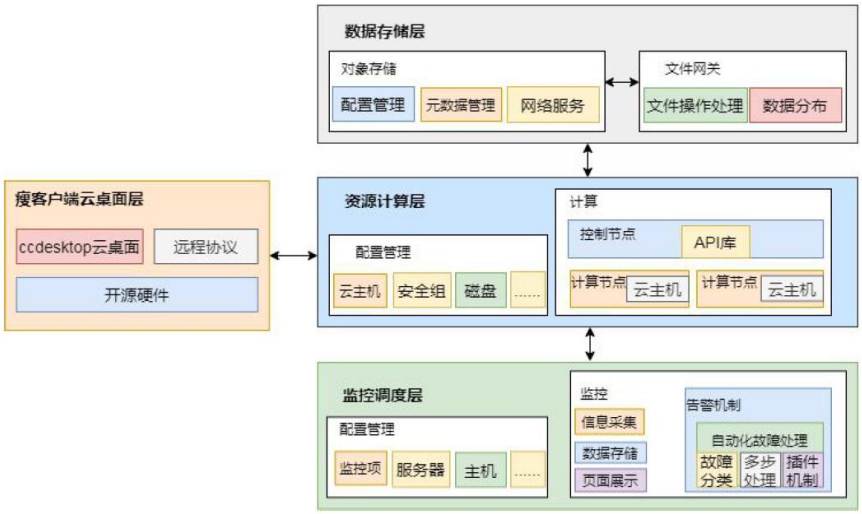
基于瘦客户端+Openstack+zabbix+grafana 的超融合智能可扩展的云桌面平台

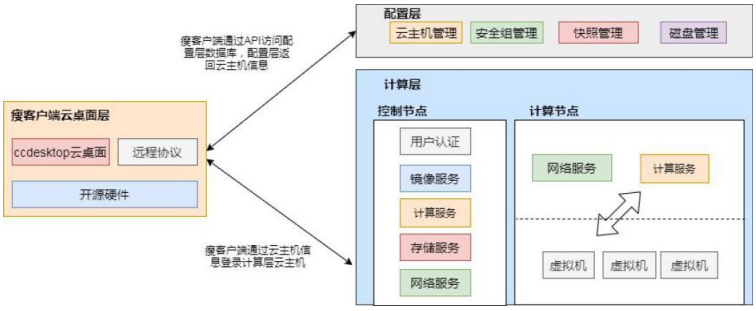
设计流程：

智能可扩展的云桌面平台分为客户端和后端两部分，后端又分为资源计算层、数据存储层、监控调度层。整体架构:



