**云应用快速部署技术软件研制技术方案**

承研单位 西安电子科技大学

负 责 人 李龙海

联系电话 18992864681

时 间 2024 年 12 月

**目 录**

[1. 范围 1](#_Toc185265369)

[1.1 项目概述 1](#_Toc185265370)

[1.2 需求分析 3](#_Toc185265371)

[1.2.1 功能需求 3](#_Toc185265372)

[1.2.2 非功能需求 6](#_Toc185265373)

[2 总体设计 7](#_Toc185265374)

[2.1 运行环境 7](#_Toc185265375)

[2.1.1 软件环境 7](#_Toc185265376)

[2.1.2 硬件环境 8](#_Toc185265377)

[2.2 架构设计 8](#_Toc185265378)

[2.2.1 容器云应用管理系统总体架构 8](#_Toc185265379)

[2.2.2 虚拟机编排调度系统总体架构 12](#_Toc185265380)

[2.3 外部接口 14](#_Toc185265381)

[2.3.1 容器云应用管理系统外部接口 14](#_Toc185265382)

[2.3.2 虚拟机编排调度系统外部接口 17](#_Toc185265383)

[2.4 软件部件 20](#_Toc185265384)

[2.4.1 容器云应用管理系统软件部件 20](#_Toc185265385)

[2.4.2 虚拟机编排调度系统软件部件 23](#_Toc185265386)

[2.5 技术方案 30](#_Toc185265387)

[2.5.1 容器云应用管理系统技术方案 30](#_Toc185265388)

[2.5.2 虚拟机编排调度系统技术方案 45](#_Toc185265389)

[2.6 内部接口设计 65](#_Toc185265390)

[2.6.1 容器云应用管理系统内部接口设计 65](#_Toc185265391)

[2.6.2 虚拟机编排调度系统内部接口设计 66](#_Toc185265392)

[3 测试与验证策略 71](#_Toc185265393)

[3.1 单元测试策略 71](#_Toc185265394)

[3.2 集成测试策略 72](#_Toc185265395)

[3.3 系统测试策略 73](#_Toc185265396)

[3.3.1 功能测试方法 73](#_Toc185265397)

[3.3.2 性能测试方法 73](#_Toc185265398)

[4 项目验收与交付 74](#_Toc185265399)

# 范围

## 项目概述

**（1）项目背景**

在分布式计算环境中，采用容器化技术、容器编排调度技术进行分布式应用的开发、打包、部署和运维管理具有重要意义。容器化技术通过标准化的封装，确保应用及其依赖的环境一致性，提高了部署的便捷性和效率。编排调度技术（如K3S）通过自动化资源分配、负载均衡和容错处理，实现了分布式应用的高效管理和动态扩展。二者结合不仅简化了分布式系统的复杂性，还提高了系统的可靠性、可扩展性和资源利用率。将分布式应用打包为容器镜像等方式有助于快速迭代开发和灵活的业务部署，提升了分布式应用的易用性和适应性。因此，容器化技术和编排调度技术在嵌入式分布式计算场景中也得到了广泛应用。

而现有的基于Docker+K3S（K3S的嵌入式版本）的容器打包、部署及编排方法虽然实现了自动化和灵活性，但仍存在不足：

a. K3S配置复杂，YAML文件繁多且难以维护，尤其在多服务或依赖复杂的场景；

b. 缺乏版本控制和一致性管理，难以追踪和回滚配置变更；部署依赖关系处理较为繁琐，不利于复用和共享；

c. 不支持参数化和动态化配置，难以适应不同环境需求；

d. 嵌入式分布式技术环境中，很多应用程序也不适合于用Docker等轻量级容器方式进行打包，而更方便包装成QEMU虚拟机镜像等方式进行发布和现场部署。然后现有的K3S编排调度系统并不直接支持对QEMU等全虚拟化虚拟机的管理。

**（2）使用场景、功能简要介绍**

针对上述问题，本项目旨在针对嵌入式容器云平台研制一套更高阶的云应用快速部署管理系统以简化分布式云应用的部署运维工作，开发一套在嵌入式容器云平台支持对QEMU等全虚拟化虚拟机进行统一编排调度的软件系统。

本项目研制目标包含云应用部署管理系统和虚拟机编排调度系统两个软件系统，它们的应用场景和主要功能如图1所示。



图1 应用场景和功能示意图

其中，云应用部署管理系统主要用于天脉嵌入式容器云平台中的分布式应用部署管理，其主要用户为嵌入式软件工程开发人员和运维部署人员。软件开发人员利用云应用部署管理系统的客户端工具，可以将分布式应用程序软件工程中涉及的容器镜像文件、配置文件、依赖应用信息、分布式部署定义等文件打包为Chart包格式，然后上传到云应用打包文件仓库中，进行统一版本化管理。当需要将某个Chart包格式打包的分布式应用部署到具体的嵌入式容器云平台时，运维部署人员利用云应用部署管理系统的客户端工具将Chart包文件及相关镜像文件下载到云平台中，然后根据Chart包文件中部署相关的宏观指令通过操控更底层的K3S编排调度系统将分布式应用的不同容器安装部署到不同节点上并控制这些容器的运行。运维部署人员还可以方便的对已安装的分布式应用进行更新、回滚和追踪等版本化管理操作。

虚拟机编排调度系统的主要目标用户为运维管理人员，其主要功能包括：统一管理容器与虚拟机：通过对底层K3S平台扩展控制器、调度器模块，实现容器与虚拟机的统一资源调度、分配和监控，简化混合工作负载的管理流程；实现虚拟机生命周期管理：支持虚拟机的创建、启动、停止、迁移和删除操作，并可与K3S的原生功能无缝集成；高可用性与弹性扩展：支持虚拟机故障自动恢复和弹性扩展，保证服务的持续性和可靠性；资源监控与性能优化：提供虚拟机资源使用情况的实时监控和优化建议，提高资源利用效率和运维效果。

## 需求分析

云应用快速部署技术软件主要应用场景如下：

（1）用于嵌入式容器云环境的云应用管理

支持将基于容器的分布式云应用软件打包成Chart包形式进行发布，支持将复杂云应用软件的各类资源配置进行封装和模版化，支持基于Chart包仓库管理功能实现云应用软件的统一存储、版本控制、分发和共享。

简化云应用软件部署工作，使云应用软件部署人员、应用人员不用关心各类容器云资源配置细节、应用包之间的依赖关系，支持用户以简单的方式在不同环境、不同应用场景中通过简单配置安装、升级、回滚或卸载云应用软件。

（2）用于嵌入式容器云的虚拟机编排调度管理

对现有的嵌入式容器云编排调度系统进行扩展，使其支持对传统虚拟机的统一管理。通过扩展，使编排调度系统可以像管理普通容器一样管理虚拟机，做到统一的资源分配、调度规划和生命周期管理；使编排调度系统可以根据配置信息分配资源，在指定节点上创建、更新和删除虚拟机实例，可以监控虚拟机运行状态。

### 功能需求

如图2所示，云应用快速部署软件要实现的功能从整体上可以分解为两个主要部分：容器云应用管理系统和虚拟机编排调度管理系统。

容器云应用管理系统需要实现的主要功能包括：（1）Chart应用包管理相关功能（Chart包封装、上传、下载等）；（2）容器云应用安装管理相关功能（安装、卸载、升级、回滚等）；（3）容器云应用信息查看相关功能；（4）容器云应用包仓库管理相关功能。

虚拟机编排调度管理系统需要实现的主要功能包括：（1）虚拟机调度资源管理，主要针对现有嵌入式容器云编排调度软件（K3S）支持的调度资源类型进行扩展，将虚拟机资源纳入K3S的统一管理；（2）虚拟机统一控制引擎，主要用于实现一套统一的、与具体虚拟机类型无关的虚拟机配置、生命周期控制和运行监控接口，为编排调度软件能够接入多种不同类型虚拟机奠定基础；（3）虚拟机编排调度管理，主要实现虚拟机的节点分配、资源配置、生命周期管理（启动、停止、休眠、迁移）和运行状态监控等功能。



图2 功能分解

#### 容器云应用管理系统功能需求

（1）Chart应用包管理相关功能：

a. 提供Chart应用工程目录初始化功能；

b. 提供Chart应用工程目录打包功能；

c. 提供Chart包文件有效性检验功能；

d. 提供Chart包内容查看功能；

e. 提供Chart包可配置参数查看功能；

f. 提供Chart包依赖信息查看功能；

g. 提供Chart包文件下载功能；

h. 提供Chart包文件上传功能；

（2）容器云应用安装、卸载、升级、回滚相关功能：

a. 支持根据应用名称从Chart仓库下载并安装容器云应用；

b. 支持基于已下载Chart包文件安装容器云应用；

c. 支持卸载已安装容器云应用；

d. 提供将指定容器云应用升级至最新版本功能；

e. 提供将指定容器云应用升级至指定版本功能；

f. 提供将指定容器云应用回滚至指定版本功能；

（3）容器云应用信息查看相关功能

a. 支持查看当前集群上已安装的全部（部分）容器云应用信息；

b. 支持查看指定容器云应用的运行状态信息；

c. 支持查看指定容器云应用的版本变更信息；

d. 支持全局操作日志查询功能；

（4）Chart应用包仓库相关功能：

a. 支持在指定仓库中添加Chart应用包并更新索引功能；

b. 支持在指定仓库中删除Chart应用包并更新索引功能；

c. 提供添加Chart应用包仓库地址功能；

d. 提供删除Chart应用包仓库地址功能；

e. 提供Chart应用包仓库地址列表功能；

f. 提供针对指定仓库的容器云应用搜索功能；

g. 仓库中Chart包资源支持基于用户角色的访问控制；

#### 虚拟机编排调度系统功能需求

（1）针对嵌入式容器云编排调度软件的资源类型扩展：

a. 扩展支持Virtual Machine Instance (VMI)资源类型，用于定义无状态匿名虚拟机实例；

b. 扩展支持Virtual Machine (VM)资源类型，用于定义带持久状态命名虚拟机；

c. 扩展支持VirtualMachineInstanceReplicaSet资源类型，用于定义多副本虚拟机集群；

d. 虚拟机资源都支持CPU、内存、磁盘设备、镜像文件、存储卷、网络环境、安全证书参数的定义；

e. 支持查看集群中虚拟机资源信息；

（2）虚拟机资源的编排调度支持：

a. 支持根据编排调度软件命令启动虚拟机运行；

b. 支持根据编排调度软件命令停止虚拟机运行；

c. 支持根据编排调度软件命令暂停虚拟机运行；

d. 支持根据编排调度软件命令恢复虚拟机运行；

e. 支持根据编排调度软件命令在不同节点上迁移虚拟机；

f. 支持根据编排调度软件命令监控虚拟机运行信息；

（3）虚拟机统一控制引擎：

a. 提供统一的虚拟机生命周期管理接口；

b. 提供统一的虚拟机资源配置接口；

c. 提供统一的虚拟机运行状态监控接口；

### 非功能需求

#### 性能要求

a. Chart包仓库可支持不少于1000个Chart包的存储和管理；

b. Chart包仓库至少可支持50个客户端的并发访问；

c. 虚拟机编排调度管理软件可同时支持不少于20个虚拟机实例的管理；

#### 软件要求

a. 采用C/C++语言实现；

b. 提交的软件源代码需要符合GJB 8114—2013《C/C++语言编程安全子集》强制性准则；

c. 虚拟机资源的编排调度支持天脉Linux；

d. 云应用管理支持天脉操作系统，并支持与天脉编排调度软件的集成。

# 总体设计

## 运行环境

### 软件环境

容器云应用管理系统的软件环境如表1所示。

表1 容器云应用管理系统软件环境

|  |  |
| --- | --- |
| 集群环境 | 部署管理宿主机+天脉容器云平台+容器编排调度软件 |
| 目标机环境 | 天脉操作系统 |
| 宿主机环境 | Windows XP或Windows7操作系统 |
| 开发语言 | C++、JavaScript、Lua |
| 开发工具 | 1. MinGW+GCC+CMake+VSCode  2. AcoreIDE集成开发环境 |
| 调试工具 | 1. GCC调试工具  2. AcoreIDE集成开发环境，串口通信工具 |

虚拟机编排调度系统的软件环境如表2所示。

表2 虚拟机编排调度系统软件环境

|  |  |
| --- | --- |
| 集群环境 | 天脉Linux版天脉容器云平台+容器编排调度系统 |
| 目标机环境 | 天脉Linux系统，QEMU虚拟机环境 |
| 宿主机环境 | Windows或Linux |
| 开发语言 | C++ |
| 开发工具 | GCC、CMake |
| 调试工具 | GDB |

### 硬件环境

（1）容器云应用管理系统硬件环境

a. 目标机环境

嵌入式开发板，处理器/内存/固态盘资源能够满足天脉容器系统运行要求；

b. 集群环境

1. 容器云控制节点1个，运行天脉编排调度软件；
2. 容器云从节点 ≥ 2个，运行天脉容器系统；
3. 云应用包仓库服务器，运行Windows或Linux；
4. 集群中所有节点通过以太网互联；

