。但随着对服务网格应用的逐步深入，当前服务网格的代理架构，数据面引入了额外的时延底噪开销，已成为业界共识的性能问题。

时延

以服务网格典型软件 istio 为例，网格化后，服务访问单跳时延增加 2.65ms，无法满足时延敏感型应用诉求。

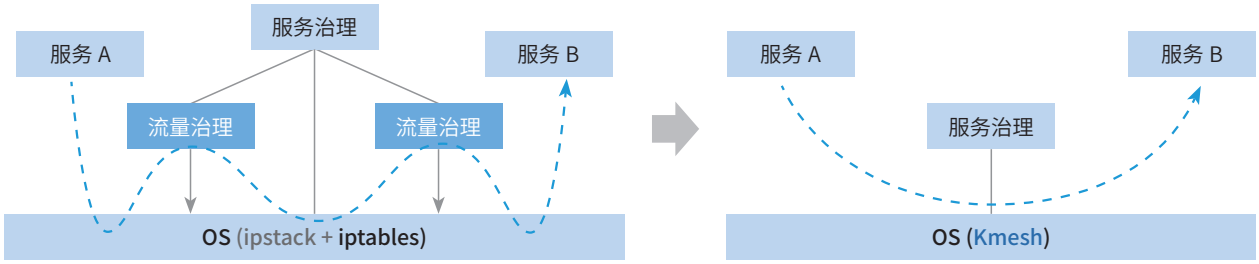
底噪

istio 中，每个 sidecar 软件占用内存 50M+，CPU 默认独占 2 core，对于大规模集群底噪开销太大，降低了业务容器的部署密度。

Kmesh 基于可编程内核，将服务治理下沉 OS，实现高性能服务网格数据面，服务间通信时延对比业界方案提升 5 倍。

Kmesh

基于可编程内核，将流量治理下沉 OS，实现流量路径多跳变一跳

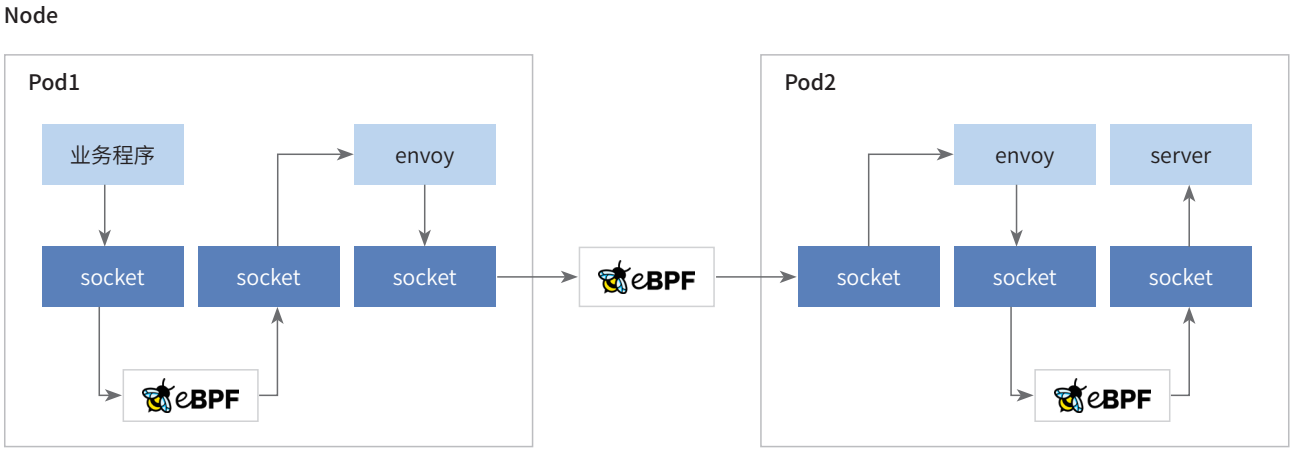


业界网格：路径过长导致时延性能 X 倍增长

Kmesh: 流量路径多跳变一跳

功能描述

- 支持对接遵从 XDS 协议的网格控制面（如 istio）
- 流量编排能力
  - 负载均衡：支持轮询等负载均衡策略。
  - 路由：支持 L4、L7 路由规则。
  - 灰度：支持百分比灰度方式选择后端服务策略。
- sockamp 网格加速能力：以典型的 service mesh 场景为例，使能 sockmap 网格加速能力之后，业务容器和 envoy 容器之间的通信将被 ebpf 程序短接，通过缩短通信路径从而达到加速效果，对于同节点上 Pod 间通信也能通过 ebpf 程序进行加速。



注：

1. 使能 sockmap 网格加速能力后创建的数据连接才会被加速，已经建立的连接不会被加速。
2. 当前仅支持同节点上 ipv4 tcp 连接的通信加速，对于跨节点的 ipv4 tcp 连接通信，会存在 10%~20% 的时延性能损耗。

应用场景

服务网格场景：优化云原生服务网格内服务通信性能。例如电商、计费、金融、物流、短视频、在线会议、云游戏等对服务时延敏感的应用。

相关使用方式请参考 Kmesh 使用介绍。

sysMaster 相关特性

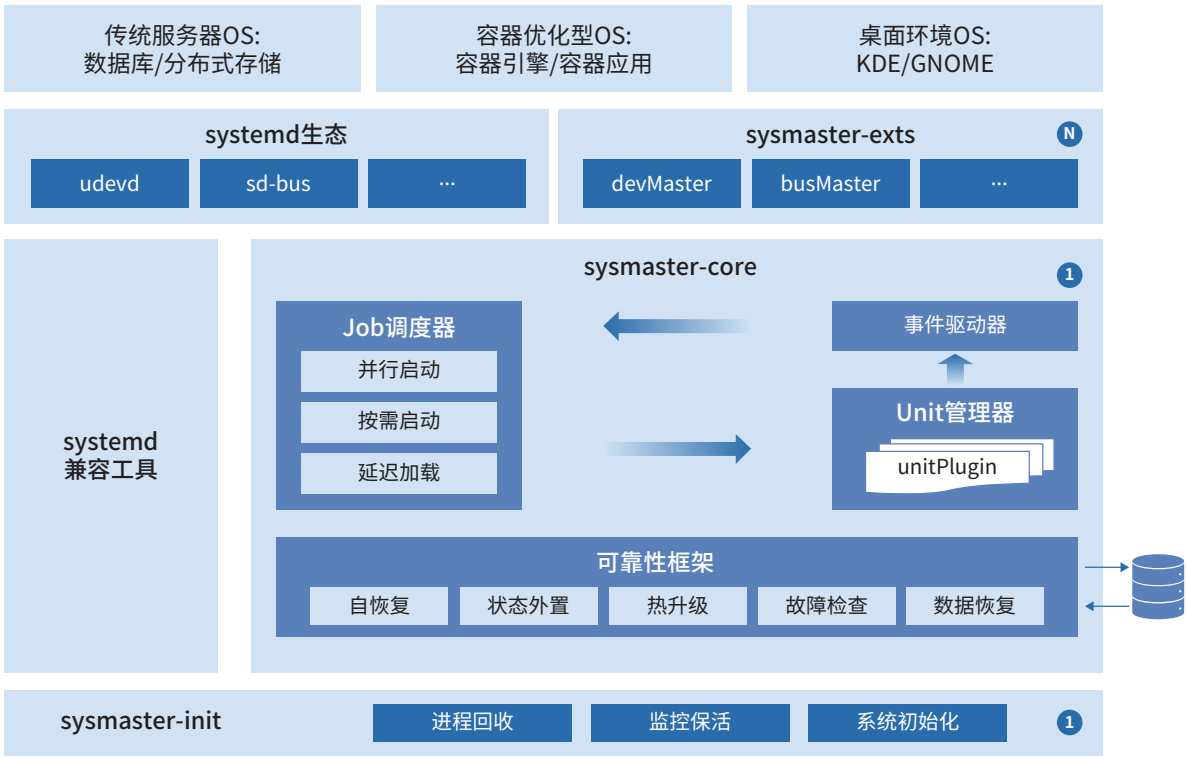
sysMaster 是一套超轻量、高可靠的服务管理程序集合，是对 1 号进程的全新实现，旨在改进传统的 init 守护进程。它使用 Rust 编写，具有故障监测、秒级自愈和快速启动等能力，从而提升操作系统可靠性和业务可用度。

sysMaster 支持进程、容器和虚拟机的统一管理，并引入了故障监测和自愈技术，从而解决 Linux 系统初始化和服务管理问题，致力于替代现有 1 号进程，其适用于服务器、云计算和嵌入式等多个场景。

功能描述

sysMaster 实现思路是将传统 1 号进程的功能解耦分层，结合使用场景，拆分出 1+1+N 的架构。如下面 sysMaster 系统架构图所示，主要包含三个方面。

1. sysmaster-init：新的 1 号进程，功能极简，代码千行，极致可靠，提供系统初始化 / 僵尸进程回收 / 监控保活等功能，可单独应用于嵌入式场景。
2. sysmaster-core：承担原有服务管理的核心功能，引入可靠性框架，使其具备崩溃快速自愈、热升级等能力，保障业务全天在线。
3. sysmaster-exts：使原本耦合的各组件功能独立，提供系统关键功能的组件集合（如设备管理 devMaster 等），各组件可单独使用，可根据不同场景灵活选用。



sysMaster 组件架构简单，提升了系统整体架构的扩展性和适应性，从而降低开发和维护成本。其主要特点如下：

- 支持服务管理、设备管理等功能，具有自身故障秒级自愈和版本热升级能力。
- 具备快速启动的能力，更快的启动速度和更低的运行底噪。
- 提供迁移工具，支持从 Systemd 快速无缝迁移到 sysMaster。
- 结合容器引擎 (iSulad) 和 Qemu，提供统一的容器实例和虚拟化实例的管理接口。

本次发布的 0.5.0 版本，支持在容器、虚机两种场景下，以 sysMaster 的方式管理系统中的服务。

新增特性：

- 新增支持 devMaster 组件，用于管理设备热插拔。
- 新增支持 sysMaster 热升级、热重启功能。
- 新增支持在虚机中以 1 号进程运行。

约束限制：

- 当前仅支持 64 位系统。
- 当前仅支持 sysMaster 使用的 toml 配置格式。
- 当前仅支持系统容器和虚拟机两种使用场景。

未来，sysMaster 将继续探索在多场景下的应用，并持续优化架构和性能以提高可扩展性和适应性。同时，我们还将开发新的功能和组件以满足容器化、虚拟化、边缘计算等场景的需求。让 sysMaster 成为一个强大的系统管理框架，为用户提供更好的使用体验和更高的效率。