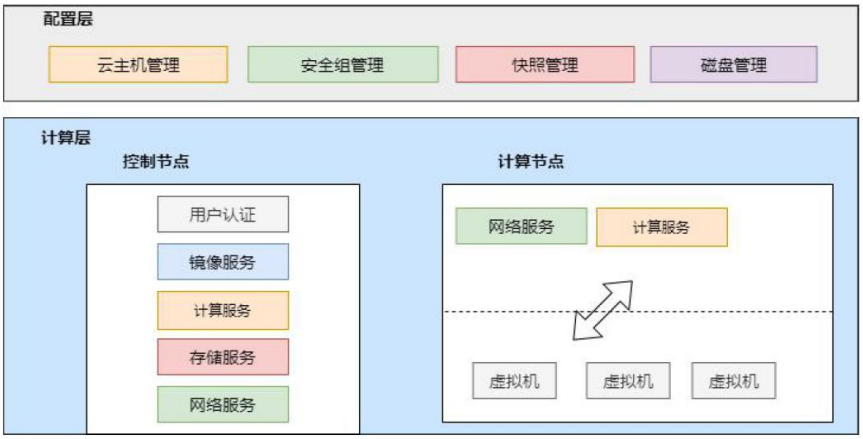
客户端：

瘦客户端与 CDC 云桌面系统是实验室自主研发的便携式主机访问系统，基于瘦客户端，CDC 云桌面系统实现开机自启，用户访问 CCCloud 配置层，登录成功后，获取主机信息，选择自己要连接的主机，CDC 云桌面会判断主机类选择相应的高效远程协议连接 CCCloud 计算节点的云主机。瘦客户端通过开源硬件实现，支持 RCAav 端口音频输出、4 个 USB 端口、HDMI 接口、POE 供电、RAM 1GB 64 位 4 核处理器、以太网端口、无线局域网蓝牙、WIFI 等。CDC 云桌面基于 Linux 系统实现开机自启，要求用户输入 CCCloud 配置层节点地址，通过用户名密码登录，获取数据库主机信息，然后选择 CCCloud 计算层创建的主机，CDC 通过选择高效的远程协议进行远程连接。



后端：

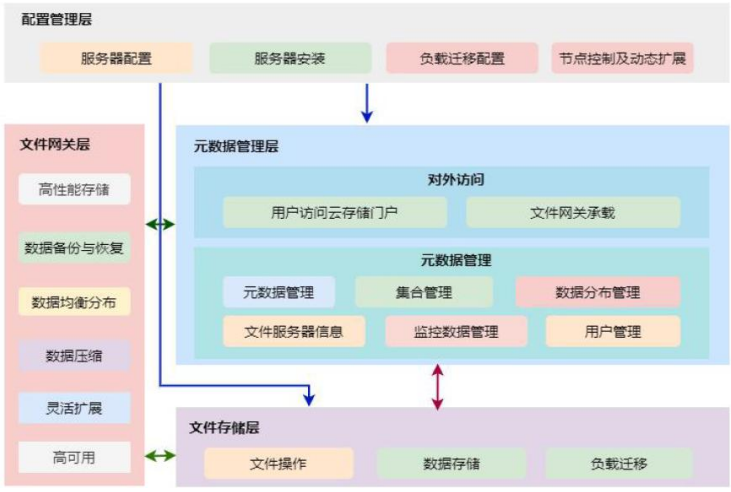
CCCloud:是实验室自主研发的云计算资源管理系统，主要实现了物理机的虚拟化，管理员可以根据用户的需求定制出虚拟机。该部分的整体架构由配置层和计算层两部分组成。云管理平台在底层虚拟出一个巨大的虚拟资源池，资源可以根据负载规模的变化动态配置虚拟主机，最优的利用资源，提高利用率，降低成本，实现主机硬件的可扩展性。设置浮动 IP，与主机 IP 与其绑定可以访问外部网络，主机关机时动态回收浮动 IP 地址，用于其他主机绑定，这样可以减少 IP 冲突，提高网络性能，实现网络的可扩展。



配置层：主要负整个集群资源的调配，以 web 界面的形式提供给管理员。管理员可以通过界面进行当期已分配资源的查看、增加、修改和删除。

计算层：针对配置层所提交的需求进行底层实现。后端资源计算层以开源的云管理平台 Openstack 为原型进行二次开发。其主要模块有云主机管理、安全组配置、快照管理、磁盘管理和用户管理。云主机管理主要负责云主机的创建、分配和状态管理。安全组配置主要负责安全组的增删改查。快照管理主要负责对当前状态的云主机进行拍摄拍照，以及快照的增删改查。磁盘管理主要负责对已经创建分配出去的云主机进行添加和删除云硬盘。用户管理主要负责对需要使用云主机的人员进行注册以及权限的变更。计算层冗余设计，采用集群系统，将各个主机系统通过网络或其他手段有机地组成一个群体，共同对外提供服务。创建群集系统，通过实现高可用性的软件将冗余的高可用性的硬件组件和软件组件组合起来，消除单点故障。