（2）虚拟机编排调度系统硬件环境

a. 目标机环境

嵌入式开发板，处理器/内存/固态盘资源能够满足天脉Linux系统运行要求；

b. 集群环境

1. 容器云控制节点1个，运行天脉Linux+天脉编排调度软件；
2. 容器云从节点 ≥ 2个，运行天脉Linux+QEMU虚拟环境；
3. 集群中所有节点通过以太网互联

## 架构设计

### 容器云应用管理系统总体架构

容器云应用管理系统的总体架构设计如图3所示。从上到下包含了用户交互命令工具模块、Web UI管理工具、Chart应用包管理模块、云应用部署管理模块、应用包仓库管理模块、资源定义生成校验模块、容器云平台交互接口模块以及日志管理模块。其中每个模块还包含对应的子模块。

容器云应用管理系统的目标用户为分布式软件开发人员和分布式软件运维部署人员。用户交互命令工具位于整个体系结构的最上层，为这两类用户提供了使用容器云应用管理系统相关功能的命令方式人机交互接口。软件开发人员可以利用该工具将自己开发的容器云应用程序包装为Chart包形态进行测试和发布。运维管理人员可以利用该工具将已经发布的Chart应用包安装部署到具体的嵌入式容器云平台中。Web UI管理工具是为用户提供基于Web的图像用户管理界面，简化用户对云应用的部署、信息查询和监控操作。本方案将仿照Helm Dashboard采用前后端分离的B/S架构，包含Web前端模块和Web后端模块，前后端之间用Restful API方式交互。Web前端基于React框架实现，，后端采用C++实现的HTTP Server，为前端提供Restful API服务。后端具体实现Chart云应用管理操作时调用上图3中“Library API层”的内部接口函数实现。



**图3 容器云应用管理系统总体架构设计**

Chart应用包管理模块负责Chart包的脚手架内容生成、内容查看、依赖管理以及包的创建、初始化、打包、上传和下载。Chart包支持包含的文件和目录结构与开源Helm系统保持兼容。云应用管理模块用于Chart包格式容器云应用的安装、升级、回滚、卸载及信息查看。资源定义生成校验模块利用内部的模板通用解析渲染模块实现Chart包模板的解析与容器编排调度资源定义文件的渲染生成，还可以实现Chart包格式的合法性校验。仓库管理模块支持Chart包仓库的管理、增删查、仓库索引维护和基于用户角色的访问控制。

关于Chart仓库中的资源访问控制模块，将基于Nginx软件采用Lua脚本语言对Nginx进行扩展的方法实现。要实现访问控制首先要对用户进行认证，这个直接利用Nginx支持的基于用户名密码的认证方式实现。用户认证后，通过查本地Sqllite库获取其角色，然后利用Lua脚本实现关于Chart应用包资源的基于角色的访问控制（RBAC）功能。在前端管理工具中增加相应的用户角色和不同角色能访问哪类资源的设置功能。

容器云接口交互模块封装了与底层嵌入式容器云平台的程序交互接口。它主要与天脉容器编排调度系统（K3S）的API Sever进行交互，通过调用API Sever的Restful API接口实现底层集群状态信息的访问，实现编排调度资源的定义、生成、属性设定和状态查询。容器云应用管理系统将与所在集群相关的云应用部署状态数据（如应用发布历史、当前版本、应用发布状态等）基于K3S提供的ConfigMap存储接口存储在天脉容器编排调度系统K3S的Sqlite存储数据库中。因此，容器云接口交互模块还需要访问天脉容器编排调度系统的储层资源接口，实现云应用创建、更新、删除和发布记录的存储。

日志管理模块用于实现用户操作日志的记录和查询功能。日志中记录了操作用户，操作类型，操作结果和操作时间等信息。管理员可以使用命令行工具和网页前端对全局日志进行查询，便于实现云应用管理操作的追踪和审计。日志管理模块将支持日志写入、日志查询和日志归档功能。日志写入功能就是为系统中其它模块提供简单的日志写入API接口，方便其它模块将用户操作信息在恰当的时机写入日志中。日志查询功能也是提供日志查询API，可以分别以时间、用户、操作类型为过滤条件对日志信息进行查询。日志存放在SQLite中。日志归档功能是指支持将某个时间点之前的日志从SQLite中导出到文件中，然后再将归档过的日志从SQLite中删除，这样可以提高SQLite的查询速度。

从软件发行形态上讲，上层的Web UI管理工具、用户交互命令工具作为可以独立运行的可执行程序，而下层的Chart包管理模块、云应用管理模块、仓库管理模块、资源定义生成校验模块等被封装成C++库的形式，并且对上层暴露API接口。用户交互命令工具或Web UI管理工具后端模块在实现具体功能时，实际上时调用这些API接口实现的。当用户命令输入时，用户交互命令工具在讲该命令解析，然后调用底层的API接口分发给下层的4个模块执行具体的操作。

用户交互模块与云应用管理模块进行交互，云应用管理会将应用部署到K3S集群，这就需要将Chart包格式的应用信息解析为K3S可识别并部署的应用配置信息，这个步骤由模版渲染模块完成；当Chart格式信息被解析完成后，需要发送解析结果至集群，这个步骤需要云应用管理模块调用接口交互模块提供的接口完成。本系统发送给底层K3S系统的是根据模板生成的完整的YAML文件，K3S将根据这些YAML文件中不同类型资源定义，自动到容器镜像仓库（如Docker Hub）中下载所需的容器镜像。

通过这些模块的协作，系统实现了对Chart包、云应用和K3S资源的全面管理与操作，提供了高效的云应用部署与管理解决方案。

### 虚拟机编排调度系统总体架构

虚拟机编排调度系统的总体架构设计如图4所示，主要包括了六个模块：管理工具扩展模块、VM资源类型扩展模块、控制器扩展模块、处理器模块、启动器模块和虚拟机统一控制引擎。这些模块在系统中分布于不同的节点，分别承担不同的职责。其中VM资源类型扩展模块、控制器扩展模块工作在K3S的Master节点中，处理器模块、启动器模块和虚拟机统一控制引擎工作在每一个Node节点中，管理工具扩展模块可以灵活地安装在部署运维管理员所在的控制台计算机上。

VM资源类型扩展模块扩展了天脉容器编排调度系统（K3S）的API Server，使其支持新的自定义资源类型（CRDs），使得K3S除了原生的容器和Pod资源之外，还可以管理虚拟机VM及其相关资源。这些新定义资源包括 VirtualMachine（VM 类型）、VirtualMachineInstance（VMI 类型）、VirtualMachineInstance ReplicaSet等，分别对应虚拟机、虚拟机实例及其虚拟机多副本集的管理。VM资源类型扩展模块是以CRD形式去管理虚拟机资源，VM资源类型扩展模块就是所有虚拟化操作的入口。



图4 虚拟机编排调度系统总体架构设计

控制器扩展模块是负责管理虚拟机实例（VMI）生命周期的核心组件。它的主要功能是根据虚拟机资源的定义，自动创建、更新、删除和监控虚拟机实例（VMI）。当用户创建或修改VM资源时，控制器扩展模块会根据虚拟机的定义生成或更新相应的VMI实例，并确保它们的状态与用户期望一致。控制器扩展模块还负责处理虚拟机实例的启动、停止、重启等操作，并确保虚拟机实例的副本数在集群中始终保持一致。它与调度器协同工作，确保虚拟机实例在合适的节点上运行，并自动处理故障恢复。此外，控制器扩展模块还确保虚拟机实例的网络和存储资源正确配置和管理。

处理器模块会以Deamonset形式部署在每一个节点上，负责监控节点上的每个虚拟机实例状态变化，一旦检测到状态的变化，会进行响应并且确保相应的操作能够达到所需（理想）的状态。处理器模块负责虚拟机实例（VMI）的实际管理和运行。它是虚拟机生命周期管理的执行者，接收来自控制器扩展模块的指令，控制虚拟机实例的启动、停止、迁移等操作。

启动器模块运行在每个包含VMI的Pod中。K3S调度系统运行在每个节点上的Kubelet进程只负责包含VMI的Pod运行状态，不会去关心VMI创建情况。处理器模块会根据CRD参数配置去通知启动器模块去使用本地的虚拟机统一控制引擎来启动VMI，随着Pod的生命周期结束，启动器模块也会去通知VMI去执行终止操作；启动器模块通过虚拟机统一控制引擎去管理VM的生命周期。

虚拟机统一控制引擎提供了统一的API来管理不同类型的虚拟机、存储、网络和其他虚拟化资源，常用的虚拟机类型包括QEMU、KVM、Xen等。虚拟机统一控制引擎使得其它模块能够以一致的方式实现虚拟机生命周期管理，比如可以创建、启动、停止、暂停和删除虚拟机；以统一方式管理虚拟机的CPU、内存、存储、网络等资源配置；以统一的方式实现网络管理，支持虚拟网络的创建和配置，实现虚拟机之间的网络连接；以统一方式实现虚拟机存储资源管理，管理虚拟机磁盘和存储卷，支持快照、克隆和存储池管理。

管理工具扩展模块是对K3S的Kubectl功能的扩展，专注于虚拟化资源的管理。管理工具扩展模块提供了一个简便的命令行界面，用户可以通过它快速管理虚拟机实例，执行虚拟化相关操作，而无需直接与K3S的的低层资源进行交互。该工具可以用来创建、启动、停止、重启、删除虚拟机实例，提供用于调整虚拟机状态的命令，如暂停、恢复虚拟机，允许用户通过 virtctl console 访问虚拟机实例的控制台，进行实时交互，支持虚拟机实例的迁移操作，使虚拟机能够在集群内的不同节点之间移动。

## 外部接口

### 容器云应用管理系统外部接口

容器云应用管理系统与其它软件系统产生的外部接口关系如图5 容器云应用管理系统外部接口所示。

（1）与用户之间的人机交互接口

容器云应用管理系统提供的命令行工具和Web UI管理工具实现了多种人机交互接口，用于管理Chart包格式的容器云应用程序的生命周期。主要功能包括：

a. Chart应用包管理相关命令：创建新的Chart包模板，帮助用户快速生成 Chart 目录结构和基本配置；将一个Chart工程目录打包为.tgz文件，便于存储和分发；从Chart包仓库中下载指定的Chart应用；帮助开发人员验证Chart包的正确性。

b. 云应用安装、升级、回滚、卸载相关命令：Chart应用包中资源描述文件定义的云应用（一般包含多个容器镜像文件）被部署到嵌入式容器云集群，生成并应用K3S编排调度资源；还提供升级已安装的应用程序，更新Chart包或配置的命令；支持将应用程序恢复到之前的版本；支持删除已安装的云应用，清除相关资源。

c. 已安装云应用查询和状态管理：列出所有已安装的容器云应用，显示发布的状态和版本信息；显示某个已发布应用的详细信息，帮助用户查看当前状态和配置；查看某个分布式应用发布的历史版本，便于跟踪版本变更。

d. 管理应用包仓库，如添加、删除、更新仓库等。

这些命令为开发人员和运维人员提供了便捷的接口来管理容器云应用程序的部署、更新、查询和清理等操作。



图5 容器云应用管理系统外部接口

（2）与底层容器云平台（天脉容器编排调度系统）之间的Restful API接口：

a. 集群信息、状态查询相关接口：通过访问K3S的API Server的相关API端点，容器云应用管理系统可以查询底层容器云集群的状态信息，包括集群版本、节点状态、Pod资源使用情况、事件日志、命名空间、服务、部署和副本集的状态等。此外，还可以访问集群的监控指标，如CPU和内存使用情况，帮助运维人员有效地监控和管理集群。

b. 调度资源管控接口：通过调用K3S的API Server的Core API实现对Pod、Deployment、Service、ConfigMap、Secret 等资源对象的创建、更新和删除。通过Namespace API来确保资源被正确地隔离和管理。集群中创建一个云应用Release，每个Release都会生成一个或多个K3S资源对象，云应用管理系统通过 Release名称和命名空间来区分不同的云应用程序实例。云应用管理系统通过与K3S API的交互，实时更新应用的状态，例如安装、升级或回滚应用时，云应用管理系统会通过K3S的API监控资源的创建和变更，确保应用状态符合预期。

c. 状态存储相关接口：

云应用管理系统使用K3S的ConfigMap和Secret来存储容器云应用Releas信息，记录每个云应用Release的历史和当前状态。通过API存储云应用部署版本、安装历史和其他元数据。

（3）与Chart应用包仓库之间的接口

Chart应用包仓库用HTTP Server软件（Nginx）实现，因此容器云应用管理系统和Chart应用包仓库之间是标准的HTTP协议。利用HTTP GET请求获取仓库的索引文件，通过GET请求获取这个索引文件来更新本地的Chart仓库信息；利用HTTP GET请求下载tgz格式的Chart应用包；利用HTTP POST请求将新的Chart包上传到Chart仓库；利用HTTP PUT请求，更新Chart的索引文件或上传新的版本，通常由仓库管理员操作。这些 HTTP 请求使得云应用管理客户端工具与远程Chart包仓库服务器交互，实现Chart应用包文件的添加、更新、下载等操作，简化容器云应用的管理和部署。