应用场景

sysMaster 致力于替代容器、虚机、服务器及边缘设备上现有 1 号进程。

SysCare 热补丁能力

在 Linux 世界，有一个困扰大家已久的难题：如何在不影响业务的情况下，快速可靠地修复漏洞、解决故障。

当前常见的方法是采用热补丁技术：在业务运行过程中，对问题组件直接进行代码级修复，业务无感知。然而，当前热补丁制作方式复杂，补丁需要代码级匹配，且管理困难，特别是用户态组件面临文件形式、编程语言、编译方式、运行方式的多样性问题，当前还没有简便统一的补丁机制。

为了解决热补丁制作和管理的问题，SysCare 应运而生。

SysCare 是一个系统级热修复软件，为操作系统提供安全补丁和系统错误热修复能力，主机无需重新启动即可修复该系统问题。 SysCare 将内核态热补丁技术与用户态热补丁技术进行融合统一，用户仅需聚焦在自己核心业务中，系统修复问题交予 SysCare 进行处理。后期计划根据修复组件的不同，提供系统热升级技术，进一步解放运维用户提升运维效率。

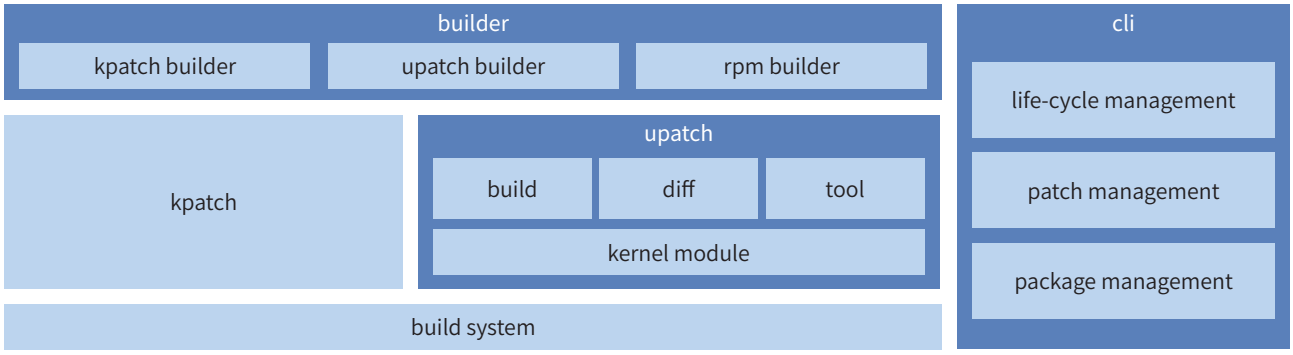
功能描述

热补丁制作

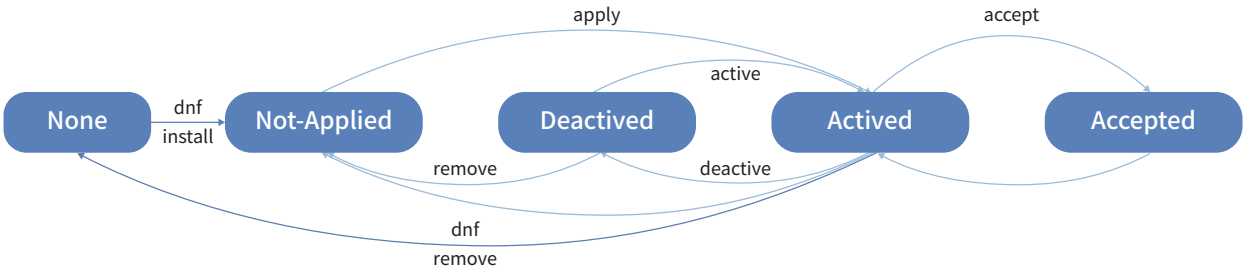
用户仅需输入目标软件的源码 RPM 包、调试信息 RPM 包与待打补丁，无需对软件源码进行任何修改，即可生成对应的热补丁 RPM 包。

热补丁生命周期管理

SysCare 提供一套完整的，傻瓜式补丁生命周期管理方式，旨在减少用户学习、使用成本，通过单条命令即可对热补丁进行管理。依托于 RPM 系统，SysCare 构建出的热补丁依赖关系完整，热补丁分发、安装、更新与卸载流程均无需进行特殊处理，可直接集成放入软件仓 repo。



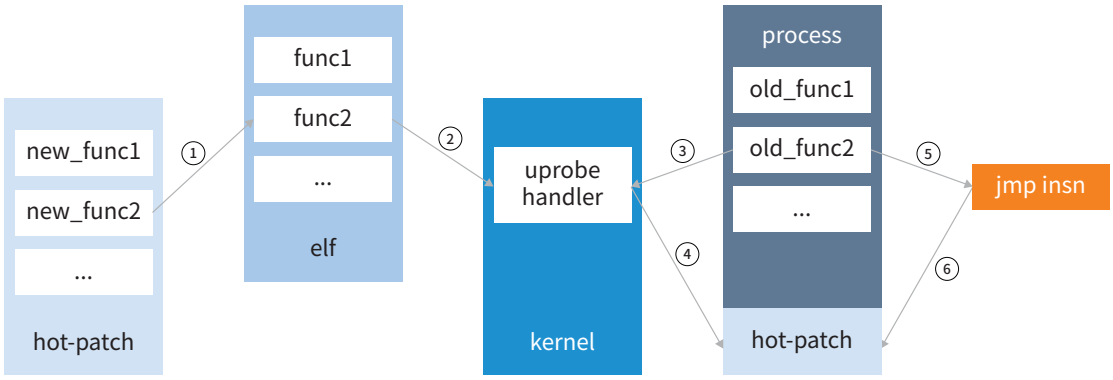
SysCare 整体架构



热补丁生命周期

针对 ELF 文件（程序可执行文件）的用户态热补丁

使用 uprobe 技术，将热补丁与 ELF 文件绑定。在 ELF 文件运行时，通过 uprobe 触发补丁生效，这样无需监控进程。因此，无论进程是否已经运行都可以在打补丁后或新进程运行时使补丁生效。同时，该技术也可以给动态库打热补丁，解决了动态库热补丁的难题。补丁生效流程如下图所示。

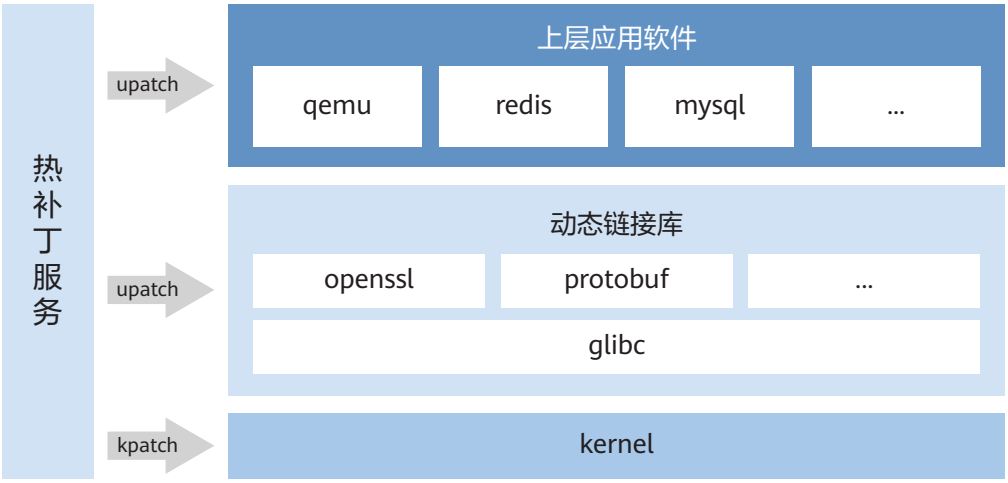


补丁生效流程

1. 执行 uprobe 系统调用，在待修改函数 func 处增加 uprobe 断点。
2. 注册 uprobe handler。
3. 进行运行到 func 时调用 uprobe handler。
4. 将 patch 映射到当前进程地址空间。
5. 进行安全检查并将 func 的第一条指令修改为 jump 指令，指向 patch 地址。
6. 跳转到 patch 地址执行。

内核热补丁与用户态热补丁融合

SysCare 基于 upatch 和 kpatch 技术，覆盖应用、动态库、内核，自顶向下打通热补丁软件栈，提供用户无感知的全栈热修复能力。



热补丁应用范围

新增特性

支持容器内构建补丁

- 通过使用 ebpf 技术监控编译器进程，实现无需创建字符设备、纯用户态化获取热补丁变化信息，并允许用户在多个不同容器内进行并行热补丁编译。
- 用户可以通过安装不同 rpm 包（syscare-build-kmod 或 syscare-build-ebpf）来选择使用 ko 或者 ebpf 实现，syscare-build 进程将会自适应相应底层实现。

约束限制

- 当前仅支持 64 位系统。
- 当前仅支持 ELF 格式的热修复，不支持解释型语言，不支持纯汇编修改。
- 当前仅支持 GCC / G++ 编译器，且不支持交叉编译。
- 暂不支持 LTO 优化。

应用场景

应用场景 1： CVE 补丁快速修复。

应用场景 2： 现网问题临时定位。

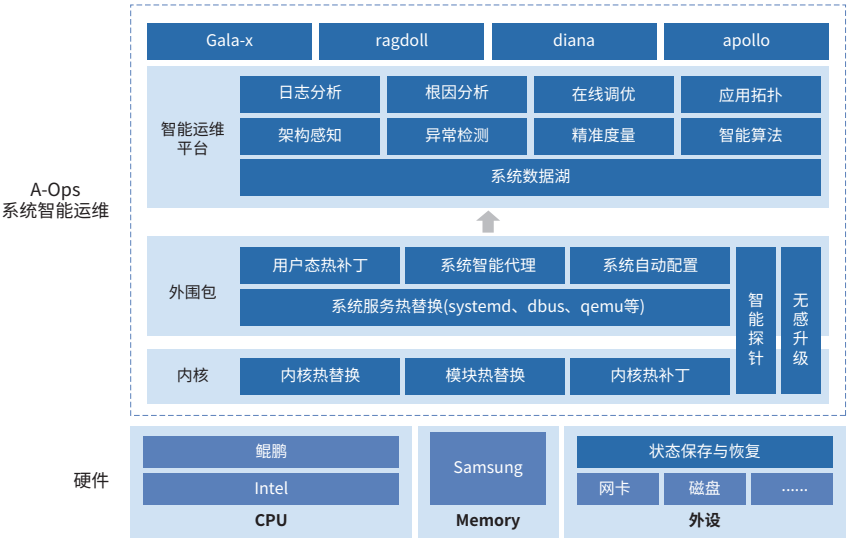
A-Ops 智能运维

IT 基础设施和应用产生的数据量快速增长（每年增长 2~3 倍），应用大数据和机器学习技术日趋成熟，驱动高效智能运维系统产生，助力企业降本增效。openEuler 智能运维提供智能运维基本框架，支持 CVE 管理、异常检测（数据库场景）等基础能力，支持快速排障和运维成本降低。