CSCloud:该系统是基于对象存储自主研发的云存储系统。存储系统由配置管理层、元数据管理层、文件存储层以及文件网关层结构组成。

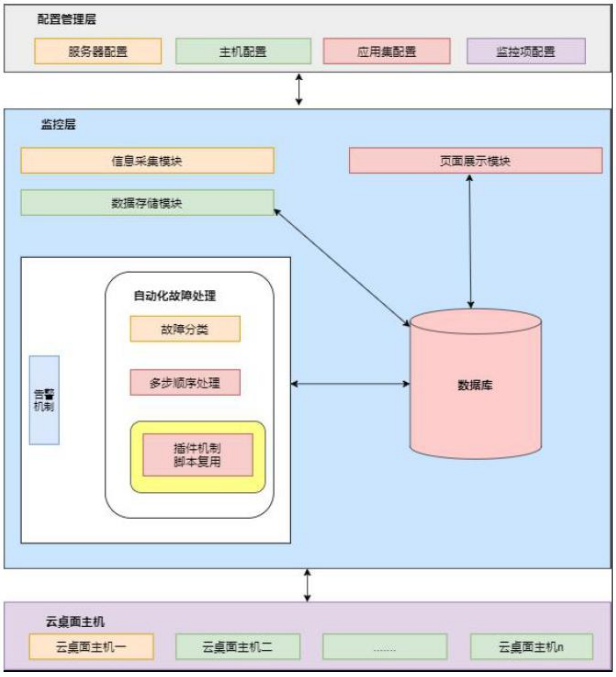


配置管理层（CS）：主要包括服务器配置、服务器安装、节点控制、负载迁移控制。它控制元数据管理层（MS）:元数据管理层主要负责整个系统中所有存储节点的基础信息，包括磁盘容量、cpu、内存等。还包括用户上传文件和存储节点间的对应关系、文件的主副存储服务器关系以及每个用户的已使用和剩余磁盘容量信息。且提供给管理员和用户一个 Web 界面进行管理以及数据访问。此外，还承接文件网关层与文件存储层的业务实现。

文件存储层（DS）:文件存储层主要负责具体数据的存储。同时，提供文件网关文件处理 API 以及负载迁移的底层实现。

文件网关层（FSG）:文件网关层是所有前端系统用户数据存储的入口。他的主要负责瘦客户端系统镜像文件的备份与恢复、高性能访问、均衡分布、数据压缩以及灵活的文件存储需求扩展。

CMCloud: 该部分是基于自动化故障处理的监控系统，整体架构分三大部分组成：云桌面主机、监控层、配置管理层，从而可以实现对被监控对象的信息采集、故障定位、故障分类、故障诊断以及故障修复的一个闭环自动化恢复过程。