### 虚拟机编排调度系统外部接口

虚拟机编排调度系统与外部软件系统或底层软件系统的外部接口关系如图6所示。

（1）与用户之间的人机交互接口

虚拟机编排调度系统提供的命令行工具virtctl，专门用于管理虚拟机实例（VMI）和虚拟机（VM）。它为用户提供了简化的控制命令接口，主要包括以下交互命令：

a. 虚拟机实例管理：启动虚拟机实例、停止虚拟机实例、暂停虚拟机实例的运行、恢复已暂停的虚拟机实例。

b. 虚拟机实例控制：连接到虚拟机实例的控制台，提供交互式终端。

c. 虚拟机操作：创建一个虚拟机实例、删除指定的虚拟机实例。

d. 虚拟机状态查看：列出当前集群中所有虚拟机实例及其状态。

virtctl 通过提供一套专用命令行接口，使得用户可以方便地操作容器云平台上的虚拟机实例，简化了虚拟机的生命周期管理。



图6 虚拟机编排调度系统外部接口

（2）与容器编排调度系统（K3S）之间的接口

虚拟机编排调度系统依赖于K3S提供的RESTful API接口来管理虚拟机和虚拟机实例的生命周期。主要涉及到以下几类接口：

a. 资源管理接口：

* /apis/core/v1/namespaces，用于管理虚拟机实例和虚拟机的命名空间，确保虚拟机资源隔离；
* /api/v1/pods：创建和管理与虚拟机实例相关的Pods，虚拟机的生命周期由Pod管理；

b. 控制器和调度接口：

* /apis/apps/v1/deployments、/apis/apps/v1/replicasets：管理虚拟机的部署和副本集，确保虚拟机实例的扩展和高可用性；
* /apis/scheduling/v1/podtopologiespreadconstraints：用于虚拟机调度时的资源分配和负载均衡。

c. 存储接口：

CSI (Container Storage Interface)：使用 K3S的CSI 插件管理虚拟机的持久化存储卷（如Ceph、NFS）。

d. 网络接口：

CNI (Container Network Interface)：为虚拟机实例提供网络连接，确保虚拟机与集群其他资源的通信。

e. 监控与指标接口：

Metrics API：收集虚拟机实例和宿主节点的资源使用情况。

通过这些接口，虚拟机编排调度系统将虚拟机管理和K3S集群的资源管理功能无缝集成，提供虚拟机的生命周期管理、调度、存储、网络等功能。

f. 事件监听接口：

控制器模块通过以下方式与K3S交互：

* 控制器使用Informer来监听VMI资源的变化。Informer是K3S中的一个机制，它提供了对特定资源的实时监控。当VMI的状态或其他字段发生变化时，Informer 会将这些变更通知给控制器。
* 控制器通过K3S API Server使用Watch请求来监视特定资源（如VMI）。当VMI资源发生任何变更（如状态改变），API Server 会将变更通过 WebSocket或HTTP事件流的方式推送给控制器模块。

（3）与虚拟机QEMU之间的接口

虚拟机编排调度系统的虚拟机统一控制引擎封装了统一的API来管理不同类型的虚拟机、存储、网络和其他虚拟化资源。虚拟机统一控制引擎采用统一接口抽象层+不同类型驱动的方式，实现对常用的虚拟机类型包括QEMU、KVM、Xen等的控制管理。驱动程序通常需要根据下层具体虚拟机类型提供的API或其它接口协议进行编写。具体对QEMU而言，虚拟机统一控制引擎利用如下接口实现对QEMU虚拟机的控制：

a. 控制命令接口：通过QEMU提供的命令行工具实现QEMU虚拟机的创建、启动、暂停、停止等生命周期操作。通过XML配置文件与QEMU进行通信，告诉QEMU虚拟机的硬件资源要求。通过 XML 格式的配置文件（如 CPU 核数、内存大小、磁盘映像等）来配置 QEMU 虚拟机。

b. 基于QMP协议的接口：QMP（QEMU Machine Protocol）协议是一种基于 JSON-RPC 的通信协议，允许外部工具或管理系统与 QEMU 虚拟机进行交互。QMP 提供了一种标准化的方式，用于控制和管理 QEMU 虚拟机的运行状态、配置、监控等操作。

## 软件部件

### 容器云应用管理系统软件部件

（1）用户交互命令工具模块负责接收并识别用户输入的命令种类，然后将命令分发到对应的功能模块来执行。

（2）Chart应用包管理模块提供了对Chart包的全面管理功能。其内容管理子模块支持查看Chart包的详细结构、依赖关系以及可配置参数，帮助用户了解Chart包的功能。包操作子模块则涵盖Chart包的创建、初始化、有效性检查、打包以及从远程仓库下载Chart包的操作，确保Chart包的完整性和可用性。

（3）云应用管理模块包括云应用版本管理和云应用信息查看两个子模块。版本管理子模块支持云应用的安装、升级、回滚和卸载，保证了云应用在版本间的平滑迁移与操作。信息查看子模块则允许用户查看云应用的运行状态、版本变更历史以及集群中已部署的应用信息，便于实时监控和管理云应用。

（4）资源定义生成校验模块包含模板渲染模块、模版解析和资源生成桑个子模块。模版解析子模块负责解析Chart包中的模板文件，提取并分析文件中的动态内容，为后续的渲染提供基础数据。资源生成子模块则将解析后的模板内容与values.yaml文件中的配置数据结合，生成符合K3S要求的资源对象信息，如Pod、Service等，完成资源的自动化生成。

（5）仓库管理模块由仓库信息管理和仓库Chart管理子模块组成。信息管理子模块支持添加、删除以及查看Chart仓库，便于用户管理多个Chart仓库源。Chart管理子模块提供了在指定仓库中对Chart包的增删改查操作，同时支持更新仓库的索引文件。

（6）容器云平台交互接口模块包括K3S资源操作接口、K3S集群信息访问接口、K3S存储层接口三个子模块。资源操作接口模块提供对K3S资源的创建、更新和删除等操作，支持与K3S集群的直接交互。存储层接口模块则负责管理K3S资源的发布记录，包括创建、更新和删除发布记录，确保资源发布历史可追踪和管理。

容器云应用管理系统每个软件单元说明如表3所示。

表3 容器云应用管理系统软件模块说明

|  |  |  |
| --- | --- | --- |
| **软件单元** | | **说明** |
| 用户交互命令工具 | 命令解析模块 | 对用户的输入命令进行解析匹配 |
| 命令分发模块 | 将命令分发到对应模块执行 |
| 配置管理模块 | 从配置文件或环境变量中读取和解析工作参数 |
| Chart应用包管理模块 | Chart包内容管理子模块 | 提供查看Chart包内容，依赖信息，查看可配置参数接口； |
| Chart包操作子模块 | 提供Chart包创建，初始化，有效性检验，打包，从仓库下载Chart包等功能； |
| 云应用管理模块 | 云应用版本管理子模块 | 提供云应用安装，升级，回滚，卸载等功能； |
| 云应用信息查看子模块 | 提供查看指定云应用运行状态信息，版本变更信息，集群上已部署的云应用信息等功能； |
| 资源定义生成校验模块 | 模版通用渲染解析子模块 | 负责将Chart包下的模版文件进行内容解析； |
| 资源校验子模块 | 实现Chart包格式的合法性校验 |
| 类资源生成子模块 | 负责将解析完成的模版文件内容和values,yaml文件一起渲染，生成K3S资源对象信息； |
| 仓库管理模块 | 仓库信息管理子模块 | 提供添加Chart包仓库，删除仓库，查看仓库列表等功能； |
| 仓库Chart管理子模块 | 提供在特定Chart仓库中添加和删除，搜索Chart包功能，以及维护仓库索引文件功能； |
| Chart包仓库 | 资源访问控制模块 | 提供用户身份认证、用户角色绑定和基于用户角色的Chart包资源访问控制功能； |
| 容器云平台接口模块 | K3S资源操作封装接口模块 | 提供操作K3S资源的内部API接口，如创建，更新，删除资源；K3S的API Server提供的是HTTP Restful形式的接口，此模块将Restful API封装成内部函数式API的形式。 |
| 存储服务封装接口模块 | 提供云应用发布记录信息存储时需要用到的存储服务相关接口，包括创建，更新，删除发布记录等功能；通过统一的存储服务接口，云应用发布记录可以存储到K3S的ConfigMap资源中（间接被存储到K3S的SQLite中），也可以存储到独立的关系数据库中。 |
| 集群信息访问接口 | 提供查询K3S集群信息的接口，如包含的Node节点列表，不同Node资源信息、容器运行信息等接口。 |

各个软件单元调用关系参考如图7所示。

**图7 软件单元调用关系图**

用户输入命令后，首先会对命令解析，然后命令分发模块会根据命令类型将命令发送到对应模块执行；

当向仓库中添加Chart包时，首先会在Chart包操作模块创建并打包Chart包，进行有效性验证等，然后才会与仓库Chart管理模块交互，添加到指定Chart仓库；

当执行云应用版本管理相关操作时，如果是安装应用，需要与Chart包操作模块交互，进行Chart包安装操作，然后和模版渲染模块交互，将Chart包信息解析成K3S资源对象，再将资源对象发送给集群，进行创建资源和创建发布记录的操作，这个过程还要和通过与接口交互模块完成；

如果用户需要查看云应用相关运行信息时，云应用信息查看模块也要调用接口交互模块，通过K3S存储层接口查看应用发布记录，资源操作接口查看当前云应用在集群上的运行状态等。

### 虚拟机编排调度系统软件部件

（1）VM资源类型扩展模块

VM资源类型扩展模块是虚拟机编排调度系统的关键组件之一，它通过对K3S API Server进行扩展，允许用户通过标准的K3S API管理虚拟机及虚拟机实例。它使虚拟机编排调度系统能够无缝集成到K3S集群中，提供虚拟机资源的创建、调度、监控等功能，并与K3S生态系统的其他部分协作，支持虚拟机的生命周期管理。该模块提供的功能主要包括以下几方面：

a. 虚拟机资源的定义和管理

虚拟机（VM）资源：VM资源扩展模块 通过定义和管理VirtualMachine资源，使得用户可以通过K3S API创建、删除、更新和查询虚拟机实例。

虚拟机实例（VMI）资源：它还支持VirtualMachineInstance（VMI）资源，允许用户管理虚拟机实例的生命周期（如启动、暂停、删除等）。

VM资源扩展模块充当虚拟机资源的入口，处理用户请求并将其转发给后端控制器（控制器扩展模块）。

b. 与K3S API Server的集成

VM资源扩展模块为虚拟机编排调度系统提供了自定义资源定义（CRD），例如VirtualMachine和VirtualMachineInstance，以及相应的操作和状态管理。这些CRD通过K3S API Server向用户暴露，并且与K3S集群中的其他资源（如Pods、Secrets）紧密集成。它将虚拟机管理的生命周期和配置管理与K3S的资源管理模型无缝对接，支持K3S风格的控制和扩展。

c. 虚拟机管理和调度功能

VM资源扩展模块 提供了用于管理虚拟机的API接口，例如启动、停止、暂停、迁移等操作。用户通过VM资源扩展模块提交请求，控制器模块处理这些请求并与底层虚拟化技术（QEMU、KVM等）交互。

d. 与其他虚拟机编排调度系统 模块的交互

VM资源扩展模块是虚拟机编排调度系统中与K3S API交互的入口。它与控制器controller、处理器handler等模块协同工作，实现虚拟机的调度、管理和监控。它通过接收用户的API请求并将其传递到后端的控制器或虚拟化层，支持虚拟机实例的创建、删除、更新等操作。

e. 健康检查与身份验证

VM资源扩展模块 提供健康检查接口，允许K3S集群监控虚拟机编排调度系统的API服务是否正常运行。

它支持K3S的身份验证和授权机制，确保只有经过授权的用户或服务才能访问虚拟机资源。

（2）控制器扩展模块

控制器 模块是负责虚拟机资源管理和调度的核心组件之一。它与 K3S 的控制循环（controller loop）密切集成，监控和管理虚拟机实例（VMI）和虚拟机（VM）的生命周期。控制器 模块实现了以下主要功能：

a. 虚拟机生命周期管理

创建与删除虚拟机实例（VMI）：控制器 负责创建和删除虚拟机实例。它监控 VirtualMachineInstance（VMI）资源，并确保虚拟机的状态与预期保持一致。当一个新的 VMI 被创建时，控制器 会与底层虚拟化技术（如QEMU/KVM）交互，启动虚拟机实例。当VMI被删除时，控制器会确保虚拟机实例及其相关资源被正确清理。

虚拟机状态管理：控制器 监控虚拟机实例的运行状态，并根据状态变化执行必要的操作。例如，启动、暂停、恢复或迁移虚拟机实例等。

b. 虚拟机配置管理

更新虚拟机配置：当虚拟机的配置（如CPU、内存、网络等）发生变化时，控制器会确保这些配置被应用到实际的虚拟机实例中。这些配置更新通常通过K3S的自定义资源定义（CRD）进行管理。