功能描述

智能补丁管理

- 补丁服务：提供冷热补丁发布，支持按照 CVE 直接获取补丁。
- 内核热修复：在不重启任何进程和机器的情况下对内核实现缺陷修复。
- 智能补丁巡检：提供单机 / 集群的 CVE 巡检和通知能力，支持一键式修复和回退，极大减少补丁管理成本，保障集群安全，提升漏洞修复效率。
- 冷热补丁混合管理：在系统兼容的情况下，自动实现补丁收编，减少在网补丁数量，减轻集群运维压力。



A-ops 软件架构

异常检测

智能故障诊断：提供 MySQL、openGauss 业务场景中出现的网络 I/O 时延、丢包、中断等故障以及磁盘 I/O 高负载故障检测能力。

配置溯源

- 配置基线：支持集群配置收集和基线能力，实现配置可管可控。
- 配置异常检查：对整体集群实现配置检查，实时与基线进行对比，快速识别未经授权的配置变更，实现故障快速定位。

应用场景

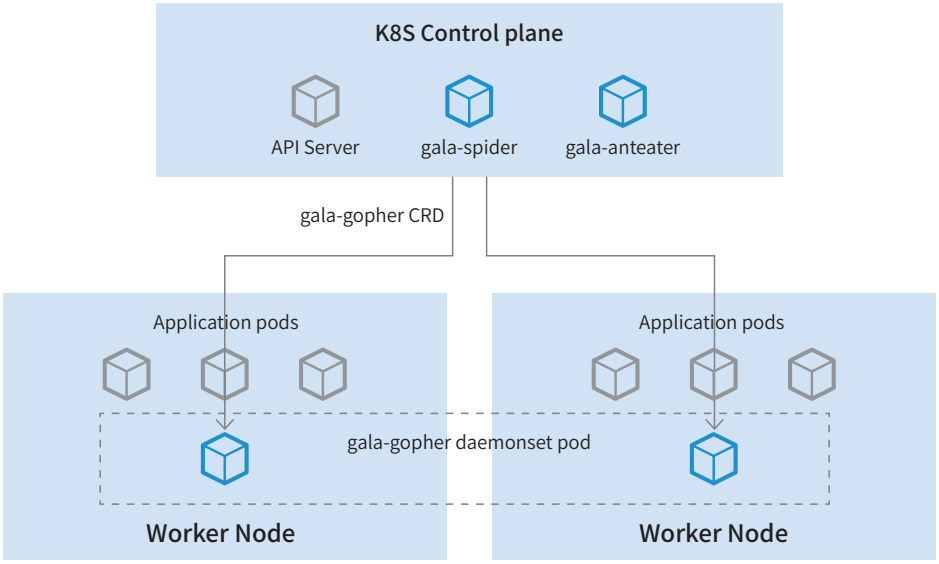
A-Ops 通过挂载社区 CVE 漏洞 repo 源，进行 CVE 漏洞巡检，使用冷热补丁发布件（rpm 包）进行修复、回退和收编等操作，提升运维效率。

A-Ops gala 相关特性

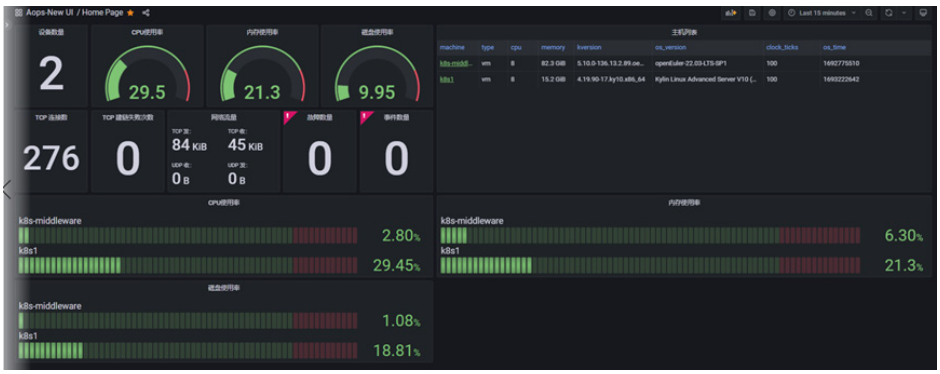
GALA 项目将全面支持 K8S 场景故障诊断，提供包括应用 drill-down 分析、微服务 &DB 性能可观测、云原生网络监控、云原生性能 Profiling、进程性能诊断等特性，支撑 OS 五类问题（网络、磁盘、进程、内存、调度）分钟级诊断。

功能描述

- K8S 环境易部署：gala-gopher 提供 daemonset 方式部署，每个 Worker Node 部署一个 gala-gopher 实例；gala-spider、gala-anteater 以容器方式部署至 K8S 管理 Node。



- 应用 drill-down 分析：提供云原生场景中亚健康问题的故障诊断能力，分钟级完成应用与云平台之间问题定界能力。
- 全栈监控：提供面向应用的精细化监控能力，覆盖语言运行时 (JVM)、GLIBC、系统调用、内核 (TCP、I/O、调度等) 等跨软件栈观测能力，实时查看系统资源对应用的影响。
- 全链路监控：提供网络流拓扑 (TCP、RPC)、软件部署拓扑信息，基于这些信息构建系统 3D 拓扑，精准查看应用依赖的资源范围，快速识别故障半径。
- GALA 因果型 AI：提供可视化根因推导能力，分钟级定界至资源节点。



- 微服务 &DB 性能可观测：提供非侵入式的微服务、DB 访问性能可观测能力，包括 HTTP 1.x 访问性能可观测，性能包括吞吐量、时延、错误率等，支持 API 精细化可观测能力，以及 HTTP Trace 能力，便于查看异常 HTTP 请求过程。
- PGSQL 访问性能可观测：性能包括吞吐量、时延、错误率等，支持基于 SQL 访问精细化观测能力，以及慢 SQL Trace 能力，便于查看慢 SQL 的具体 SQL 语句。

- 云原生应用性能 Profiling：提供非侵入、零修改的跨栈 profiling 分析工具，并能够对接 pyroscope 业界通用 UI 前端。技术特点包括：