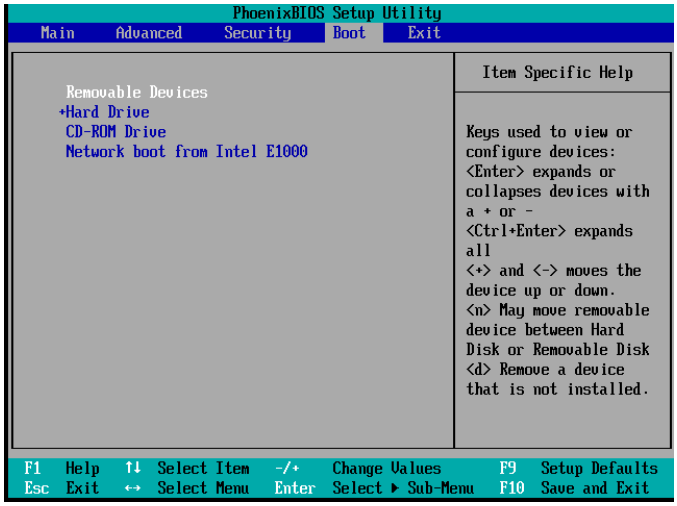
云桌面主机：指别监控端（Agent）的集合，而监控端（Server）是通过主动轮询和被动捕获这两种模式对别监控端进行主动和被动的监控。
配置管理层：负责给用户和后端提供友好交互，主要是为了方便用户对被监控的信息进行设置，包括对被监控的主机（Host）,该主机对应的监控项（Item）,该监控项所对应阈值（Trigger）以及对应的动作（Action）等四部分的 CRUD(增删改查)
监控层：基于自动化故障处理的监控系统 zabbix 的底层实现。该部分分为信息采集模块、数据存储模块、故障处理模块以及页面可视化展示模块。首先是信息采集模块，该模块是在监控端获取到被监控对象的各种信息后，将采集到的各种数据在数据存储模块进行实时存储；页面可视化展示模块主要负责用户和监控服务器进友好交互，其作用是方便用户在监控端对被监控对象管理配置，除此之外，可以借助 Grafana 插件把监控端采集的相关数据进行可视化展示；自动化故障处理模块主要负责对监控系统的被监控端所产生的故障进行处理，以及对故障恢复程序的管理。故障处理指故障分类、故障报警、插件机制。故障分类是指当出现新的故障时，管理人员会受到报警信息，同时平台会首先查询智能库，判断智能库中是否存在解决故障的插件，如果存在解决方案，可下载尝试处理，实现处理脚本的复用。如果不存在该插件，该系统将调用大模型的 API，把报警信息传递给大模型，根据得到的处理解决方案依次尝试，如果故障依然无法解决，用户可以自定义解决方案，成功解决的方案会被智能库收录；

实现方案：

1. 瘦客户端云桌面系统实现：

瘦客户端是一种体积小、低功耗接入终端、为用户提供输入、输出接口，网络通讯等功能。基于以上要求，我们选择开源硬件进行二次开发作为终端。它的系统可以支持嵌入式系统，如Linux，已经没有可以去除的其他软件了，但它具有电脑的所有基本功能，只需接通显示屏和键盘，远程连接到虚拟机，就可以应用传统电脑的全部功能。ccdesktop 云桌面通过shell 脚本启动，启动后根据用户输入的CCCloud 配置层地址，判断云管理平台是否存在，然后通过Restfull API 进行信息交互。通过用户名密码登录CCCloud 配置层，动态的获取配置层数据库中虚拟主机的IP 地址、操作系统、Mac 地址等信息，通过IP 地址、操作系统信息，选择合适协议进行远程连接。RDP协议用于windows->windows,linux->windows连接。VNC:适用于windows->linux和linux->linux之间。SPICE主要用于linux->linux虚拟机的连接。远程桌面协议是瘦客户端云桌面的关键技术，它可以传输图像以及键盘鼠标的指令。所有的计算由瘦客户端将数据通过远程协议传送到服务器端完成，服务器端更新图像信息传输到瘦客户端，降低本地客户端额配置。通过研究几种主流远程协议，选择合适的远程桌面协议，协议要满足Linux、Windows虚拟桌面的音频、视频、usb 传输等功能，并且支持多个用户同时登录。Windows 系统已选择满足我们云桌面平台的远程协议，Linux 系统最终通过开源远程连接协议，进行二次开发，满足系统的Linux 远程桌面音频视频等功能。

无盘系统和瘦客户端系统的所有配置功能是在一台机器上实现的，最后烧录到嵌入式单片机上。整个过程通过使用交换机，确保了所有机器在同一个局域网中。单片机开机运行，处在该局域网中的其他机器通过 pxe 网络启动选项（PXE 是一个标准协议，用于在没有本地存储设备的情况下通过网络启动计算机）可以发现在该局域网中的单片机机器，通过 PXE 选项进入到该机器内部，通过与脚本文件交互，可以获取到管理员后台为指定用户通过 openstack 创建的虚拟机器列表。用户可以选则需要连接的虚拟机类型和地址，然后通过 spice 协议可以完成对创建的虚拟机连接。该单片机器主要实现了无盘启动和瘦客户端系统功能，无盘启动的功能体现在，可以通过相应的配置，让处在同一网络中机器通过 pxe 启动来发现该单片机，瘦客户端系统功能体现在，编写了一系列 http 请求脚本，可以获取到管理台通过 openstack 为用户创建的虚拟机等。创建虚拟机的过程是通过自己开发的页面，调用 openstack 后台的功能实现的。