动态资源调整：控制器还负责处理虚拟机的动态资源调整请求，如CPU或内存的变更。这些更改通常涉及重新调度或修改虚拟机实例的运行时配置。

c. 快照与存储管理

虚拟机快照：控制器支持虚拟机的快照功能，允许用户对虚拟机实例的状态进行保存，并在需要时恢复。它通过与存储后端（如Ceph、NFS）集成来管理虚拟机的快照。

存储卷管理：控制器负责虚拟机实例的存储卷管理，确保虚拟机实例的磁盘映像、数据卷等被正确挂载和卸载。

d. 事件和状态同步

事件通知：控制器通过K3S的事件系统，监控虚拟机实例的状态变化，并将重要的事件（如启动、停止、迁移等）记录并报告。

状态同步：控制器 不断同步虚拟机实例的实际状态与期望状态。当虚拟机实例的实际状态与期望状态不一致时，控制器 会采取适当的措施来修复状态不一致。

e. 与其他虚拟机编排调度模块的集成

与API扩展模块集成：控制器和API扩展模块紧密集成，API扩展模块提供了与K3S API Server的接口，而控制器则实现了虚拟机的实际操作逻辑。API扩展模块处理来自用户的请求，将其传递给控制器进行处理。

与处理器模块协作：控制器与处理器模块紧密合作，后者负责虚拟机实例的实际运行和监控。控制器发出命令，处理器模块执行具体的操作（如启动虚拟机实例、挂载磁盘等）。

f. 与外部系统的集成

集成外部存储和网络系统：控制器扩展模块支持与外部存储系统（如Ceph、NFS）和网络系统（如CNI）集成，以便为虚拟机实例提供存储卷和网络连接。

（3）处理器扩展模块

处理器扩展模块直接与K3S Node点上的虚拟机系统（如QEMU/KVM）交互，执行虚拟机实例的创建、启动、暂停、恢复、迁移等操作。处理器扩展模块 的主要功能如下：

a. 虚拟机实例的生命周期管理

启动虚拟机实例：处理器扩展模块在虚拟机实例（VMI）创建后，负责在宿主节点上启动虚拟机。它会调用底层虚拟化技术（如QEMU/KVM）来启动虚拟机实例，并确保虚拟机的状态与预期一致。创建虚拟机实例时所需的虚拟机镜像被包装成OCI标准容器镜像格式，被统一保存到容器镜像仓库中。经过扩展后的K3S会在需要时自动从镜像仓库中拉取虚拟机镜像。

暂停与恢复虚拟机实例：处理器扩展模块支持对虚拟机实例进行暂停和恢复。当虚拟机需要暂停时，处理器扩展模块 会调用虚拟化层的API暂停虚拟机的运行。当需要恢复时，它会恢复虚拟机实例的运行。

停止和销毁虚拟机实例：当虚拟机实例被删除时，处理器扩展模块会清理相关资源，停止并销毁虚拟机。

b. 虚拟机实例的状态监控

监控虚拟机状态：处理器扩展模块持续监控虚拟机实例的运行状态，确保它们处于期望的状态。如果虚拟机出现问题或状态不一致，处理器扩展模块 会尝试修复状态，确保虚拟机实例正常运行。

定期报告虚拟机状态：处理器扩展模块会定期更新虚拟机实例的状态并将其报告给K3S控制器。这包括虚拟机实例的健康状况、运行状态、资源使用情况等。

c. 虚拟机硬件资源管理

网络和存储配置：处理器扩展模块负责将虚拟机实例的网络接口和存储卷挂载到虚拟机中。它与K3S的CNI（容器网络接口）和CSI（容器存储接口）集成，确保虚拟机实例能够正确连接到网络并挂载所需的存储卷。

动态资源调整：处理器扩展模块可以根据虚拟机实例的配置调整虚拟机的资源分配，例如CPU、内存、磁盘等。如果虚拟机需要动态扩展或调整资源，处理器扩展模块会通知底层虚拟化平台进行相应的修改。

d. 虚拟机实例迁移

虚拟机实例迁移：处理器扩展模块 支持虚拟机实例的迁移操作。通过与底层虚拟化技术（如QEMU/KVM）配合，处理器扩展模块 可以在集群中的不同节点之间迁移虚拟机实例，支持虚拟机的高可用性和负载均衡。

迁移状态同步：在迁移过程中，处理器扩展模块 会持续监控虚拟机实例的状态并确保迁移过程的成功，保证虚拟机在迁移过程中不中断或丢失数据。

e. 与虚拟机控制器的交互

接收控制指令：处理器扩展模块与控制器模块协作，接收来自控制器的控制指令（如启动、停止、迁移虚拟机等）。控制器会将虚拟机的生命周期操作请求传递给处理器扩展模块，后者负责在宿主节点上实际执行这些操作。

更新虚拟机状态：处理器模块会向控制器报告虚拟机实例的运行状态，以便 控制器更新虚拟机的状态并进行进一步的操作。

（5）启动器扩展模块

主要负责在Node节点上启动和管理虚拟机实例（VMI）。它是与K3S集群中每个虚拟机实例（VMI）对应的一个Pod，负责确保虚拟机实例能够正确启动、运行和终止。启动器模块的主要功能包括：

a. 虚拟机实例的启动

启动虚拟机：启动器launcher在虚拟机实例（VMI）创建时启动，并且通过与底层虚拟机系统QEMU交互，在宿主节点上启动虚拟机实例并于具体的Pod绑定。

虚拟机实例配置：它会根据VMI的配置（例如虚拟机的CPU、内存、磁盘、网络等）生成相应的虚拟机启动命令，并确保虚拟机实例按照用户定义的方式启动。

监控虚拟机启动过程：启动器launcher监控虚拟机实例的启动过程，确保其在预期时间内成功启动并运行。

b. 虚拟机实例的运行时管理

管理虚拟机生命周期：启动器launcher持续运行并管理虚拟机实例的生命周期。在虚拟机实例运行期间，启动器launcher会与QEMU或其他虚拟化技术进行交互，处理如暂停、恢复、重启等操作。

虚拟机资源挂载：启动器launcher负责将虚拟机的存储卷和网络接口挂载到虚拟机实例中，确保虚拟机能够访问必要的资源。

监控虚拟机状态：它监控虚拟机的运行状态，检测虚拟机是否正常运行，并确保虚拟机在运行时的资源（如 CPU、内存）符合预期。

Pod生命周期管理：当虚拟机实例的状态发生变化时（如删除或停止），启动器launcher会与K3S的调度和控制平面交互，确保虚拟机实例的Pod被正确处理。

集群调度：启动器launcher与调度器集成，根据集群资源的状态（如CPU和内存使用情况）来调度虚拟机实例的运行。

c. 虚拟机实例的暂停与恢复

暂停虚拟机实例：启动器launcher支持通过与底层虚拟化平台的交互暂停虚拟机实例。当虚拟机实例被暂停时，启动器launcher会在宿主节点上执行暂停操作，并保持虚拟机的内存和状态。

恢复虚拟机实例：当虚拟机实例需要恢复时，virt-launcher 会与虚拟化平台交互，恢复虚拟机的运行状态，恢复其暂停时的内存和运行状态。

虚拟机编排调度系统每个软件单元说明如表4所示。

表4 虚拟机编排调度系统软件模块说明

|  |  |  |
| --- | --- | --- |
| **软件单元** | | **说明** |
| 控制器扩展模块 | 监听模块 | 持续获取资源的变更事件并将结果压入事件队列中 |
| 事件队列 | 保存资源变更事件的相关信息 |
| 本地缓存 | 用来存储资源对象并自带索引功能，本地缓存中的数据与Etcd集群中的数据保持完全一致 |
| 工作队列 | 保存资源对象的key，每次从该队列中取出key进行调谐操作 |
| 处理单元 | 真正执行调谐操作 |
| 调度器扩展模块 | 预选模块 | 过滤不符合要求的节点 |
| 优选模块 | 选取一个最合适的节点 |
| 绑定模块 | 将Pod和节点绑定，通知api-server |
| VM资源类型扩展模块（API扩展模块） | API资源扩展 | 利用容器编排调度平台支持自定义资源扩展机制扩展支持VM、VMI和VM RecplicateSet三种虚拟机相关资源类型 |
| 处理器模块 | 监视器模块 | 负责实时监听本节点上所有VMI对象和虚拟机实例的状态和事件 |
| 虚拟机管理模块 | 负责虚拟机的生命周期管理任务，并将虚拟机的状态信息实时同步给K3S集群 |
| 服务端模块 | 作为Web服务器，提供了RESTful类型的API，负责处理来自服务器模块对虚拟机的操作请求 |
| 启动器模块 | 虚拟机管理模块 | 负责管理由虚拟机统一控制引擎启动的虚拟机实例的生命周期 |
| 服务端模块 | 作为gRPC服务器，基于UNIX套接字，为启动器模块提供了一系列操作虚拟机的接口 |

这些模块之间的典型交互流程如所示：

（1）用户通过标准工具（Kubectl）将定义好的VirtualMachine（VM）对象的YAML文件应用到集群中。该文件描述了虚拟机实例的配置和需求；

（2）K3S API Server接收到用户的请求后，会解析YAML文件，并在集群中创建一个对应的VMI对象；

（3）控制器模块持续监听集群中的VMI对象，当检测到新的VMI对象被创建时，控制器模块会根据该VMI的配置信息生成一个对应的Pod对象；

（4）调度器模块监控到新创建的Pod对象后，会根据集群中各节点的资源情况、负载和调度策略，将该Pod调度到最合适的节点中；

（5）当Node节点上的Kubelet监听到Scheduler调度的Pod后，会在该节点上创建并启动该Pod。启动的Pod内部包含的启动器模块会准备接收处理器模块发送的虚拟机配置信息；

（6）处理器模块监听到节点上VMI对象的创建事件时，处理器模块会收集VMI对象的所有配置数据，并将这些配置发送给对应节点上的启动器模块；

（7）启动器模块接收到处理器模块发送的配置后，使用虚拟机统一管理系统在节点上创建实际的虚拟机实例。启动器模块负责管理虚拟机的整个生命周期，包括启动、停止、重启和销毁虚拟机实例。

图8 软件单元交互关系图

## 技术方案

### 容器云应用管理系统技术方案

#### 应用包管理模块技术方案

1. Chart应用包工程目录初始化

目录初始化组件，目的是为用户提供一个快速启动点来创建标准化的 Chart 包。其核心机制是创建 Chart 包的目录结果，然后和模板组件交互，通过内置模板生成符合 Helm 规范的目录和文件完成模板初始化，方便用户进一步修改和使用，并且通过CLI框架将程序封装，向外提供 CLI 接口。

1. Chart包工程目录打包

工程目录打包组件工作流程如图 9所示，作用是将Chart包打包成.tgz压缩文件，方便后续管理（如上传到云端Chart仓库）。Chart包工程目录打包如下流程图所示，根据路径导入Chart包，检测Chart包的依赖项，设置目的路径并调用包管理工具组件将 Chart打包归档保存在目的路径下。用户可通过命令行标志指定Version、appVersion 以及是否启用签名。



图 9 Chart包打包流程图

1. Chart包文件有效性检验

Chart包检验组件用于验证一个Chart包的语法、文件结构和模板逻辑，确保Chart符合规范并能正确渲染，方便用户在云原生应用Chart开发阶段发现潜在问题。Chart包文件有效性检验遍历Chart包的所有文件，每个文件进行语法检查（具体流程如下流程图），记录语法检查的消息，检查错误，最后返回语法检查结果。



图10 Chart包有效性检验流程

1. Chart包内容和可配置参数查看

Chart包的查看命令，由多个子命令组成，如show chart、show values 等子命令分别查看Chart包的元数据、默认配置、模板结构等内容，也可以使用show all展示所有信息。查看命令在执行时，首先载入目标Chart包，创建字符串缓存用于写入信息。然后分别根据命令要求，将Chart.yaml文件、Values.yaml文件等内容写入字符串缓存中，最后返回缓存。

1. Chart包依赖查看

Chart包依赖组件，用于管理Chart的依赖，支持查看、更新、添加或移除Chart的依赖项。在Chart包的Chart.yaml文件中的dependencies字段定义Chart的依赖，并通过charts/目录存储具体的依赖包。当执行dependency list命令查看该Chart包的依赖时，首先根据路径找到并载入Chart包，检查Chart.yaml文件的dependencies字段，以表格格式化形式输出依赖项信息。

1. Chart包下载

Chart 包下载组件用于从远程的Chart仓库中下载指定的Chart包到本地。它支持多种选项，可以选择指定版本、将Chart解压到本地目录等。执行流程：调用downloader组件创建ChartDownloader对象，检验OCI标准，从默认远程仓库或者指定仓库下载Chart包。另外，如果启用verify则验证Chart包，如果启用解压缩则利用包管理工具组件解压Chart包到指定目录，最后返回执行情况消息。



图11 Chart包下载流程

1. Chart包上传

将Chart包打包归档后得到的压缩文件推送到一个远程的仓库，方便Chart包的使用和管理。执行流程：创建一个字符串缓存用于记录操作结果。调用uploader组件配置上传选项，包括通过凭证（客户端证书、客户端私钥、CA证书）验证客户端身份，是否进行TLS认证，是否使用HTTP通信等。再验证远程仓库是否符合OCI标准，如果符合则将压缩Chart包推送当远程仓库，否则报错返回。

#### 云应用部署管理模块技术方案

1. 安装容器云应用

Chart包安装是核心命令之一，负责将Chart包应用部署到容器云集群中，Chart包可以是本地的Chart也可以使用远程仓库中的Chart包。通过该命令，用户可以将一个Chart转换为K3S资源并安装到指定的命名空间中，同时支持自定义配置和版本控制。执行流程如流程图 12所示。