  1) 底噪低：benchmark 测试场景，对应用干扰 <2%。

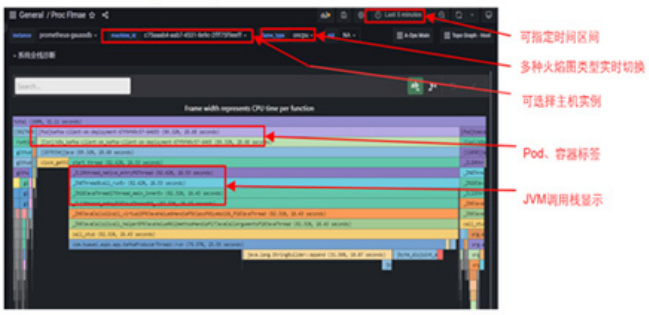
  2) 多语言：支持常见开发语言 C/C++、Go、Rust、Java。

  3) 多实例：支持同时监控多个进程或容器，UI 前端可以对比性分析问题原因。

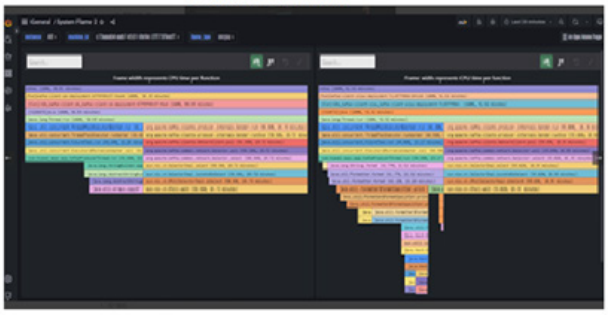
  4) 细粒度：支持指定 profiling 范围，包括进程、容器、Pod。

  5) 多维度：提供 OnCPU、OffCPU、MemAlloc 不同维度的应用性 Profiling。

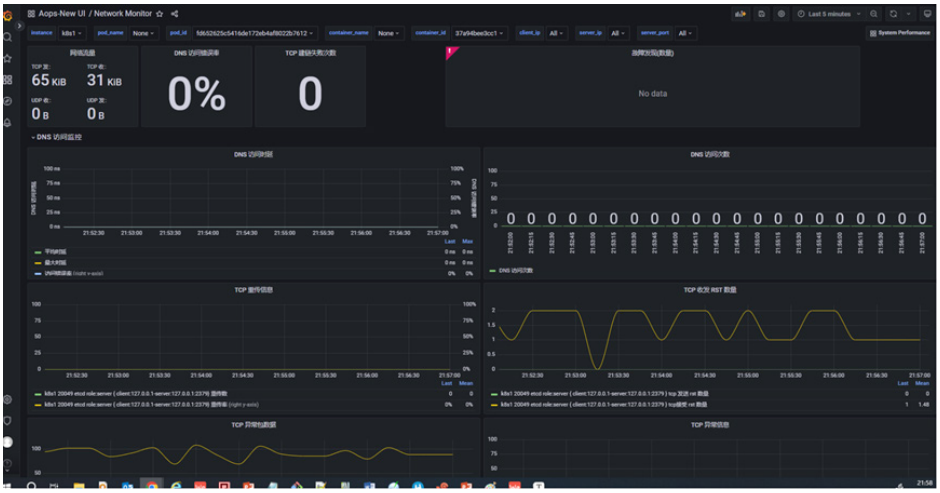
Pyroscope UI界面使用介绍



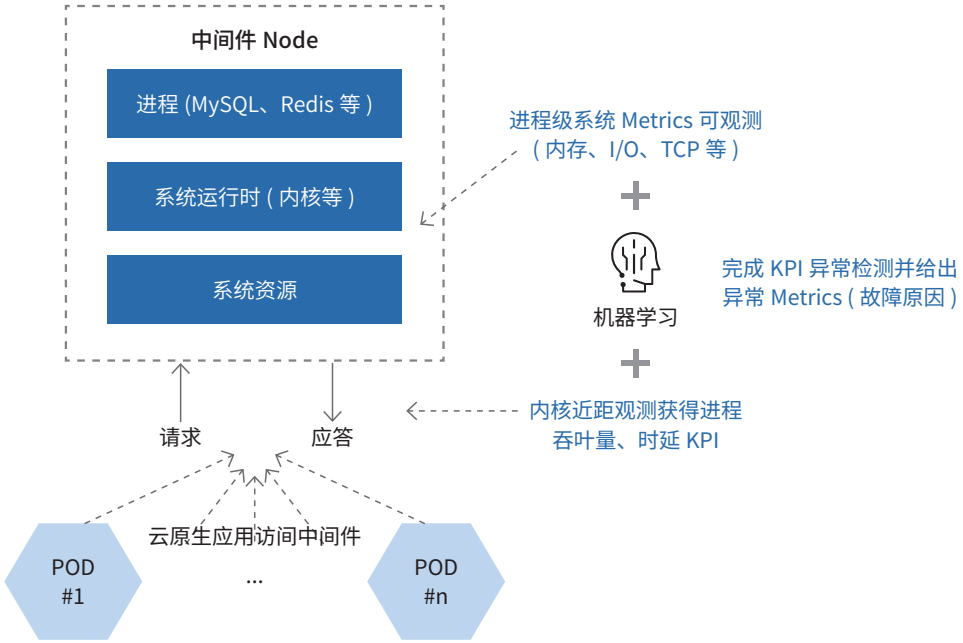
多实例对比效果图



- 云原生网络监控：针对 K8S 场景，提供 TCP、Socket、DNS 监控能力，具备更精细化网络监控能力。



- 进程性能诊断：针对云原生场景的中间件（比如 MySQL、Redis 等）提供进程级性能问题诊断能力，同时监控进程性能 KPI、进程相关系统层 Metrics（比如 I/O、内存、TCP 等），完成进程性能 KPI 异常检测以及影响该 KPI 的系统层 Metrics（影响进程性能的原因）。

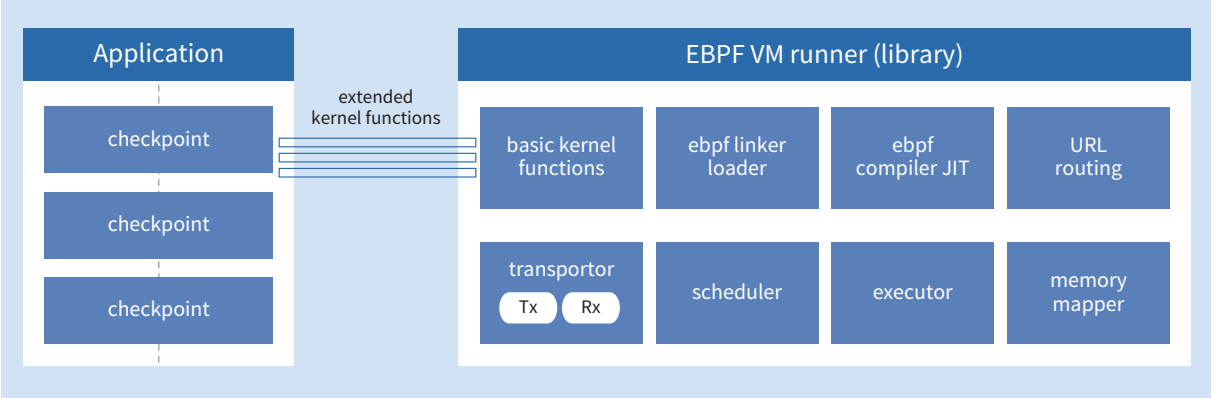


应用场景

相关使用方式请参考 gala-gopher daemonset 部署介绍和 REST 配置介绍。

CTinspector 项目

CTinspector 是天翼云科技有限公司基于 ebpf 指令集自主创新研发的语言虚拟机运行框架。基于 CTinspector 运行框架可以快速拓展其应用实例用于诊断网络性能瓶颈点，诊断存储 I/O 处理的热点和负载均衡等，提高系统运行时诊断的稳定性和时效性。

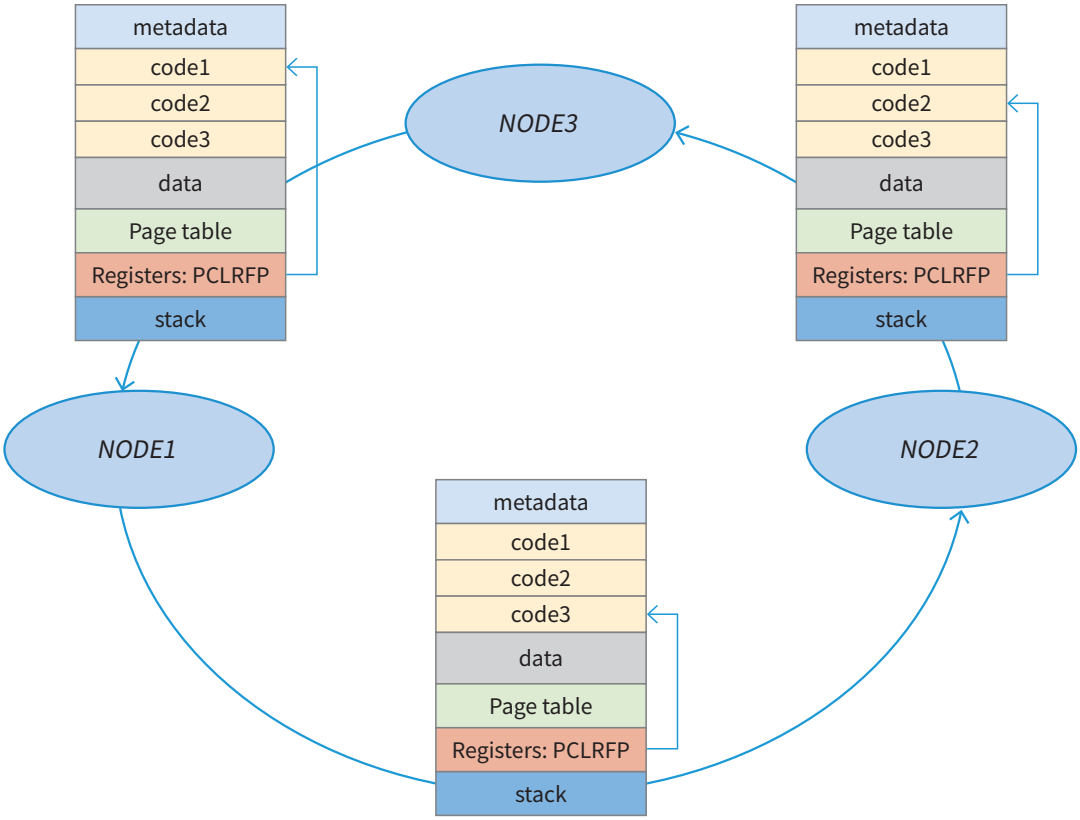


CTinspector 框架的主要部件包括：

- ebpf compiler/JIT：将 C 代码编译为 ebpf 二进制码，JIT 则负责将 ebpf 指令及时编译为机器码。
- ebpf linker/loader：负责加载和链接库函数即 kernel functions。
- runner：执行 ebpf VM，这包括加载寄存器、代码段、加载堆栈、映射数据段等。
- scheduler：决定何时执行 ebpf VM，这包括判断 VM 的状态，需要等待的数据依赖条件等。
- basic kernel functions：基本库函数，包括迁移、映射内存、fork、join_meeting 等核心基本功能。
- extended kernel functions：除了 ebpf VM runner 提供的核心基本功能外，应用程序的各个 hook 点都可以提供自定义的库函数。
- memory mapper：将应用程序数据映射进 ebpf VM 以方便 ebpf 程序读写应用数据。

功能描述

CTinspector 采用一个 ebpf 指令集的语言虚拟机 Packet VM，它最小只有 256 字节，包含所有虚拟机应有的部件：寄存器，堆栈段，代码段，数据段，页表。Packet VM 支持自主的 migration，即 packet VM 内的代码可以调用 migrate kernel function，以将 packet VM 迁移至它自己指定的节点。Packet VM 同时支持断点续执行，即 packet VM 迁移至下一个节点后可以沿着上一个节点中断的位置继续执行下一条指令。此功能对于需要在网络拓扑不同节点中有状态的执行命令起到重要作用。



应用场景

OVS 的连接跟踪 CT 流表最大可以达到 10M 级别，每一条流表的字段多达 20 几个。运维人员在网络故障时经常需要 dump OVS 的流表查看问题在哪，但是流表太多根本不方便查看，另外 dump 大量的流表也耗时甚多，这就需要对流表进行过滤。CTinspector 可应用于调查分析虚拟网络 OVS 流表。

传统 ACL 首先的问题是配置复杂且不直观，一个报文最终是 ACCEPT 还是 DROP 需要把全部 chain 的规则看完才能知道。第二个问题是 ACL 实现起来也比较复杂，特别是范围比较，掩码匹配等都需要特殊处理，命令行的解析就是一个很麻烦的工作。基于 CTinspector 可以开发应用实例应用于 ACL 规则的配置和下发。

CTinspector 断点执行命令的特性可应用于网络连通性测试，诊断网络利用率性能指标。

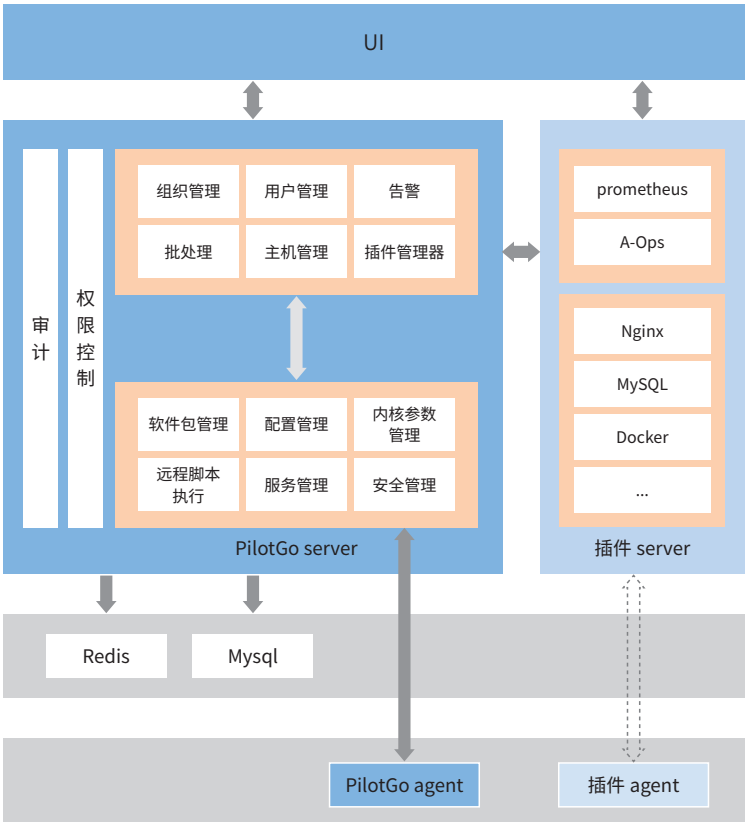
增加 PilotGo 运维管理平台特性

PilotGo 是 openEuler 社区原生孵化的运维管理平台，采用插件式架构设计，功能模块轻量化组合、独立迭代演进，同时保证核心功能稳定；同时使用插件来增强平台功能，并打通不同运维组件之间的壁垒，实现了全局的状态感知及自动化流程。

