<集群性能监控系统及其自动化>

软件架构文档

版本 <2.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| <日/月/年> | <x.x> | <详细信息> | <姓名> |
| <11/11/2024> | <1.0> | <文字总结> | <吴伯涛> |
| <12/12/2024> | <2.0> | <绘画视图> | <吴伯涛> |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 4

1.2 参考资料 4

2. 用例视图 4

3. 逻辑视图 4

4. 部署视图（可选） 4

5. 进程视图（可选） 4

6. 实现视图（可选） 4

7. 技术视图 4

8. 数据视图（可选） 5

9. 算法视图（可选） 5

10. 性能视图（可选） 5

11. 可靠性视图（可选） 5

12. 安全性视图（可选） 5

13. 易用性视图（可选） 5

14. 可维护性视图（可选） 5

软件架构文档

# 简介

## 目的

本文档将从构架方面对系统进行综合概述，其中会使用多种不同的构架视图来描述系统的各个方面。它用于记录并表述已对系统的构架方面作出的重要决策。

1. 指导系统开发：

软件构架文档会详细描述系统的整体架构，包括各个模块的功能、相互之间的关系以及数据流向。这为开发人员提供了清晰的开发方向，避免了开发过程中的混乱和重复劳动。文档会明确说明系统所采用的技术栈，例如编程语言、框架、数据库等。这有助于开发人员快速上手，并确保系统的一致性和可维护性。文档会定义开发过程中的规范和标准，例如代码风格、文档格式、测试方法等。这有助于提高开发效率，并确保代码质量。

2. 促进团队协作：

软件构架文档是开发团队、项目经理和利益相关者之间进行沟通的重要工具。它能够清晰地表达系统设计思路，帮助团队成员理解彼此的工作，并协调开发进度。文档记录了系统设计的经验和教训，为后续的开发和维护提供了宝贵的参考资料。

3. 保障系统质量：

通过软件构架文档，可以在开发之前对系统设计进行验证，确保设计的合理性和可行性。文档可以指导测试人员设计测试用例，确保系统功能的完整性和稳定性。

4. 降低开发成本：

明确的架构设计和规范的开发流程可以避免开发过程中的返工和修改，从而提高开发效率。良好的系统架构和文档可以方便后续的维护和扩展，降低维护成本。

## 参考资料

本项目使用的技术工具和框架的开源社区和论坛：

Prometheus 社区: https://prometheus.io/community/

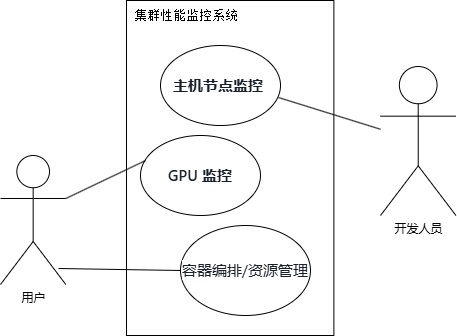
Grafana 社区: https://grafana.com/community/

NVIDIA 开发者论坛: https://forums.developer.nvidia.com

InfluxDB: 时序数据库的官方文档，例如 https://docs.influxdata.com/influxdb/latest/

Grafana: 监控数据可视化工具的官方文档，例如 https://grafana.com/docs/grafana/latest/

# 用例视图



主机节点监控：该功能涉及对整个集群的主机节点的监控，确保每个节点的正常运行和数据的一致性。

GPU监控：由于现代高性能计算常常依赖于GPU进行加速运算，因此对GPU的使用情况进行实时监控至关重要，以确保计算的效率和资源的合理利用。

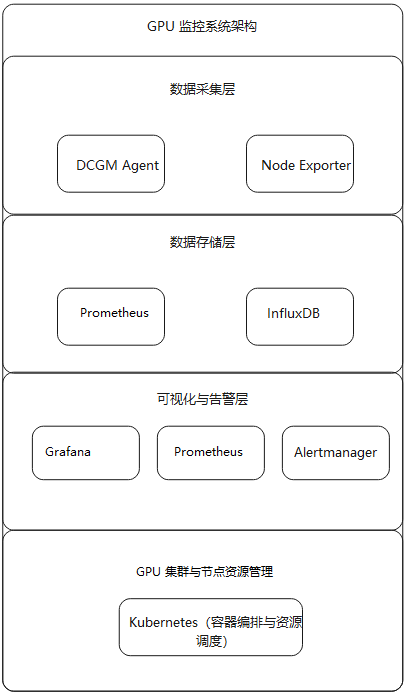
容器编排/资源管理：这一功能涉及到对容器的管理和资源的调度，使得系统能够高效地分配和管理各种计算资源，提高整体的运行效率。

用户交互与反馈：图中展示了用户与系统的交互，表明系统需要具备友好的用户界面和及时的反馈机制，以便用户能够轻松地操作和了解系统的状态。

开发人员的参与：开发人员通过监控系统来获取性能数据和异常信息，从而进行必要的调整和优化，确保系统的稳定性和可靠性。

# 逻辑视图

[本节说明系统的逻辑结构，例如系统被分解为多个子系统和包。而每个重要的包又被分解为多个类。您应该介绍那些在构架方面具有重要意义的包和类，并说明它们的职责，以及几项非常重要的关系、操作和属性。]



1. 数据采集层：主要使用DCGM（用于监控GPU）和Node Exporter（用于监控主机节点），获取GPU和主机资源的实时状态。
2. 数据存储层：将监控数据分别存储到Prometheus（短期监控数据）和InfluxDB（长时间序列数据存储）中，以供进一步处理和查询。
3. 可视化与告警层：使用Grafana进行监控数据的可视化，Alertmanager用于设定告警规则，提供故障预警。
4. GPU集群与节点资源管理：Kubernetes作为底层容器编排与资源管理工具，确保系统的高可用性和可扩展性。

# 部署视图（可选）

# 进程视图（可选）

# 实现视图（可选）

# 技术视图

编程语言：

Python: 主要用于编写数据采集、处理脚本以及监控服务的自动化配置。Python 生态丰富，拥有 Flask、Pandas 等库，能够高效完成系统开发。

Golang: 用于开发高效的数据采集和导出程序，例如为 DCGM 或 Node Exporter 创建数据导出器。Golang 性能优越，适合处理高并发数据采集任务。

开发工具：

Enterprise Architect: 用于绘制 UML 图，进行系统设计与建模。

Lucidchart 或 Draw.io: 简化的架构图和流程图设计工具，方便沟通和展示系统架构。

开发框架与工具：

Flask/Django (Python): 用于实现 Web API 和数据服务接口，将 GPU 监控数据提供给其他系统模块。

Prometheus Exporter: 自定义数据导出器，用于将 DCGM 采集的数据导入 Prometheus。

Kubernetes: 用于系统的容器化部署和资源调度，确保系统在大规模 GPU 集群中的扩展能力。

数据库：

Prometheus: 用于存储短期监控数据。

InfluxDB: 用于存储长时间序列数据，支持高效的历史数据查询。

中间件：

Grafana: 用于监控数据的可视化，允许用户自定义图表和仪表盘，进行复杂的实时数据分析和监控。

Alertmanager: 用于设定告警规则，提供故障预警。

# 数据视图（可选）

# 算法视图（可选）

# 性能视图（可选）

# 可靠性视图（可选）

# 安全性视图（可选）

# 易用性视图（可选）

# 可维护性视图（可选）