图 12 云应用安装流程

1. 卸载容器云应用

Chart包卸载组件，用于从容器云集群中移除已经部署的云应用程序和关联的K3S资源。通过该命令，用户可以彻底删除指定云应用Release管理的资源，同时选择保留或清理与该云应用Release相关的历史记录。执行流程如流程图13所示。



图13 云应用卸载流程

1. 将指定容器云应用升级和回滚

应用的升级和回滚操作可以分为两部分，首先是构建K3S资源对象，其次是K3S资源的创建。

应用升级时，构建K3S资源流程如图14所示，首先从用户输入的命令中获取Chart包名称，检查是否有对应的安装实例，没有则进行安装；如果指定了升级版本，那么将从安装过的应用列表里找到对应版本进行安装，如果用户没有指定升级版本，那么将获取最新的安装版本。根据最终获取的Chart包，进行配置检查，对配置项进行配置重用和Chart包资源渲染。

应用回滚时，首先会计算出回滚的版本号，然后查询历史版本记录，找到对应的版本号的相关应用信息，根据已安装的应用信息可以直接构建K3S资源对象。相关流程如图15所示。



图14 云应用升级时资源创建流程



图15 云应用回滚时资源创建流程

构建好的资源会发送给K3S，进行应用升级和回滚；在升级或回滚前，会在K3S存储库中创建一条对应的Release发布记录，在应用升级完成后，会再对这条记录进行更新。如图16所示。



图16 K3S集群应用升级回顾流程

1. 查看已安装的云应用信息

查看已安装的应用信息流程如下，用户输入查询命令后，获取其中的查询条件进行查询，查询条件可能包括应用名称，应用状态，命令空间等；根据用户指定的输出结果格式，对查询结果进行处理后输出。

1. 查看云应用运行状态，版本变更信息

查看云应用运行状态流程如图17，首先会获取查询应用的版本号，结合应用名称查询K3S发布记录信息，其中会包含应用运行状态信息和版本信息；如果命令指定了showResource参数，则会解析Release的相关信息生成K3S资源信息，向K3S获取对应资源的当前状态，并将这些信息附加到返回结果中。



图17 云应用信息查看流程

1. 日志管理模块

志管理模块负责为整个系统提供与用户操作相关的日志管理功能，主要关注的是用户端而非系统内部的事件记录，目的在于提供系统活动的可追溯性，增强操作的透明度，以及确保操作的可问责性，满足内部监管的要求。

为了管理用户操作的日志信息，该模块主要提供了日志写入、日志查询和日志归档等功能，其中日志写入和查询是对后台SQLite数据库操作的封装，为其他模块提供了简单的接口；而日志归档是定期将数据库的内容导出到文件中，减轻服务端存储的压力，维持系统的可用性。

从技术实现的角度，日志管理模块可以分为以下几个部分：

1. 日志信息结构定义

后台SQLite数据库存储的是结构化数据，为了能够存储日志信息，首先需要定义日志信息结构，包括用户的标识、操作的类型、操作的结果以及操作的时间。其中用户标识是每个用户唯一的身份信息，在创建新用户时指定，通常为字符串类型；操作类型指明了本次操作的内容，如“上传某个Chart包”或“创建某个虚拟机”等等，定义为字符串类型；操作结果简单分为成功和失败，定义为布尔类型；操作时间是发起请求并被系统接收的时间点，定义为SQLite提供的日期类型；最后是唯一标识某条数据的自增主键。

1. 日志写入模块

日志信息中用户标识、操作类型和操作结果是无法事先预知的，而操作时间可以认为是发起日志写入请求的时间，所以API接口的参数为用户标识、操作类型和操作结果。该模块接收到写入请求后，从请求参数和当前系统时间中构造出完整的日志信息，并调用SQLite数据库提供的存储接口将日志信息写入其中。

1. 日志查询模块

在日志查询过程中，基于用户标识、操作类型和操作时间的查询较为常见且覆盖了绝大多数情况，所以该模块提供了这三种基本的查询，并在此基础上组合出更为复杂（如多条件）的查询接口。在接口设计上，这三种查询仅仅是查询条件的类型不同，即API参数的类型不同。该模块收到查询请求后，从请求参数中构造出查询条件，并调用SQLite数据库提供的检索接口查询日志信息，然后将查询结果反序列成内部的数据结构以待后续的进一步处理。

1. 日志归档模块

日志归档是指将某个时间点之前的数据导出到某个文件中，即该功能需要两个参数，所以该模块提供了一个对应的API接口，参数为SQLite数据库支持的日期类型和归档文件的文件路径。该模块收到归档请求后，根据给定的时间参数，查询出所有该时间前的数据，并将它们储存到一个新的数据库中，然后调用SQLite数据库提供的备份接口将新的数据库归档到给定的文件参数中，最后调用SQLite数据库提供的删除接口删去原数据库中该时间前的数据。此外，为了方便其他模块的使用，该模块还提供了另一个API接口，参数只有时间点，没有文件路径，此时数据会被归档到事先配置的默认文件中。

#### 应用包仓库管理模块技术方案

1. 添加，删除，列举Chart包仓库地址

包管理系统使用单独的一个yaml配置文件记录仓库相关信息，对仓库的增删改等操作本质上是对该配置文件的修改。添加和删除Chart包仓库时，首先需要加载读入仓库配置文件，然后对仓库记录相关内容进行增加和删除操作，进行添加操作时，还需对仓库进行有效性验证，修改完后更新文件信息。列举Chart包仓库地址功能只需要遍历配置文件中的仓库记录，在结果输出时支持yaml，json和表格格式输出。

1. 在指定仓库中添加和删除Chart，并更新索引文件

在仓库中添加Chart时，需要将Chart打包好之后上传到指定仓库目录下；同时会解析添加的Chart包下的chart.yaml文件，获取Chart相关信息，添加到仓库的索引文件中并更新索引文件。

在仓库中删除Chart文件时，将对应的Chart包文件删除，仓库的索引文件会删除详细的Chart信息，然后进行索引文件更新。

更新索引文件时，增加或删除对应Chart包相关信息，然后对Chart包信息按名称，版本等信息进行排序，在重新写入索引文件完成更新。

（3）Chart包仓库访问控制功能实现方案

Chart仓库访问控制模块基于Lua脚本扩展Nginx功能，结合本地SQLite数据库存储用户角色和权限信息，形成完整的RBAC（基于角色的访问控制）方案。整体方案划分分为前端模块配置、用户认证、角色查询与权限判断等部分。如图 18所示，具体说明如下：



图 18 Chart仓库资源访问控制功能实现

a. 前端模块配置

管理员可通过前端模块，设置用户角色和角色可访问的资源范围。

设置用户角色：管理员在前端界面中为用户分配角色，每个用户对应唯一角色。角色可以是管理员、开发者或查看者等不同级别。

设置角色可访问的资源：管理员定义每个角色可以访问的资源类别，如特定的Chart包资源、管理页面或查看权限等。

这些信息将存储到仓库服务器的本地SQLite数据库，作为后续访问控制的依据。

b. 用户认证

用户访问Chart仓库资源时，首先需要通过Nginx的用户名密码认证机制。Nginx接收用户的登录请求，基于自带的用户名密码认证方式，验证用户名和密码是否正确。认证成功后，用户名会传递到后续的处理阶段。

c. 角色查询与RBAC权限判断

完成认证后，基于Lua脚本查询用户的角色，并进行权限判断。

首先查询用户角色，通过Lua脚本与本地SQLite数据库交互，依据用户身份查询其分配的角色，在SQLite数据库中存储了用户与角色的对应关系。

系统根据查询到的角色信息，判断用户是否具备访问指定资源的权限。 如管理员角色可以访问所有资源，查看者角色仅具备查看权限，若用户角色不满足访问要求，系统将拒绝其资源访问请求，从而实现基于角色的访问控制。

#### 资源定义生成校验技术方案

1. 模版渲染模块实现

模板渲染模块由模板解析子模块和资源生成子模板两个部分组成。如图19。

模板解析子模块复杂将 Chart 包下的模板文件进行内容解析，将模板文本解析成语法树。具体来说，lex 先对模板文件进行词法分析，将源文本转换为一系列的词法单元 tokens。再由 parse 进行语法分析，通过对源文本的结构进行分析，以确认代码代码是否符合语言的语法规则，并生成一个结构化的表示，也即语法树。



图19 模版渲染流程

资源生成子模板复杂将解析完成的模板文件的语法树和上下文数据、values.yaml默认值结合起来，并逐步生成最终的渲染结果，生成 K3S 资源对象的信息。

1. 资源验证模块实现

该模块可以检验渲染得到的manifest清单文件是否合法。通过调用容器云平台交互模块的资源管控接口执行一个模拟安装，渲染模板后尝试将清单文件部署到K3s上，但并不会真正创建任何资源。其会报告任何错误，检验渲染得到的manifest清单文件是否合法，帮助用户在部署前发现问题。

#### Web UI管理工具技术方案

Web UI管理工具旨在为用户提供Web UI驱动的方式来查看已安装的Chart包、查看器修订的历史记录和相应的K3s资源，帮助用户导航和排除集群故障。提供的主要功能包括：

1. 总体布局和导航功能

提供一个全局视图，展示集群中已安装的云应用的状态、版本信息、命名空间等，轻松浏览K3s集群中的资源。

1. 云应用管理功能

用户可以查看已安装Chart包的详细信息（名称、Chart包版本、App版本、安装时间等），实时显示Pod的状态和资源关联（如Service、ConfigMap、Secret等）。另外提供了已安装Chart包的版本历史记录，通过历史记录用户可以进行简单的云应用版本管理，例如回滚到修订版本或升级到较新的版本。也可以通过版本历史记录，查看当前版本和过去修订版本的清单差异。

1. Chart包仓库管理功能

用户可以查看和管理仓库配置，如添加新的Chart包仓库、从已有的仓库中搜索指定的Chart包、查看Chart包的详细信息（描述、版本、维护者等）。

1. 测试功能

对于所有版本（已安装的Chart包），用户可以针对该版本执行测试。该项功能通过执行特定的测试任务（例如运行一个临时Pod或者Job）来验证Chart包是否成功将相应云应用部署在K3s集群上，以及云应用运行情况。

Web UI管理工具实现架构如图20示。

本方案将仿照Helm Dashboard，采用前后端分离的B/S架构。从技术实现的角度，该模块包含Web前端部分和Web后端部分，前后端之间用RESTful API方式交互：

1. Web前端模块

Web前端部分提供UI驱动用于可视化和管理K3s集群，为用户提供图形化界面来替代命令行操作，提高操作效率。本部分基于React框架实现，利用React框架丰富且强大的组件和样式构建Web前端UI界面，并调用后端提供的RESTful API实现逻辑功能。



图20 Web UI管理工具架构

1. Web后端模块

Web后端部分是一个基于bRPC框架的、C++实现的HTTP Server服务器，通过bRPC框架能够为前端提供高效安全的RESTful API接口，拥有优秀的延迟和吞吐量。后端具体的功能是通过调用Library API层的内部接口函数实现。

### 虚拟机编排调度系统技术方案

#### 控制器技术方案



图21 控制器设计

控制器通过持续监控集群中的资源状态，并根据需要调整或更新资源，确保系统的实际状态符合期望状态。

ListAndWatch 表示实现了List()和Watch()接口的一个RESTClient，用于向api-server发送Http请求，List()会被首先执行并且它仅被执行一次，用于先一次性获取所有资源集合，得到的结果将存入本地缓存中，之后通过Watch()持续获取资源的变更事件并将结果压入事件队列中。

事件队列是一个键值对结构，保存事件类型和对应的资源对象，监听到的每一个资源变更事件都被保存到该队列中，事件类型包括Added、Updated、Deleted，同时也会根据事件对本地缓存进行更新。

本地缓存用来存储资源对象并自带索引功能，本地缓存中的数据与Etcd集群中的数据保持完全一致。控制器可以很方便地从本地存储中读取相应的资源对象数据，而无须每次都从远程Etcd集群中读取，这样可以减轻K3S APIServer和Etcd集群的压力。

工作队列用于保存资源对象的key，该队列主要作用在于解耦事件队列的生产者和消费者，避免生产者阻塞丢失事件。

处理单元从工作队列中获取key，根据key从本地缓存中得到资源对象，通过比对资源对象的期望状态和实际状态执行实际的业务逻辑。

扩展定制资源仅仅是让K3S能够识别该资源，并提供了基于RESTful API的增删改查操作，也就是对结构化数据的数据库相关操作，而这些对实现虚拟化管理操作还远不够。要实现真正的虚拟化管理声明式API，还需要定制控制器。控制器是这么一类组件，它监听集群中的某些资源，根据这些资源的预期状态执行相关操作，控制集群逐渐调整至该状态，并更新这些资源的状态字段以反映集群当前状态。当监听的资源是定制对象时，这类模式又被称为Operator模式。

具体来说，控制器的控制流程如下：从K3S API中获取资源信息，根据资源的预期状态（spec），执行某些操作，然后向K3S API请求更新资源的状态信息（status）。因为绝大多数组件都有赖于从K3S API中获取资源信息，所以K3S API提供了一种List/Watch机制，以便客户端能够高效的检测资源变更，它允许客户端对对象或者集合发出初始请求get/list，然后跟踪自该初始请求以来的更改watch，为了实现该机制，K3S还为每个资源增加了资源版本（resourceVersion）字段，以此来标识某次请求。