功能描述

PilotGo 核心功能模块包括：

- 用户管理：支持按照组织结构分组管理，支持导入已有平台账号，迁移方便。
- 权限管理：支持基于 RBAC 的权限管理，灵活可靠。
- 主机管理：状态前端可视化、直接执行软件包管理、服务管理、内核参数调优、简单易操作。
- 批次管理：支持运维操作并发执行，稳定高效。
- 日志审计：跟踪记录用户及插件的变更操作，方便问题回溯及安全审计。
- 告警管理：平台异常实时感知。
- 插件功能：支持扩展平台功能，插件联动，自动化能力倍增，减少人工干预。



当前 OS 发布版本还集成了以下插件：

- Prometheus：托管 Prometheus 监控组件，自动化下发及配置 node-exporter 监控数据采集，对接平台告警功能。
- Grafana：集成 Grafana 可视化平台，提供美观易用的指标监控面板功能。

应用场景

PilotGo 可用于典型的服务器集群管理场景，支持大批量的服务器集群基本管理及监控；通过集成对应的业务功能插件，还可实现业务集群的统一平台管理，例如 MySQL 数据库集群、Redis 数据缓存集群、Nginx 网关集群等。

CPDS 支持对容器 TOP 故障、亚健康检测的监测与识别

云原生技术的广泛应用，致使现代应用部署环境越来越复杂。容器架构提供了灵活性和便利性，但也带来了更多的监测和维护挑战。CPDS（容器故障检测系统）应运而生，旨在为容器化应用提供可靠性和稳定性的保障。

功能描述

集群信息采集

在宿主机上实现节点代理，采用 systemd、initv、ebpf 等技术，对容器关键服务进行监 控，采集集群基础服务类数据；对节点网络、内核、磁盘 LVM 等相关信息进行监控，采集集群 OS 类异常数据；采用无侵入的方式在节点、容器内设置跨 NS 的代理，针对对应用状态、资源消耗情况、关键系统函数执行情况、IO 执行状态等执行异常进行监控，采集业务服 务异常类数据。

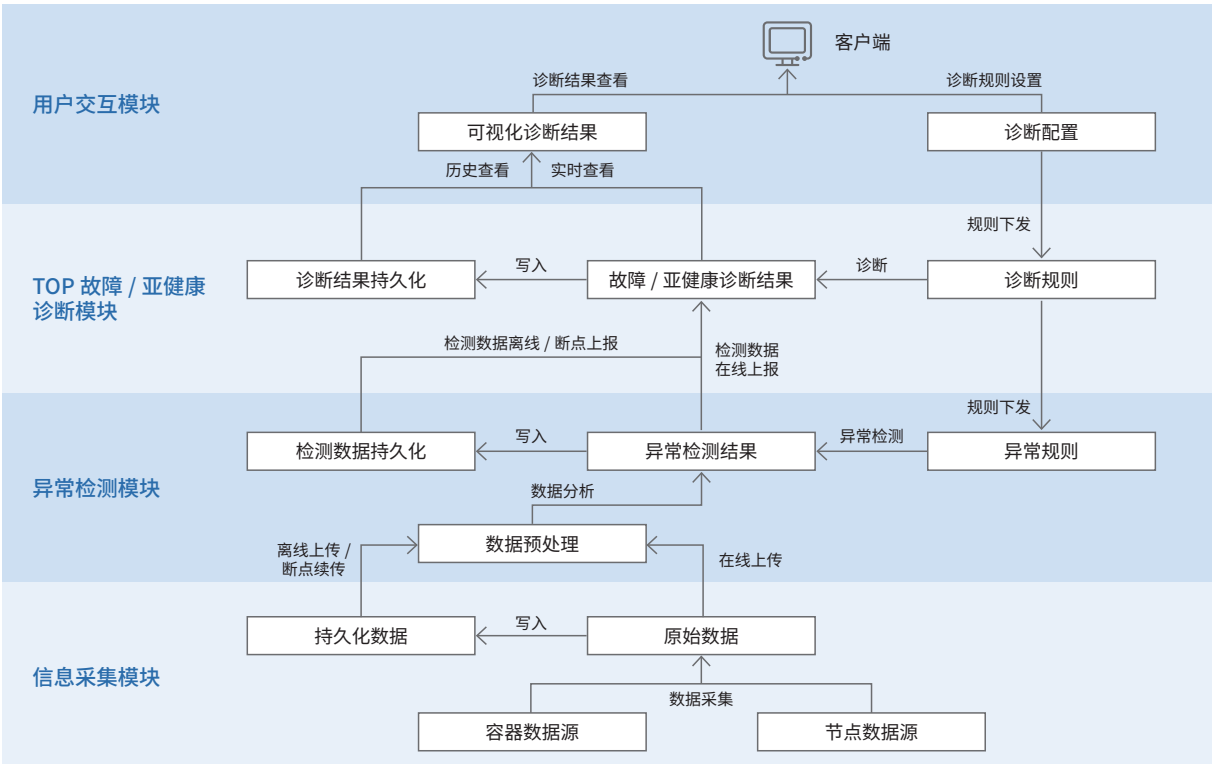
集群异常检测

处理各节点原始数据，基于异常规则对采集的原始数据进行异常检测，提取关键信息。 同时基于异常规则对采集数据进行异常检测，后将检测结果数据和原始据进行在线上传，并 同步进行持久化操作。

节点、业务容器故障 / 亚健康诊断

基于异常检测数据，对节点、业务容器进行故障 / 亚健康诊断，将分析检测结果进行持久化存储，并提供 UI 层进行实时、历史的诊断数据查看。

系统架构图如下：



应用场景

随着企业数字化发展浪潮及云原生技术的普及，低时延和高并发的线上场景频繁出现在企业日常经营中，业务创新的需求也在倒逼企业不断运用新兴技术手段。现如今，容器技术被广泛应用于人工智能、大数据、边缘计算等场景，作为轻量化的计算载体，为更多的场景 赋予高度的弹性与敏捷性。

业务规模的增长，容器集群规模不断扩张，IT 运维压力也成比例增大。各种软、硬件故 障而造成的业务中断，成为稳定性影响的重要因素之一。目前业内对容器集群故障的检测方 案主要基于集群组件状态检测、服务入口监控、自定义接口检活等，具有一定的局限性，难 以对服务的亚健康状态进行检测与识别。处理方式也缺乏故障的诊断与执行策略的制定，难 以处理一些关键、核心故障。

CPDS（容器故障检测系统）在各种容器化应用和基础设施的管理、监控和故障处理中都有着重要的应用价值。利用 CPDS 可以监控节点、业务容器的状态、资源使用情况和性能 指标，协助团队快速响应和修复潜在的故障，提高系统的稳定性、可用性和维护效率，保障 应用的可靠运行，解决了行业内的一个核心痛点问题。

CPDS 现支持对以下故障项进行检测：

序号	故障检测项
1	容器服务是否正常
2	容器节点代理是否正常
3	容器组是否正常
4	节点健康检测是否正常
5	日志采集是否正常
6	磁盘用量占容量 85%
7	网络故障
8	内核 Crash 故障
9	残留 LVM 盘故障
10	CPU 使用率超过 85%
11	节点监控是否正常
12	容器内存申请失败
13	容器内存申请超时
14	容器网络响应超时
15	容器磁盘读写缓慢
16	容器应用僵尸子进程监测
17	容器应用占用子进程、线程创建失败监测

utsudo 项目发布

Sudo 是 Unix 和 Linux 操作系统中常用的工具之一，它允许用户在需要超级用户权限的情况下执行特定命令。然而，传统 Sudo 在安全性和可靠性方面存在一些缺陷，为此 utsudo 项目应运而生。

utsudo 是一个采用 Rust 重构 Sudo 的项目，旨在提供一个更加高效、安全、灵活的提权工具，涉及的模块主要有通用工具、整体框架和功能插件等。

功能描述

- 访问控制：可以根据需求，限制用户可以执行的命令，并规定所需的验证方式。
- 审计日志：可以记录和追踪每个用户使用 utsudo 执行的命令和任务。
- 临时提权：允许普通用户通过输入自己的密码，临时提升为超级用户执行特定的命令或任务。
- 灵活配置：可以设置参数如命令别名、环境变量、执行参数等，以满足复杂的系统管理需求。

应用场景

通过部署 utsudo，管理员能够更好地管理用户权限，确保合适的授权级别，防止未经授权的特权操作，从而降低安全风险。 常见的应用场景有：系统管理维护、用户权限控制、多用户环境等。

utshell 项目

utshell 是一个延续了 bash 使用习惯的全新 shell，它能够与用户进行命令行交互，响应用户的操作去执行命令并给予反馈。并且能执行自动化脚本帮助运维。

功能描述

utshell 功能具体如下：

- 命令执行：可以执行部署在用户机器上的命令，并将执行的返回值反馈给用户。
- 批处理：通过脚本完成自动任务执行。
- 作业控制：能够将用户命令作为后台作业，从而实现多个命令同时执行。并对并行执行的任务进行管理和控制。
- 历史记录：记录用户所输入的命令。
- 别名功能：能够让用户对命令起一个自己喜欢的别名，从而个性化自己的操作功能。

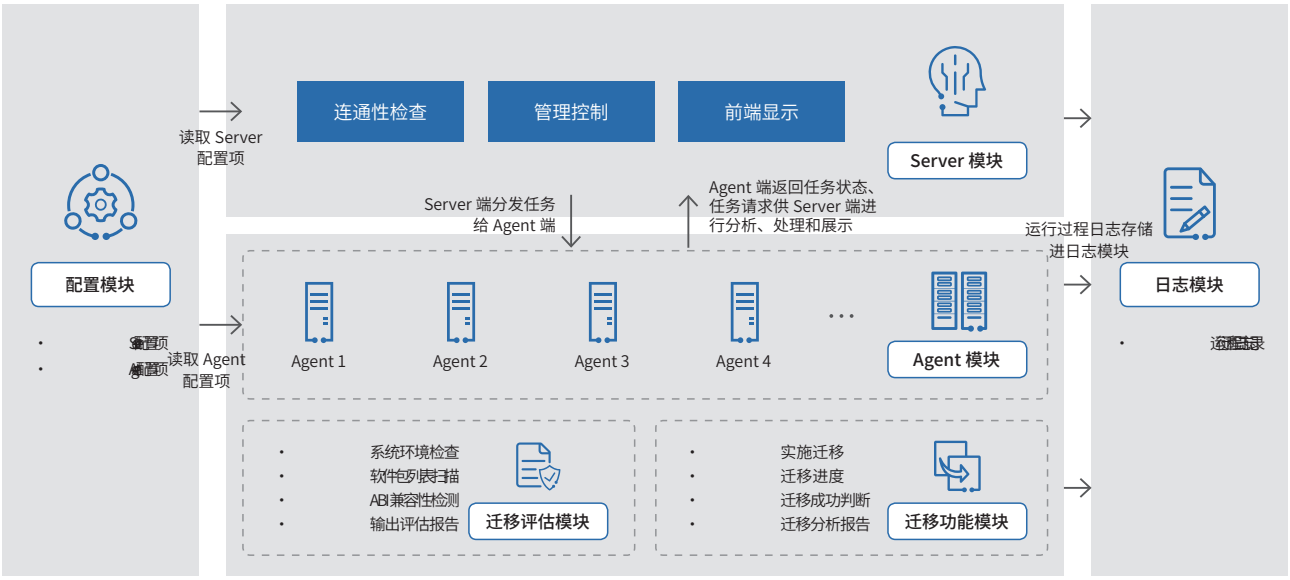
应用场景

utshell 适用于传统服务器系统以及云原生环境，有人值守或无人值守下自动化脚本运维的生产环境。也适用于桌面命令行爱好者的日常使用。

migration-tools 项目（统信）

migration-tools 是由统信软件开发的一款操作系统迁移软件，面向已部署业务应用于其他操作系统且具有国产化替换需求的用户，帮助其快速、平滑、稳定且安全地迁移至 openEuler 系操作系统。

功能描述



迁移软件的系统架构分为：Server 模块、Agent 模块、配置模块、日志模块、迁移评估模块、迁移功能模块。

- Server 模块：Server 模块为迁移的软件的核心，采用 pythonflaskweb 框架研发，负责接收任务请求，同时处理相关执行指令并分发至各 Agent。
- Agent 模块：Agent 模块安装在待迁移的操作系统中，负责接收 Server 发出的任务请求，执行迁移等功能。
- 配置模块：为 Server 模块和 Agent 模块提供配置文件的读取功能。
- 日志模块：提供迁移的全部运行过程记录日志。
- 迁移评估模块：提供迁移前的基础环境检测、软件包对比分析、ABI 兼容性检测等评估报告，为用户的迁移工作提供依据。
- 迁移功能模块：提供一键迁移、迁移进度展示、迁移结果判断等功能。

应用场景

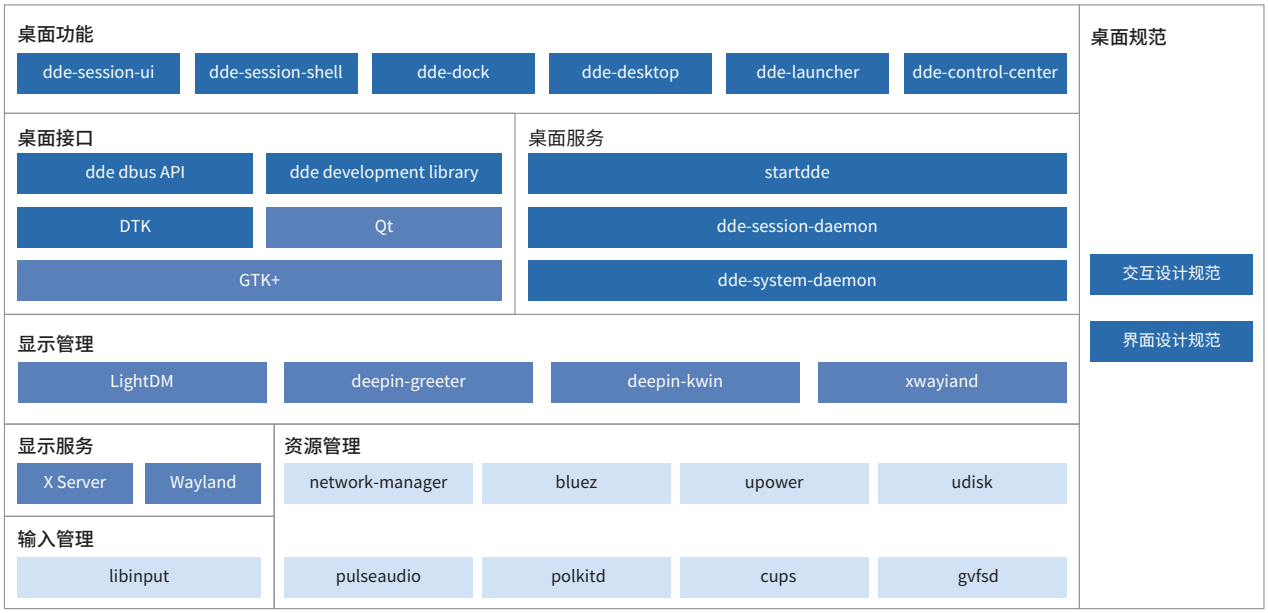
在金融、电信、能源等关键行业，涉及大量存量硬件设备（AMD64 等架构）中操作系统的国产化替代，需要将原存量操作系统中的应用软件、系统组件迁移至 openEuler 操作系统中时，都可以使用 migration-tools 进行迁移。

DDE 组件

统信桌面环境（DDE）是统信软件为统信操作系统（UniontechOS）开发的一款桌面环境，统信桌面操作系统、统信操作系统服务器版和统信操作系统专用设备版均在使用统信桌面环境。

功能描述

统信桌面环境专注打磨产品交互、视觉设计，拥有桌面环境的核心技术，主要功能包含：登录锁屏、桌面及文件管理器、启动器、任务栏（DOCK）、窗口管理器、控制中心等。由于界面美观、交互优雅、安全可靠、尊重隐私，一直是用户首选桌面环境之一，用户可以使用它进行办公与娱乐，在工作中发挥创意和提高效率，和亲朋好友保持联系，轻松浏览网页、享受影音播放。



统信桌面环境的核心技术是拥有统一界面元素设计、讲究细节交互设计的 DTK 框架及 Qt、GTK+ 等三方图形库。

显示服务、输入管理、资源管理较为底层，一般是基于 golang 开发的后端服务，为上层 GUI 程序提供桌面环境中所需功能接口，如创建用户、设置屏幕亮度、设置设备音量、管理网络连接等功能。

显示管理、桌面接口、桌面服务属于 shell 层，一般是基于 DBus 接口协议与后端服务进行通信，为定义用户界面、交互操作提供支撑，如登录界面、窗口外观、GUI 应用程序控件等。

应用场景

桌面功能属于应用层，一般是面向用户可操作的功能界面，比如启动器、任务栏（DOCK）等。

RISC-V 架构 QEMU 镜像

openEuler 提供了针对 RISC-V 架构的基础镜像，以满足该架构环境下的操作系统需求。RISC-V 作为一种开放、免费、可定制的指令集架构，具备简洁、清晰的设计、高度可定制性和可移植性的特点。近年来，RISC-V 架构快速发展，扩大了应用领域，其生态系统也迅速扩展。在嵌入式系统、高性能计算、云计算和学术研究等领域具有广泛应用。

openEuler 对 RISC-V 架构的支持始于 2020 年 4 月，并经过多年的努力，终于在 23.09 版本中发布了官方支持的 RISC-V 架构的操作系统。该版本的操作系统底座旨在为上层应用程序提供基础支持，具备高度可定制性、灵活性和安全性。它为 RISC-V 架构的计算平台提供稳定、可靠的操作环境，方便用户进行上层应用的安装和验证，共同推动 RISC-V 架构下软件生态的丰富和质量的提升。

功能描述

版本功能如下：

- 该操作系统底座的功能包括升级到 6.4.0 版本的内核，与主流架构保持一致。
- 提供稳定的基础系统底座，包括处理器管理、内存管理、任务调度、设备驱动等核心功能，以及常用的工具等。

应用场景

基于 QEMU 运行环境的 openEuler RISC-V 架构镜像主要适用于以下应用场景：

- 开发和测试：openEuler RISC-V 镜像为开发人员和测试人员提供了稳定可靠的操作系统基础，用于在 RISC-V 架构上构建和验证他们的应用程序。它允许他们在控制环境中测试软件的兼容性、性能和功能。
- 研究和教育：openEuler RISC-V 镜像的可用性为研究人员和教育工作者提供了一个平台，用于探索和研究 RISC-V 架构。它使他们能够进行实验、开发新的算法，并教授关于基于 RISC-V 的系统和软件开发的课程。
- 软件生态系统丰富：通过提供基础的操作系统基础设施，openEuler RISC-V 镜像为 RISC-V 架构的软件生态系统的增长和丰富做出贡献。它鼓励开发人员和组织针对 RISC-V 平台创建和优化软件解决方案。
- 系统定制：openEuler RISC-V 镜像具有高度可定制性，用户可以根据自己的特定需求定制操作系统。他们可以添加或删除组件，优化配置，并根据应用需求或特定用例自定义系统。
- 探索和创新：openEuler RISC-V 镜像通过提供稳定可靠的基础，鼓励个人和组织探索新思路、开发新应用，并为 RISC-V 生态系统的进步做出贡献。

总之，openEuler RISC-V 镜像作为 RISC-V 架构的基础操作系统底座，促进了应用程序开发、研究、教育、生态系统增长、系统定制以及推动 RISC-V 社区的探索和创新。后续 openEuler 也会将更多应用和特性合入到主线，通过测试验证后开放给用户。用户可以下载 preview 版本尝试更多功能。

动态完整性度量特性

DIM（Dynamic Integrity Measurement）动态完整性度量特性通过在程序运行时对内存中的关键数据（如代码段）进行度量，并将度量结果和基准值进行对比，确定内存数据是否被篡改，从而检测攻击行为，并采取应对措施。