**图 22 控制器执行流程**

（1）控制器采用HTTP API List/Watch API Server中指定的资源，然后将对应的事件和资源对象保存到事件队列中等待处理。

（2）在一个循环中从事件队列中拿出资源对象进行处理。

（3）将从事件队列中拿出的资源对象放到本地缓存中。该缓存提供了索引功能，允许基于特定条件（如标签、注释或字段选择器）快速有效地查找资源。

（4）调用对应的事件处理函数，将资源对象的变化通知到对应的执行逻辑。

（5）在事件处理函数中对资源对象的变化进行处理。事件处理函数先将收到的消息放入到一个队列中，然后在一个循环中处理该队列中的消息；

（6）控制器处理逻辑先从工作队列中获取一个对象key，再从本地缓存中获取key对应的资源对象执行调谐逻辑。

（7）向API Server发送HTTP请求，更新资源状态。

#### 资源扩展模块技术方案

为了将虚拟机资源纳入到天脉容器编排调度软件K3S的统一管理，使K3S可以以一致的方式混合编排和调度虚拟机实例、容器实例，必须对K3S能够支持的资源类型进行扩展。

K3S提供了（CRD：CustomResourceDefinitions）自定义资源服务，可通过CRD机制对K3S资源进行扩展，使其支持VM、VM和VMIRS（Virtual Machine Instance ReplicaSet）。CRD服务通过CustomResourceDefinitions对象进行管理，并通过extensionsapiserver.Scheme资源注册表管理CRD相关资源。

定制资源是对K3S API资源的扩展，它可以通过动态注册的方式在运行中的集群内或出现或消失，集群管理员可以独立于集群更新定制资源。一旦某定制资源被安装，用户可以使用kubectl来创建和访问其中的对象，就像他们为Pod这种内置资源所做的一样。

当集群管理员创建新的CRD资源时，K3S API会为该资源的每个版本生成一个RESTful资源路径。通过使用标准的HTTP方法，如GET、POST、PUT、DELETE、PATCH等访问该路径，用户能够对该定制资源执行增删改查类别的操作，并且K3S会保证这些操作符合定制资源的定义。从某种意义上来说，CRD为定制资源提供了模式，可以类比为面向对象中类的概念，而定制资源则是类的实例对象，与现实世界中的实际资源相对应。

在扩展定制资源时，首先要创建CRD对象，规定该资源的属性和状态信息，定义中核心部分有组（group）、版本（version）、名称（name）和字段定义（spec），其中组、版本和名称唯一标识了定制资源，被用在RESTful资源路径中，格式为/apis/<group>/<version>/namespaces/<namespace>/<names>/，而字段定义则规定了该资源有哪些属性可供配置，并为K3S API提供对该资源的验证功能。接下来就是将该定义应用到集群中，假定CRD的定义存放在crd.yaml文件中，集群管理员则可使用kubectl apply -f crd.yaml命令应用该定制资源，成功后用户就能够像管理Pod等内置资源一样管理定制资源，例如，可以使用命令kubectl create创建定制对象，使用命令kubectl replace/patch修改定制对象，使用命令kubectl delete删除定制资源，使用命令kubectl get获取定制对象信息。

为了使K3S能够管理虚拟机资源，需要扩展支持以下几种定制资源：Virtual Machine Instance(VMI)资源类型用于定义无状态匿名虚拟机实例、Virtual Machine(VM)资源类型用于定义带持久状态命名虚拟机、Virtual Machine Instance ReplicaSet(VMIRS)资源类型用于定义多副本虚拟机集群。具体来说：

* 对于VMI资源，它的属性字段中应包含虚拟机的CPU、存储、网络等的要求；状态字段中应包含虚拟机运行的节点名、Pod信息和当前的运行状态，如运行、暂停、崩溃等。
* 对于VM对象，它的属性字段中应包含虚拟机的CPU、存储、网络等的要求以及启动设备信息；状态字段中应包含虚拟机的当前运行状态。
* 对于VMIRS对象，它管理多个类似的VMI对象，它的状态字段中除了与虚拟机运行有关的信息外，还包含VMI对象的数量以及VMI对象的选择算符；状态字段中包含所有VMI对象的运行状态的总体概要信息。

#### 处理器模块技术方案

图 23 处理器模块设计

该模块以DaemonSet的形式部署在集群中，K3S自身会确保每个节点上都有一个实例在运行。该模块的主要职责有监听本地节点上VirtualMachineInstance对象（来自K3S API Server）和虚拟机实例（来自启动器模块），并执行相关操作以维持两者状态的一致性；以及接收来自服务器模块的HTTP请求，并调用启动器模块的对应接口，以此提供虚拟机启动、停止、暂停、恢复、迁移和状态监控等功能。

监视器子模块：该子模块负责实时监听本节点上所有VMI对象和虚拟机实例的状态和事件。这些事件包括VMI对象的创建和删除，虚拟机实例的启动、停止、暂停、恢复和崩溃等。该子模块会捕捉这些事件，并将数据信息报告给虚拟机管理子模块，由其负责VMI对象和虚拟机实例的状态保持一致。

服务端子模块：该子模块作为Web服务器，提供了RESTful类型的API，负责处理来自服务器模块对虚拟机的操作请求，主要有启动、停止、迁移和监控等。该子模块会根据请求的类型，调用虚拟机管理子模块的相应接口，并将请求的结果返回给服务器模块。

虚拟机管理子模块：该模块是处理器模块的核心，负责虚拟机的生命周期管理任务，并将虚拟机的状态信息实时同步给K3S集群。一方面，它直接执行本节点的VMI对象和虚拟机实例的调谐操作，维持两者状态的一致；另一方面，它接收来自服务端子模块的虚拟机实例管理请求，执行相应的操作，将指令通过基于UNIX套接字的gRPC请求传达给启动器模块。

处理器模块可以看作是定制控制器和服务器的组合，它的功能主要可以分为三个部分：

（1） 监听本节点VMI对象的状态变更，该功能可以通过调用K3S API提供的List/Watch机制来实现。首先向K3S API Server发送VMI对象的List请求，然后根据返回的资源版本再发送Watch请求，这样之后所有VMI对象的变更都会通过chuncked响应的形式通知给处理器模块，但每个节点上的处理器模块实际上只关心本节点上VMI对象的变更，所以这里还会有一个选择操作，该操作可以通过VMI对象的标签或者nodeName字段来实现。经过上述流程，实际接收到的就是本节点VMI对象的状态变更，处理器根据预期状态执行对应的操作来更新当前状态。

（2） 监听虚拟机实例的状态变更，该功能可以通过调用启动器模块的接口来实现。实际上，管理单个虚拟机实例的是每个Pod中的启动器模块，所以处理器模块想要得知每个虚拟机实例的状态，就必须向对应Pod内的启动器模块发送请求来获取。为了方便两者的通信，可以设计一个Domain定制资源，它代表虚拟机实例的当前状态，每当需要得知虚拟机状态时，启动器模块会返回该资源的实例给处理器模块。

（3） 处理虚拟机实例的生命周期管理操作请求，如启动、暂停、恢复、停止等。为了实现该功能，可以在处理器模块内部启动一个RESTful API服务器，提供虚拟机操作的接口。此外，既然实际管理虚拟机的是启动器模块，所以还要将这些请求转发给该模块，这可以通过在启动器模块中启动gRPC服务器，而处理器模块作为客户端请求相应接口来实现。目前来说，处理器模块在执行虚拟机生命周期操作时，是能够得知该虚拟机运行在哪个Pod中的，所以可以有两种通信方式，一种是基于TCP/IP协议，还有一种是基于UNIX套接字。

基于UNIX套接字的方案要求处理器模块和启动器模块之间能够共享某些文件，这需要在两者间创建节点级别的额外共享目录来实现，优点一是安全性高，可以保证启动器模块只会接收来自当前节点上的处理器模块的请求，因为该套接字仅会出现在本节点内；优点二是传输效率高，因为数据不用经过网络协议栈。缺点是耦合性较高，创建额外的共享目录也提升了实现复杂度和维护成本。

基于TCP/IP协议的方案只需要处理器模块和启动器模块之间协商好通信所用的端口，因为每个就绪Pod都有集群内唯一的IP地址，处理器模块可以通过它来请求启动器模块提供的服务，优点是耦合性低，实现简单，可维护性高；缺点是安全性稍差，理论上存在某个启动器接收来自其他节点上处理器请求的可能。考虑到本系统的应用场景较为封闭，安全性可以暂时不做考虑。

综上所述，本方案计划采用基于TCP/IP协议的套接字通信。

#### 启动器模块技术方案



图 24 启动器设计

该模块作为普通的Pod运行在集群中的某个节点上，并关联到一个VMI对象。该模块的主要职责是接收来自处理器模块的请求，主要有启动、停止、暂停、恢复、迁移和监控虚拟机等，并根据请求的类型，调用虚拟机统一控制引擎提供的相关接口来控制虚拟机，以及将虚拟机的状态同步给处理器模块。

服务端子模块：该子模块作为gRPC服务器，基于UNIX套接字，为启动器模块提供了一系列操作虚拟机的接口。当收到gRPC请求后，它会调用虚拟机实例管理子模块的相应接口，由其负责将请求转化为对虚拟机统一控制引擎接口的调用，同时将结果返回给启动器模块。

虚拟机实例管理子模块：该子模块负责管理由虚拟机统一控制引擎启动的虚拟机实例的生命周期，直接与虚拟机统一控制引擎交互，是链式调用的最后一层，向外它输出虚拟机实例的状态信息，向内它执行虚拟机管理操作。从某种意义上，它是对虚拟机统一控制引擎的一层薄封装，但保证了有且仅有一台虚拟机实例可供运行。

启动器模块的主要功能是接收处理器模块的请求，然后调用虚拟机统一控制引擎执行虚拟机管理操作。

为了能够接收处理器模块的请求，启动器模块内部需要运行一个服务器，根据实际的需求，这里选择使用gRPC作为服务类型。通信的底层可以选择TCP/IP协议，或者是UNIX套接字，前者的好处是耦合性较低，但是隔离性较差，处理器模块可能接收到来自其他节点处理器模块的请求；而后者的好处是性能较好，但是需要额外的存储共享策略，例如共享kubelet的临时存储。启动器模块还需要事先定义一系列与虚拟机管理操作有关的函数，并将它们和具体的请求路径绑定。每当处理器模块请求某个路径时，服务器就会负责转到实际的函数去处理。

启动器模块还要能够调用虚拟机统一控制引擎执行虚拟机管理操作，所以要求启动器模块也能够管理虚拟机统一控制引擎的生命周期，例如在虚拟机统一控制引擎崩溃后重启，这可以通过将虚拟机统一控制引擎作为启动器模块的子进程来实现。此外，启动器模块还要维持和虚拟机统一控制引擎的连接，以便能够实时监测虚拟机实例的状态，并能够及时响应对虚拟机的管理操作请求。最后，由于启动器模块和处理器模块之间通信的主要媒介是Domain定制资源，所以还要负责将虚拟机的实际状态转化为该资源的实例。

#### 虚拟机统一控制引擎

虚拟机统一控制引擎层级以及功能模块划分如下：



图 25 虚拟机统一控制引擎层级结构及功能模块划分

其中各层模块主要负责的功能如下：

功能接口层主要为用户提供操作接口，用户可以通过调用该层提供的接口进行对虚拟机的相关操作，包括创建、管理虚拟机，监控虚拟机状态以及配置虚拟网络和存储等。该层封装了统一的抽象接口，通过抽象化操作实现对不同虚拟化技术的支持，用户调用接口后会根据用户指定的虚拟机类型执行对应的操作，来实现对不同类型的虚拟机的功能操作。

驱动实现层主要是针对需要支持的不同的虚拟化技术类型，进行的功能接口层所定义的用户功能接口的实现以及自身平台相关的功能支持。以QEMU驱动模块为例，接口实现部分是针对功能接口层所定义的接口根据QEMU所需要的业务逻辑进行了具体的实现，用户在调用功能接口对QEMU管理的虚拟机进行操作时会调用该模块的具体实现来完成用户执行的操作；功能支持部分则是实现QEMU操作业务逻辑所需要的具体功能，如与QEMU通信所需要的QMP（QEMU Monitor Protocol）的monitor封装以及消息传递功能等。系统所支持的每个虚拟化技术类型都需要实现对应的驱动模块，这些模块会在系统启动时进行注册，从而用户能够调用统一的抽象层接口来操作不同类型的全虚拟化虚拟机。

底层通用模块层主要是系统所需要的通用功能支持，该层与QEMU以及Linux所提供的库函数进行直接交互，并进行适当的封装，为整个系统提供底层的功能支持，包括内存管理、进程与子进程管理、事件与线程相关操作以及基础工具类如文件操作、字符串处理等，从而支持驱动实现层的开发和功能接口层的运行，确保高效、稳定的基础功能。上层的功能接口通过调用该层所提供的通用功能，能够方便的实现所需要的功能业务逻辑。