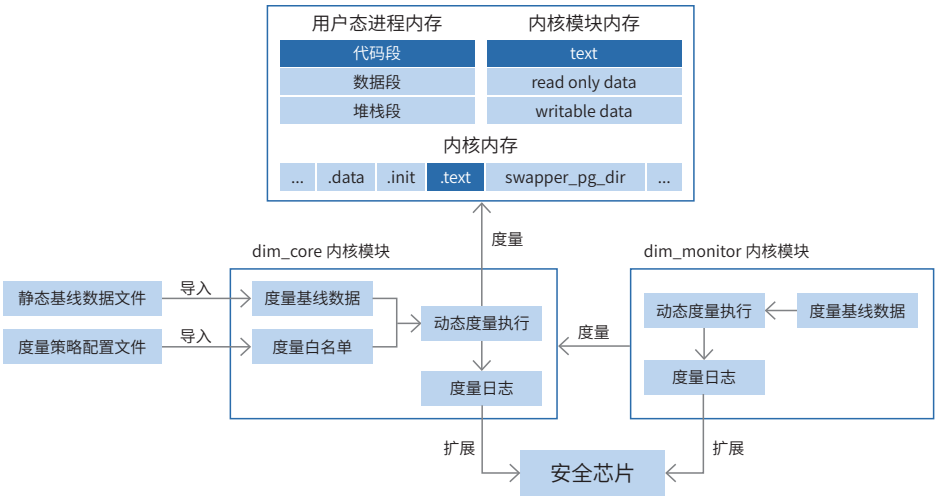
功能描述

DIM 动态完整性度量特性支持如下功能：

- 支持度量用户态进程、内核模块、内核内存代码段数据。
- 支持将度量结果扩展至 TPM 2.0 芯片 PCR 寄存器，用于对接远程证明。
- 支持配置度量策略，支持度量策略签名校验。
- 支持工具生成并导入度量基线数据，支持基线数据签名校验。
- 支持配置国密 SM3 度量算法。

DIM 动态完整性度量特性包含两个软件包 dim_tools 和 dim：

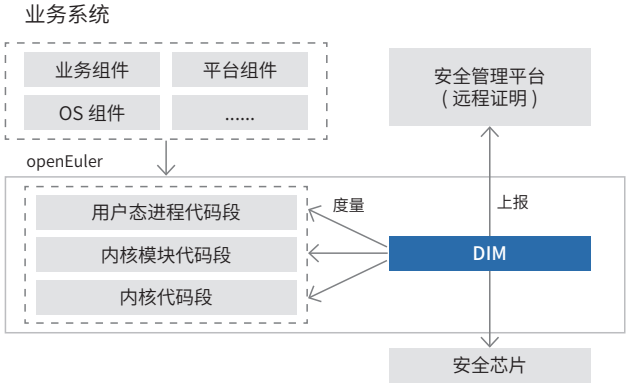
- dim_tools：提供 dim_gen_baseline 命令行工具，通过解析 ELF 二进制文件生成指定格式的代码段度量基线。
- dim：提供 dim_core 和 dim_monitor 内核模块。dim_core 为动态完整性度量核心模块，解析并导入用户配置的度量策略和度量基线，获取内存中的度量目标数据并执行度量功能；dim_monitor 对 dim_core 的代码段和关键数据执行度量保护，防止由于 dim_core 被篡改而导致度量功能失效。



应用场景

DIM 动态完整性度量特性可作为 OS 提供的基础安全机制，为信息系统各个组件提供内存数据的完整性保护。其典型使用场景如右图：

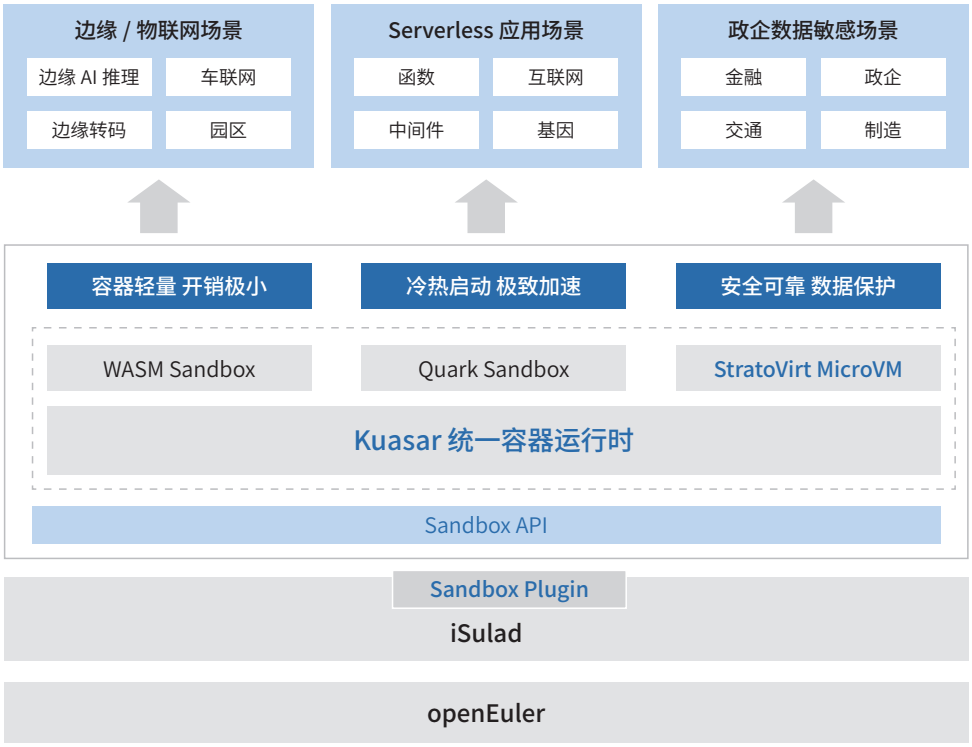
用户通过配置动态完整性度量策略，为系统中关键程序配置动态度量功能。DIM 完成目标数据度量后，将度量结果上报至安全管理平台，用户可以查询当前系统中的关键进程内存数据是否完整。在安全要求较高的场景下，用户还可对接可信计算远程证明机制，通过 TPM 证明度量结果的完整性。



Kuasar 统一容器运行时特性

Kuasar 是一款支持多种类型沙箱统一管理的容器运行时，可同时支持业界主流的多钟沙箱隔离技术，例如包括基于内核的原生容器沙箱、基于轻量级虚拟化技术的 microVM 沙箱、基于进程级虚拟化的 App Kernel 沙箱，以及新兴的 WebAssembly 沙箱。

openEuler 基于 Kuasar 统一容器运行时并结合已有 openEuler 生态中 iSulad 容器引擎和 StratoVirt 虚拟化引擎技术，打造面向云原生场景轻量级全栈自研的安全容器极低底噪、极速启动的关键竞争力。



功能描述

Kuasar 统一容器运行时通过 Sandbox API 接口，让 Sandbox 真正成为容器运行时接口中的“一等公民”，实现对 K8S CRI 接口标准中的 PodSandbox 概念的原生支持。Kuasar 支持用户态内核 Quark 沙箱、MicroVM 类型轻量级虚拟机沙箱和 WASM 类型语言运行时沙箱多种沙箱形态，满足不同云原生场景对沙箱的诉求。

本次发布的 Kuasar 0.1.0 版本，支持 StratoVirt 类型轻量级虚拟机沙箱，支持通过 K8S+iSulad 创建 StratoVirt 类型的安全容器实例。

支持功能特性：

- 支持 iSulad 容器引擎对接 Kuasar 容器运行时，兼容 K8S 云原生生态。
- 支持基于 StratoVirt 类型轻量级虚拟机沙箱技术创建安全容器沙箱。
- 支持 StratoVirt 类型安全容器进行资源精准限制管理。

约束限制：

- 当前仅支持 K8S CRI v1 版本接口中 Sandbox 和 Container 生命周期相关的接口，支持接口详细列表如下：

  rpc Version(VersionRequest) returns (VersionResponse) {}
  rpc RunPodSandbox(RunPodSandboxRequest) returns (RunPodSandboxResponse) {}
  rpc StopPodSandbox(StopPodSandboxRequest) returns (StopPodSandboxResponse) {}
  rpc RemovePodSandbox(RemovePodSandboxRequest) returns (RemovePodSandboxResponse) {}
  rpc PodSandboxStatus(PodSandboxStatusRequest) returns (PodSandboxStatusResponse) {}
  rpc ListPodSandbox(ListPodSandboxRequest) returns (ListPodSandboxResponse) {}
  rpc CreateContainer(CreateContainerRequest) returns (CreateContainerResponse) {}
  rpc StartContainer(StartContainerRequest) returns (StartContainerResponse) {}
  rpc StopContainer(StopContainerRequest) returns (StopContainerResponse) {}
  rpc RemoveContainer(RemoveContainerRequest) returns (RemoveContainerResponse) {}
  rpc ListContainers(ListContainersRequest) returns (ListContainersResponse) {}
  rpc ContainerStatus(ContainerStatusRequest) returns (ContainerStatusResponse) {}
  rpc Exec(ExecRequest) returns (ExecResponse) {}
  rpc Status(StatusRequest) returns (StatusResponse) {}

应用场景

单机多租户共享场景

- 以容器形态对外提供 Serverless 服务，需要容器规格小，单机密度高，启动速度快。
- 在一台物理机上会部署多个不同租户的容器，需要容器之间有强隔离的安全保证。
- 用户通常会应用在弹性扩缩容的场景，需要容器启动和销毁的速度快。