每个模块说明如下表所示：

|  |  |  |
| --- | --- | --- |
| **功能模块** | | **说明** |
| 功能接口模块 | 虚拟机管理子模块 | 虚拟机的创建、删除、迁移等操作接口 |
| 虚拟机控制子模块 | 虚拟机的启动、停止、状态监控等操作接口 |
| 虚拟网络子模块 | 虚拟网络配置与管理等操作接口 |
| 虚拟存储子模块 | 虚拟存储池和卷的管理操作接口 |
| 驱动实现模块 | QEMU接口实现子模块 | 功能接口针对QEMU的实现 |
| QEMU功能支持子模块 | QEMU相关的功能支持的实现 |
| 其他虚拟机接口实现子模块 | 功能接口在其他虚拟化技术的实现 |
| 其他虚拟机功能支持子模块 | 其他虚拟化技术相关的功能支持的实现 |
| 底层通用模块 | 网络管理子模块 | 虚拟机网络管理相关的底层工具功能 |
| 存储管理子模块 | 虚拟机磁盘和存储卷相关的底层工具功能 |
| 进程子进程管理子模块 | 子进程的创建、销毁、管理等功能支持 |
| 事件与线程管理子模块 | 多线程环境事件调度与锁机制等功能支持 |
| 通用基础工具子模块 | 基础通用功能，如文件操作、字符串处理等 |

功能模块调用关系参考如下图所示。



图 26 功能模块调用关系图

用户接口层中的抽象驱动模型定义了管理虚拟机所需要的各种接口，之后所有支持的虚拟化技术类型对应的驱动模块都需要分别对控制虚拟机以及虚拟机网络、存储相关的驱动功能进行实现。每种类型的驱动模块中都定义了注册函数，在系统初始化时，每种驱动模块会使用该注册函数将自己的驱动类型注册到系统中，当用户使用系统提供的接口进行功能调用时，会根据系统中注册的驱动类型定位到用户指定的虚拟机类型，执行对应的业务逻辑，并通过调用底层通用模块提供的功能支持完成用户所需要的操作。底层通用模块则是直接调用操作系统所提供的各类基础运行时库（如pthread、glib等）并进行了一定的封装，同时可以通过构造QEMU命令等方式直接与QEMU等虚拟化平台进行交互，为上层功能提供底层的工具支持。

系统所提供的的几个关键接口设计如下：



图 27 关键接口类型设计

其中虚拟机连接表示与虚拟化管理程序的连接，是操作虚拟化环境的入口点，对所有虚拟资源的操作都需要从该对象发起。用户执行功能操作时，需要先通过传入指定的URI等方式来获取虚拟机连接，之后可以通过虚拟机连接来获取到对应虚拟化技术类型的驱动对象，从而调用所需要的功能函数。

虚拟机域接口用于提供和虚拟机相关的操作，包括对虚拟机生命周期管理的接口，如创建、删除虚拟机以及节点间虚拟机的迁移等，以及提供对虚拟机实例的管理控制功能，如启动、暂停、恢复、停止以及监控虚拟机实例的状态信息等。

虚拟网络接口提供对虚拟网络生命周期的管理接口，如虚拟网络的定义、启动、销毁等，用于管理虚拟机之间的通信以及与外部网络的通信，并支持网络配置的查询和更新等操作。

存储池和存储卷接口用于提供虚拟机的存储相关功能。存储卷通常是虚拟机所使用的块设备（如虚拟磁盘等），该接口提供对存储卷的管理功能，如创建、删除、扩展以及拷贝等操作，并支持存储卷内容的读写操作，存储卷与虚拟机相关联，为虚拟机提供存储资源。而存储池则是虚拟化环境中存储卷的逻辑分组，该接口用于管理存储资源的分配。存储池可以是本地目录、分布式存储系统活其他形式的存储后端。

每种类型的虚拟机都需要在系统初始化时将自身驱动注册到系统中，用户获取连接对象时，需要传入指定的URI用于确定对应的虚拟机类型。系统初始化时注册虚拟机驱动以及用户获取指定类型虚拟机连接的流程如下：



图 28 系统初始化注册驱动流程

系统启动时会进行驱动注册流程，系统调用初始化函数，加载系统所支持的各个驱动模块，每个驱动模块中定义了注册函数，加载驱动模块的过程将调用其驱动函数，模块各自的驱动函数中会调用统一的系统驱动注册函数，将对应的虚拟机驱动注册到系统的驱动表中（图中以QEMU和VBOX类型的虚拟机驱动为例）。

系统初始化完成后，用户需要通过获取虚拟机连接来调用系统提供的虚拟机管理操作接口，用户需要传入指定虚拟机类型对应的URI调用系统提供的open()接口，系统会解析用户传入的URI内容，并根据此信息遍历查找系统驱动表寻找对应的驱动，找到对应驱动后，会构造包含对应驱动的虚拟机连接对象，并将其返回给用户，用于调用其他的虚拟机管理功能操作。

整个系统驱动模块加载以及后续用户调用功能接口的时序图如下所示。



图 29 系统初始化注册驱动以及执行后续功能时序图

用户通过传入指定URI获取到虚拟机连接对象后，便可以创建虚拟机。以QEMU为例，用户创建虚拟机流程的关键步骤如下：



图 30 创建虚拟机流程图

用户通过传入open接口获取到的虚拟机连接对象以及创建虚拟机的xml配置文件调用系统提供的创建虚拟机接口，系统会根据用户传入的连接对象定位到对应虚拟机类型的驱动模块，从而调用该驱动的创建虚拟机接口实现。

在驱动类型为QEMU的情况下，系统将首先将用户传入的xml文件解析为虚拟机配置，并根据此信息创建一个虚拟机实例对象，之后将这个对象添加到QEMU的虚拟机列表中，用户可以根据虚拟机的uuid以及name获取到虚拟机对象。虚拟机对象创建完成后，将根据配置信息构建QEMU创建虚拟机的命令，之后系统将fork出一个子进程通过exec等函数创建一个QEMU进程并执行创建虚拟机的命令，此子进程的pid会保存到上述创建的虚拟机对象中，便于用户后续对虚拟机进行功能操作。

执行虚拟机创建命令后的操作中，将在虚拟机对象中添加一个Monitor以及一个Agent用于与QEMU虚拟机进行交互。

其中Monitor对象通过QMP（QEMU Monitor Protocol）与虚拟机通信。QEMU对外提供了一个socket接口，称为QEMU Monitor，而QMP是一种基于JSON格式的传输协议，可以通过QEMU Monitor使用QMP与一个QEMU虚拟机实例进行交互。

而Agent主要用于与QGA（QEMU-Guest-Agent）进行交互。QGA是运行在虚拟机内部的一个守护程序（QEMU-guest-agent.service），可以用于管理应用程序，执行宿主机发出的命令。QEMU为宿主机和虚拟机提供了一个数据通道（channel），这个通道的两端分别是在虚拟机内看到的串口和在宿主机上看到的unix socket文件。宿主机与虚拟机内的QGA通讯就扩展了对虚拟机的控制能力，例如在宿主机上获取虚拟机的IP地址等。

系统提供了通过QMP或者QGA发送指令的接口，并将部分指令操作封装为接口以简化用户的调用。

以暂停虚拟机运行为例，流程如下：



图 31 暂停虚拟机操作流程图

用户可以直接调用系统提供的发送Monitor命令接口或者调用系统封装过的暂停虚拟机接口来执行虚拟机暂停操作。系统会根据用户传入的命令参数构造对应的QMP命令，并通过Monitor对象发送给QEMU Monitor，QEMU之后执行对应的操作，停止虚拟机的CPU运行，并关闭虚拟机对外开放的网络接口等，之后系统将虚拟机状态更新为暂停状态并保存，便于后续的恢复等操作。

通过QEMU Monitor和QGA的相互配合，用户可以实现对虚拟机生命周期的管理操作以及对虚拟机运行状态的监控。

系统还提供了在不同节点迁移虚拟机的操作，迁移虚拟机流程如下：



图 32 虚拟机迁移流程图

主要分为以下五个阶段：

（1） 开始阶段：在源主机中根据需要迁移的虚拟机生成对应的XML配置文件，用于传递给目标主机并创建虚拟机。

（2） 准备阶段：目的主机准备接收来自源主机的虚拟机

（3） 执行阶段：之后源主机开始迁移虚拟机，等待目标主机接收完成并创建虚拟机

（4） 完成阶段：目标主机等待迁移完成并检查是否成功，若迁移失败则将终止目标主机上的虚拟机。结果将返回给源主机。

（5） 确认阶段：源主机检查迁移过程的结果，若迁移成功则终止源主机上的虚拟机，否则恢复虚拟机，完成整个迁移过程。

用户执行迁移功能后即会启动上述流程，其中每个过程都有对应的QEMU或其他虚拟化技术驱动实现接口，用于根据虚拟机类型完成对应的操作。整个过程的时序图如下所示。

图 33 虚拟机迁移时序图

## 内部接口设计

### 容器云应用管理系统内部接口设计

（1）Chart包管理模块对外接口：

PackageChart()：该接口用于将指定的 Chart 应用工程目录打包成一个 .tgz 格式的 Chart 包。

ValidateChart()：该接口用于对 Chart 包文件进行有效性检验。通过此接口，用户可以检查 Chart 包是否符合的标准。如不符合，系统会返回相应的错误信息。

ViewChartConfigParams()：该接口提供查看指定 Chart 包中可配置的参数列表。以及这些配置项的默认值。

ViewChartDependencies()：该接口提供查看指定 Chart 包的依赖信息。

（2）模版渲染模块对外接口设计如下：

renderResource( )：此方法负责渲染 Helm 模板。它将读取指定的模板文件，解析并将其与 values.yaml 配置文件及其他用户提供的配置数据进行合并，生成符合 K3S 配置标准的资源对象。此方法可用于 Helm Chart 的安装、升级等操作中，确保生成的资源文件可以成功部署到 K3S 集群。

（3）云应用管理模块对外接口：

Install(): 基于Chart 包文件安装容器云应用，指定发布名称。若发布名称已存在，则会提示冲突。

UnInstall()：卸载已安装的容器云应用。通过指定 release-name，删除与该发布相关的所有集群资源。

UpGrade(): 将指定容器云应用升级至最新版本或者指定版本。

RollBack(): 将指定容器云应用回滚至上一版本或者指定版本。

（4）仓库管理模块对外接口：

DownloadIndexFile()：下载并验证仓库的索引文件，确保仓库可访问。如果仓库不可达，返回相应错误信息。

NewChartRepository():方法用于创建一个新的 Chart 仓库对象，配置仓库的基本信息，如仓库 URL、认证信息等。

RepoAdd():此方法用于向本地仓库管理配置中添加一个新的仓库信息，并更新配置文件，以便后续操作可以使用该仓库进行 Chart 管理

RepoRemove:( ) 此方法根据提供的仓库信息，从本地仓库配置中移除指定的仓库。执行此操作后，相关仓库将不再出现在仓库列表中。

RepoList（）：此方法用于列出所有已配置的仓库，并返回仓库的详细信息，包括仓库名称、URL 和认证状态等。

RepoIndex（）：此方法负责生成或更新仓库的索引文件，索引文件包含仓库中所有可用的 Chart 及其元数据。索引文件通常用于加速 Chart 的查找和下载。

IndexAdd()：此方法用于将一个新的 Chart 添加到仓库的索引文件中。更新后的索引文件会反映出新的 Chart 信息，以便用户在后续操作中能够检索到该 Chart。

IndexDelete():此方法用于在删除 Chart 后，更新仓库的索引文件，移除已删除的 Chart 信息。更新后的索引文件将确保仓库中不再包含该 Chart 的信息。LoadIndexFile（）：此方法用于加载本地或远程仓库的索引文件，以便对仓库中的 Chart 进行管理。加载后的索引文件会存储在内存中，供后续操作使用。

AddChartToRepo():此方法将一个新的 Chart 添加到指定的仓库中，并更新仓库的索引文件和存储。添加后，Chart 将出现在仓库的可用列表中。DeleteChartFromRepo（）：此方法将一个已存在的 Chart 从仓库中删除，并更新索引文件，确保删除后的 Chart 不再出现在仓库中。

### 虚拟机编排调度系统内部接口设计

#### 资源扩展模块接口

与K3S API Server关系：服务器模块接口通过扩展K3S API Server，提供了一套与虚拟机相关的API。这些API可分为两部分，一是基于定制资源（CRD）的扩展，例如VirtualMachine(VM)、VirtualMachineInstance(VMI)、VirtualMachineInstanceReplicaSet(VMIRS)等。通过这些扩展，用户可以向管理Pod一样来管理虚拟机实例。这些API允许集群管理员和用户创建、更新、删除虚拟机对象，并触发相应的操作，如启动和停止虚拟机。另一是基于API聚合层的扩展，它申领了某些特定的URL路径。当用户通过RESTful API访问这些路径时，K3S API Server会将这些请求转发给定制的服务器。目前，这些API包含启动、停止、暂停、恢复和迁移虚拟机。

与用户接口关系：用户可以通过命令行工具kubectl或者virtctl与服务器模块接口进行交互。这些工具向K3S API Server发送请求，以执行虚拟机管理操作。用户可以通过kubectl提交YAML配置文件，创建或更新VirtualMachine和VirtualMachineInstance等对象，或者通过virtctl使用简化的命令启动、停止虚拟机。这种接口设计使得用户能够以K3S原生的方式管理虚拟化工作负载，同时保留了使用虚拟机的灵活性。

#### 控制器模块接口

控制器模块持续监视K3S API Server中与虚拟机相关的资源对象，如VirtualMachine和VirtualMachineInstance(VMI)等。当这些对象发生变化时（如创建、删除或状态更新），控制器模块会根据设定的策略做出相应的操作。例如，当检测到新的VMI对象时，控制器模块会负责调度该虚拟机实例到合适的节点。而当VMI对象被删除时，控制器模块会清理相应的资源，如果该VMI对象来自于某个VM对象，它还会负责同步VM对象的状态。控制器模块的这种设计确保了虚拟机实例的自动化管理和高效调度，同时保持集群状态的一致性。

#### 处理器模块接口

与服务器模块关系：处理器模块提供了RESTful API，接收并处理来自服务器模块的指令，主要负责执行虚拟机的生命周期管理操作，包括启动、停止、暂停、恢复、迁移和监控虚拟机实例。处理器模块确保这些操作在节点上正确执行，并将执行结果反馈给集群。通过这种机制，处理器模块在节点层面实现了对虚拟机实例的精细控制，并保持与集群的整体一致性。

与启动器模块关系：处理器模块充当指令传递的桥梁，将服务器模块下达的指令传达给启动器模块。启动器模块负责执行这些指令，具体来说，启动器模块通过调用底层的虚拟化管理工具（如虚拟机统一控制引擎和QEMU）来实现虚拟机的启动和停止操作。处理器模块确保启动器模块准确地接收到指令，并在必要时执行相应地错误处理或重试操作。这里采用的是基于UNIX套接字的gRPC通信协议，处理器模块作为客户端，启动器模块作为服务端。

#### 启动器模块接口

启动器模块提供了基于UNIX套接字的gRPC接口，接收并处理来自处理器模块的指令，并通过调用虚拟机统一控制引擎的API来管理虚拟机的生命周期。虚拟机统一控制引擎是虚拟化管理的一层抽象，封装了对虚拟机的各种操作，并对外提供统一的API接口。启动器模块利用这些API来执行虚拟机额创建、启动、暂停、恢复和迁移等操作。启动器模块的这种接口设计使得它能够在不同的虚拟化平台之间灵活切换，同时保持一致的操作接口。

#### 虚拟机统一控制引擎接口

描述各组件之间的接口设计，必要时给出图表说明。

系统的功能接口层对外提供的接口示例如下：

connectOpen()：用于获取虚拟机连接，用户获取到虚拟机连接对象后可通过将其作为参数传入来调用系统提供的其他功能接口。

domainCreateXML()：通过XML文件创建虚拟机对象，用户传入虚拟机连接，系统会根据虚拟机连接定位到对应的虚拟机类型驱动，执行对应的创建函数，创建虚拟机实例并添加到虚拟机列表。

domainLookupByUUID()：通过UUID获取虚拟机对象，此接口会根据用户传入的连接参数查询对应驱动的虚拟机列表，通过UUID获取对应的虚拟机对象并返回给用户，以用于虚拟机相关的其他功能接口的调用。

domainLookupByName()：通过虚拟机名称获取虚拟机对象，此接口会根据用户传入的连接参数查询对应驱动的虚拟机列表，通过虚拟机名称获取对应的虚拟机对象并返回给用户，以用于虚拟机相关的其他功能接口的调用。

domainShutdown()：关闭虚拟机，用户传入指定的虚拟机域对象，系统将定位到对应虚拟机类型驱动的实现函数来完成关闭虚拟机的功能，如QEMU将通过Monitor对象向QEMU Monitor发送JSON命令来控制虚拟机进行关闭。

domainSuspend()：暂停虚拟机，用户传入指定的虚拟机域对象，系统将定位到对应虚拟机类型驱动的实现函数来完成暂停虚拟机的功能，如QEMU将通过Monitor对象向QEMU Monitor发送JSON命令来控制虚拟机进行暂停。

domainResume()：继续虚拟机，用户传入指定的虚拟机域对象，系统将定位到对应虚拟机类型驱动的实现函数来完成暂停后继续运行虚拟机的功能，如QEMU将通过Monitor对象向QEMU Monitor发送JSON命令来控制虚拟机进行暂停后的继续运行。

domainMigrate()：迁移虚拟机，用户指定需要迁移的虚拟机以及目标主机，执行虚拟机的迁移操作，迁移过程将涵盖2.5中描述的五个阶段。

驱动实现层为上层提供的支持接口如下（以QEMU驱动为例）：

QEMUConnectOpen()：获取QEMU虚拟机连接的实现接口，识别到用户传入的URI为QEMU类型后，将对其进行进一步的解析，创建一个虚拟机连接对象并将QEMU驱动对象保存到其中返回给用户，之后用户传入该对象进行其他功能接口的调用时，将依此定位到QEMU驱动。

QEMUDomainCreateXML()：通过XML文件创建QEMU虚拟机对象，用户传入QEMU类型的虚拟机连接，系统会根据虚拟机连接定位到QEMU虚拟机类型驱动，执行该创建函数，构造QEMU创建命令，fork子进程并调用execv执行命令创建虚拟机实例并添加到QEMU驱动的虚拟机列表。

QEMUDomainLookupByUUID()：通过UUID获取QEMU虚拟机对象，此接口会根据用户传入的QEMU类型的连接参数查询QEMU驱动的虚拟机列表，通过UUID获取对应的虚拟机对象并返回给用户，以用于虚拟机相关的其他功能接口的调用。

QEMUDomainLookupByName()：通过虚拟机名称获取QEMU虚拟机对象，此接口会根据用户传入的QEMU类型的连接参数查询QEMU驱动的虚拟机列表，通过虚拟机名称获取对应的虚拟机对象并返回给用户，以用于虚拟机相关的其他功能接口的调用。

QEMUDomainQEMUMonitorCommand()：发送QMP指令到QEMU Monitor。此接口能够根据用户传入的指令类型以及参数构造JSON格式的QMP指令，并通过系统与QEMU虚拟机建立的连接将指令发送给QEMU Monitor以执行控制虚拟机、监控虚拟机资源状态等操作。

QEMUDomainQEMUAgentCommand()：发送指令到QEMU Guest Agent。此接口能够根据用户传入的指令类型以及参数构造JSON格式的QGA指令，并通过系将指令发送给QEMU虚拟机内部的QEMU Guest Agent以控制虚拟机内部行为或监控虚拟机的内部运行状态等。

QEMUDomainMigrateBegin()：虚拟机迁移开始阶段函数的QEMU实现，在源主机上执行，生成创建QEMU虚拟机对应的XML文件并传递给目标主机。

QEMUDomainMigratePrepare()：虚拟机迁移准备阶段的QEMU实现，在目标主机上执行，目标主机将根据源主机传递的XML文件创建QEMU虚拟机，准备接收来自源主机的虚拟机运行状态数据。

QEMUDomainMigratePerform()：虚拟机迁移执行阶段函数的QEMU实现，在源主机上执行，将通过QEMU Monitor获取需要迁移的虚拟机的运行状态数据，并通过与目标主机建立的连接进行数据传输。

QEMUDomainMigrateFinish()：虚拟机迁移完成阶段函数的QEMU实现，在目标主机上执行，目标主机将确认虚拟机是否迁移成功，并将结果返回给源主机，在迁移失败的情况下会终止失败的虚拟机。

QEMUDomainMigrateConfirm()：虚拟机迁移确认阶段的QEMU实现，在源主机上执行，将根据目标主机返回的迁移结果执行对应操作，迁移成功时将终止源主机上的对应虚拟机，否则将其恢复以保证虚拟机的正常工作。

底层通用模块层为上层提供了底层功能工具的封装，用于实现系统的基础功能，示例接口如下：

virJSONValueObjectAdd()：向JSON对象中添加键值对，可用于构建JSON格式数据，如QMP命令等。

virMutexLock()：封装pthread库操作，对互斥锁进行加锁操作，确保多个线程在访问共享资源时避免数据竞争，实现线程同步。

virFork()：封装了fork功能，用于创建子进程执行上层指定的任务，子进程pid将返回给调用者用于监控子进程状态。

virExec()：封装了exec功能，用于执行上层指定的命令，其中创建的子进程的pid将返回给调用者用于监控子进程状态。

virCommandNew()：用于根据上层传入的参数创建一个命令对象，命令对象可通过virExec()等进行执行来完成指定的任务。

virURIParse()：用于解析上层传入的URI并保存到一个URI对象，便于对URI进行分析，用于定位虚拟机的类型等。

virDirRead()：用于读取目录条目，上层接口可通过此接口对目录文件进行操作。

# 测试与验证策略

## 单元测试策略

（1）测试框架

单元测试框架采用Google Test (gTest)，该框架是C++社区中最流行的测试框架之一。

gTest框架提供了丰富的断言机制，如 EXPECT\_EQ、ASSERT\_TRUE 等，可以方便地检查各种条件。

gTest框架提供了Mock支持，与 Google Mock 配合使用，可以进行模拟测试（mocking），模拟外部依赖。

gTest框架支持自定义测试过滤器、测试事件和其他扩展功能。

gTest框架跨平台能力好，支持多种平台，包括 Linux和Windows。

（2）单元测试流程

a. 将所研发软件系统划分成清晰的模块：将被测系统的功能划分为多个清晰的模块，例如 Chart 包管理、云应用管理、仓库管理模块等。每个模块都有单独的职责，便于单元测试。

b. 编写单元测试用例：在Google Test中，每个参与测试用例通过TEST宏定义。每个测试用例通过Google Test提供的丰富的断言用来检查被测试的C++函数或模块的返回值是否符合预期。

c. 构建和运行测试：使用CMake编译和构建项目，然后运行测试。

d. 调试和查看测试结果：如果测试用例失败，利用Google Test提供的详细失败信息，包括失败的测试用例、断言失败的具体行号、以及失败的断言类型。根据这些信息，可以定位并修复代码中的问题。使用调试工具（如 gdb 或 LLDB）来逐步执行测试用例，查看失败的根本原因。

## 集成测试策略

（1）确定参与集成测试的软件单元模块

容器云应用管理系统包含的关键软件单元包括用户交互命令工具模块、Chart应用包管理模块、云应用部署管理模块、应用包仓库管理模块、资源定义生成校验模块以及容器云平台交互接口模块。

虚拟机编排调度系统包含的关键软件单元包括管理工具扩展模块、VM资源类型扩展模块、控制器扩展模块、调度器扩展模块、处理器模块、启动器模块和虚拟机统一控制引擎。

（2）确定不同软件模块之间的依赖关系，然后依据依赖关系由下层到上层的顺序逐层组装各个单元模块。



图 34 云应用管理系统软件模块依赖关系及组装顺序

编写测试用例，验证由依赖关系的软件模块能够正确组装和交互。

（3）当所有模块组装到一起之后，通过设计典型的应用流程测试系统中多个模块之间可以正确协作完成系统级功能。

## 系统测试策略

### 功能测试方法

在功能测试用例设计上遵循以下设计原则：

（1）采用了黑盒测试法，只通过软件的输入和输出验证软件的功能实现是否符合js协议的要求。

（2）通过场景式测试用例进行功能测试

a. 以多个类别Chart应用包的创建、打包、发布、下载、安装、更新、回滚和卸载为典型流程，测试容器云应用管理系统的关键功能。

b. 以多副本虚拟机集群的创建、暂停、迁移为典型流程测试虚拟机编排调度系统的关键流程

（3）从用户角度，逐个测试Chart包管理工具、虚拟机编排调度工具virtctl各个命令测试关键功能

（4）测试结果验证上，对于Chart包创建、发布、安装、卸载等主功能项通过观察K3S、容器和后端存储等不同节点的行为进行验证；能力查询、健康状态查询等内部辅助功能通过行为观察、日志分析等方法验证。

（5）分别在天脉容器、天脉Linux环境下进行兼容性测试。

### 性能测试方法

（1）通过脚本连续创建不少于1000个Chart包并循环发布到Chart包管理系统，然后再用脚本循环方式下载1000个Chart包并进行安装，通过行为观察判断是否发布成功。

（2）利用Jemeter等并发测试工具测试Chart包仓库至少可支持50个客户端的并发访问。

（3）通过脚本连续创建20个VM资源和VMI资源验证虚拟机编排调度管理软件可同时支持不少于20个虚拟机实例的管理。

# 项目验收与交付

项目交付计划以下具体如表 5所示。

表 5 交付计划说明

|  |  |
| --- | --- |
| 交付物 | 说明 |
| 技术方案 | 2024.12.10前交付评审稿 |
| 需求文档 | 2025.01.10前交付初稿  2025.02.10前评审通过 |
| 设计文档 | 2025.02.05前交付初稿  2025.02.10前评审通过 |
| 源码 | 2025.05.10前完成云应用快速部署软件0.1版  2025.07.10前完成云应用快速部署软件0.2版  2025.09.10前完成云应用快速部署软件定版 |
| 测试文档 | 2025.06.10前完成软件测试大纲  2025.08.10前完成软件测试报告 |
| 验收文档：  软件用户手册  软件研制总结报告  软件溯源分析报告  软件知识产权说明 | 2025.09.10前完成 |