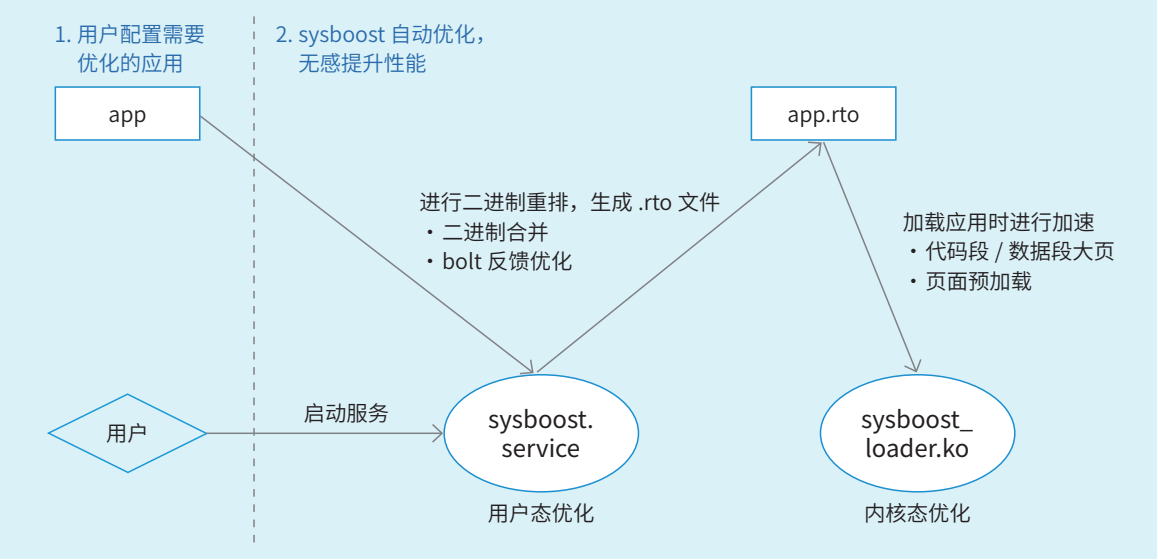
可信与不可信应用混合部署

在同一个物理机上会同时部署用户自行开发的可信应用和从外部第三方获取的不可信应用，需要保证第三方应用不会访问和攻击物理机上的其它应用，容器需要有强隔离安全保障。

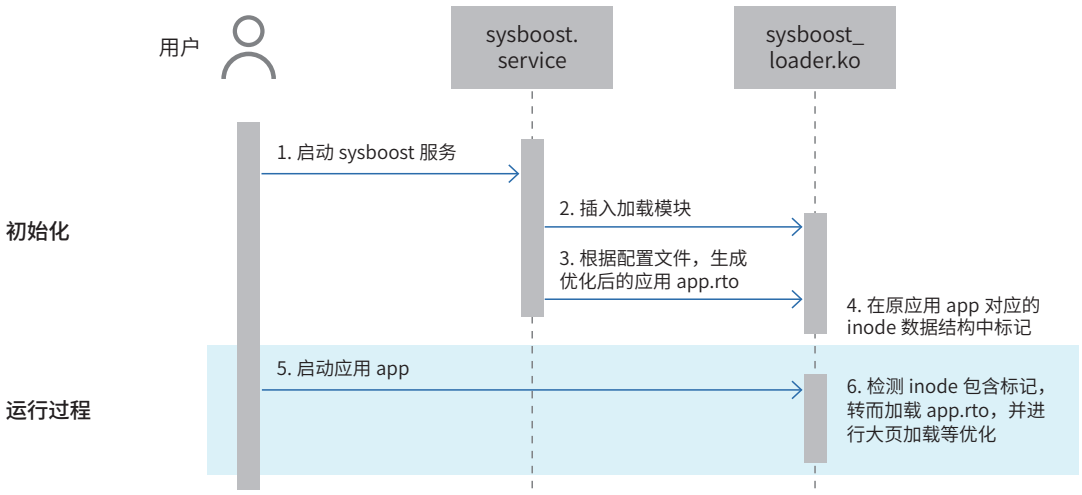
sysBoost 项目

sysBoost 是一个为应用进行系统微架构优化的工具，优化涉及汇编指令、代码布局、数据布局、内存大页、系统调用等方面。

功能描述



部署视图



流程设计图

二进制文件合并

目前只支持全静态合并场景，将应用与其依赖的动态库合并为一个二进制，并进行段级别的重排，将多个离散的代码段 / 数据段合并为一个，提升应用性能。

后续希望提供其他两种方式供用户选择：

- 支持运行时合并动态库场景。
- 支持部分合并场景 , 例如 : 内部闭源产品二进制不能合并 LGPL 开源库。

sysBoost 守护进程服务

为了达到性能开箱最优的目标，sysBoost 需要做到自动对系统中的二进制进行优化。sysBoost 使用注册 systemd 服务的方式达到这个效果，系统启动后，systemd 将会拉起 sysBoost 守护进程。

随后，sysBoost 守护进程读取配置文件获取需要优化的二进制以及对应的优化方式，按照用户的要求进行优化，并将优化好的二进制存储在“.rto”后缀的文件中。

目前 sysBoost 只支持优化 bash，且会默认开启优化；后续会开放更多应用。

rto 二进制加载内核模块

我们希望 sysBoost 优化的过程对用户无感知。考虑到应用升级、异常回退等场景，我们无法直接用优化后的“.rto”文件替换原文件，而是采用新增二进制加载模块的方法，在内核加载二进制时自动加载优化的二进制。

内核加载二进制时，会根据二进制的类型匹配一个合适的处理函数。目前 openEuler 包含的二进制都是 elf 格式，因此一定会使用 elf 加载函数。sysBoost 会向内核注册一个新的 elf 加载函数，该函数会检查待加载的二进制文件对应的 inode 中是否有 sysBoost 写入的特殊标记，如果有，则会加载优化过的“.rto”文件；没有则按原流程进行加载。

这个加载机制依靠 inode 中的特殊标记来识别应用是否被 sysBoost 优化，因此 sysBoost 优化流程在生成“.rto”文件时，都会在通过本内核模块将该标记置上。设置标记的方式是新增一个设备文件，用户态通过 ioctl 系统调用通知本内核模块进行设置。

二进制代码段 / 数据段大页预加载

用户态页表映射物理内存时，使用大页（2M）映射可以提升性能，而当前 openEuler 不支持文件页的大页映射。sysBoost 提供大页预加载的功能，在二进制优化完成后立即将其内容以大页形式加载到内核中，在应用启动时将预加载的内容批量映射到用户态页表，减少应用的缺页中断和访存延迟，提升启动速度和运行效率。

应用场景

sysBoost 为用户态程序提供系统微架构优化，优化效果较为通用，尤其适合频繁调用动态库 /itlb miss 较高的程序。

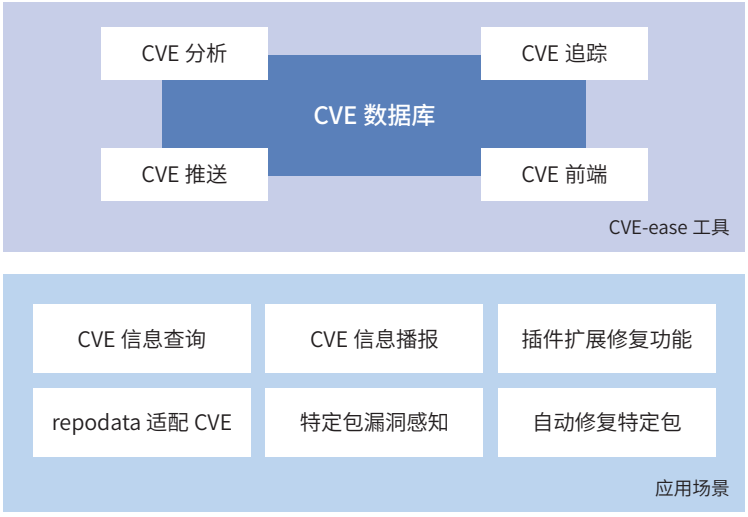
CVE-ease 项目

CVE-ease 是天翼云自主创新开发的一个专注于 CVE 信息的平台，它搜集了多个安全平台发布的各种 CVE 信息，并通过邮件、企业微信、钉钉等多种渠道及时通知用户。

功能描述

CVE-ease 平台旨在帮助用户快速了解和应对系统中存在的漏洞，在提高系统安全性和稳定性的同时，用户可以通过 CVE-ease 平台查看 CVE 信息的详细内容，包括漏洞描述、影响范围、修复建议等，并根据自己的系统情况选择合适的修复方案。目前 CVE-ease 具备以下能力：

- repodata 适配多厂家 OSV (Operating System Vendor)。
- motd 登陆播报功能。
- dnf 插件扩展修复功能。
- 自动修复特定包功能。
- 特定包感知功能。



技术挑战：面对计算机安全形势日益复杂严峻的背景下，如何从海量 CVE 信息实时整理出其中有效 CVE 信息及时告知用户，对 CVE 漏洞信息进行深度挖掘，对提升操作系统安全保障能力尤为重要。目前 CVE-ease 主要功能包括以下：

- CVE 信息动态获取和整合，实时跟踪多平台 CVE 披露信息，并进整合放入 CVE 数据库。
- CVE 信息提取和更新，对收集到的 CVE 信息提取关键信息并实时更新发生变更的 CVE。
- CVE 数据保存和管理，自动维护和管理 CVE 数据库。
- 历史 CVE 信息查看，通过交互方式查询各种条件的 CVE。
- CVE 信息实时播报，通过企业微信、钉钉、邮箱等方式实时播报历史 CVE 信息。

应用场景

用于操作系统安全漏洞感知、管理和修复。

EulerMaker 构建系统

EulerMaker 构建系统是一款软件包构建系统，完成源码到二进制软件包的构建，并支持开发者通过搭积木方式，组装和定制出适合自己需求的场景化 OS。主要提供增量 / 全量构建，分层定制与镜像定制的能力。

功能描述

- 增量 / 全量构建：基于软件包变化，结合软件包依赖关系，分析影响范围，得到待构建软件包列表，按照依赖顺序并行下发构建任务。
- 构建依赖查询：提供工程中软件包构建依赖表，支持筛选及统计软件包依赖及被依赖的软件包内容。
- 分层定制：支持在构建工程中，通过选择与配置层模型，实现对软件包的 patch，构建依赖，安装依赖，编译选项等内容的定制，完成针对软件包的场景化定制。
- 镜像定制：支持开发者通过配置 repo 源，生成 iso、qcow2、容器等 OS 镜像，并支持对镜像进行软件包列表定制。

应用场景

社区开发者及合作伙伴基于统一构建系统建设自己的用户个人仓，OS 核心仓，定制出适合自己需求的场景化 OS。

著作权说明 07

08 商标

openEuler 白皮书上使用和显示的所有商标、标志皆属 openEuler 社区所有，但注明属于其他方拥有的商标、标志、商号除外。未经 openEuler 社区或其他方书面许可，openEuler 白皮书所载的任何内容不应被视作以暗示、不反对或其他形式授予使用前述任何商标、标志的许可或权利。未经事先书面许可，任何人不得以任何方式使用 openEuler 社区的名称及 openEuler 社区的商标、标记。

附录 09

附录 1：搭建开发环境

环境准备	地址
下载安装 openEuler	https://openeuler.org/zh/download/
开发环境准备	https://gitee.com/openeuler/community/blob/master/zh/contributors/prepare-environment.md
构建软件包	https://gitee.com/openeuler/community/blob/master/zh/contributors/package-install.md

附录 2：安全处理流程和安全批露信息

社区安全问题披露	地址
安全处理流程	https://gitee.com/openeuler/security-committee/blob/master/security-process.md
安全披露信息	https://gitee.com/openeuler/security-committee/blob/master/security-disclosure.md

OpenEuler

商标声明

在本手册中以及本手册描述的产品中，出现的商标，产品名称，服务名称以及公司名称，由其各自的所有人拥有。

免责声明

本文档可能含有预测信息，包括但不限于有关未来的财务、运营、产品系列、新技术等信息。由于实践中存在很多不确定因素，可能导致实际结果与预测信息有很大的差别。因此，本文档信息仅供参考，不构成任何要约或承诺，不对您在本文档基础上做出的任何行为承担责任。可能不经通知修改上述信息，恕不另行通知。