2. CCCloud 云计算资源管理系统实现

CCCloud 云计算资源管理系统模块与模块之间松耦合设计，添加其它组件可以使用其

规范的API。CCCloud 云计算资源管理系统的API 均为Restfull API 风格，可以灵活的进行使用。配合其优秀的框架可以便捷稳定的进行开发，降低了节点扩展的复杂度，确保了在计算资源方面进行节点可扩展的可持续互操作性。实现了源代码层面的资源调度接口。为系统服务器节点和扩展流程提供了云服务。为了方便管理员更好的进行资源调度以及避免与开源云管理平台在整体界面操作方面的复杂性，选取轻量级的Flask 框架进行配置层开发。在配置层通过对管理员透明的方式来进行云主机管理、快照管理、安全组管理等调度资源操作。

配置层：选择轻量级的插件式框架Python Flask，可以灵活地选择自己需要的插件。选取Flak_aqlachemy 来保持数据库选择的灵活性，使用Extension 机制来支持自定义的RESTful 风格的云资源计算API。Flask 框架遵循WSGI 协议，WSGI（Web Server Gateway Interface）协议是 Python 编程语言与 web 服务器之间的一种接口标准，它定义了 Python web 应用程序与 web 服务器之间的通信方式。WSGI 主要用于 Python 的 web 框架和 web 服务器之间的交互。使用Werkzeug 工具（Werkzeug 是一个用于构建 Python Web 应用程序的库，它提供了一个灵活和强大的工具集，可以帮助开发者更容易地开发 Web 应用）规范了控制层的HTTP 请求，通过Jinjia2（Jinja2 是一个现代的、功能强大的 Python 模板引擎，广泛用于将动态内容嵌入到 HTML 文件中。它常被用于 Web 开发，尤其是在框架如 Flask 中，用于处理 HTML 模板渲染） 使得模板引擎支持路由解析，在处理业务逻辑层的请求响应基于集成沙箱环境。这些特性在资源调度层极为重要，可以满足系统中灵活多变的管理请求。为了实际环境稳定。特选取Nginx、Flask、WSGI 三者组合的方式进行开发。一方面在API 库的调用上灵活可靠，另一方面保证了配置层和计算层的一个低耦合。使其在资源的调度上起到了一个流程的转化，实时对计算层进行云同步，防止了一些管理上数据不一致的可能。更大程度上使得资源调度系统的稳定性、操作性和维护性得到了极大提升。计算层：服务器包括控制节点和计算节点。本文在此层主要工作是在控制节点上建立API 库。API 库主要分为：认证模块库、创建模块库、查询模块库和配置模块库。这些库中封装好了所有与资源有关的脚本，资源计算层的前台可以随时调用这些库进行操作。在监控调度层亦可调用此库进行自主的节点扩展。认证模块库主要进行资源计算系的认证，包括管理员的账号密码、项目ID、网络ID、安全组ID 等信息。创建模块库主要进行节点的创建、网络创建、安全组创建、云磁盘创建、快照创建。查询模块库主要进行系统所有资源的查询反馈。配置层主要针对已经调度好的资源进行按需调整。

管理员访问配置层以统一资源定位符URL 登录并进行操作。首先访问资源调度层宿主机的URL，宿主机通过预设端口转发至容器内连接上Nginx 服务器Web 端进行资源操作，之后配置层将请求以标准通信协议提交给计算层，然后对提交的需求进行API 库选择，挑选脚本处理请求。

3. CSCloud 云存储系统实现

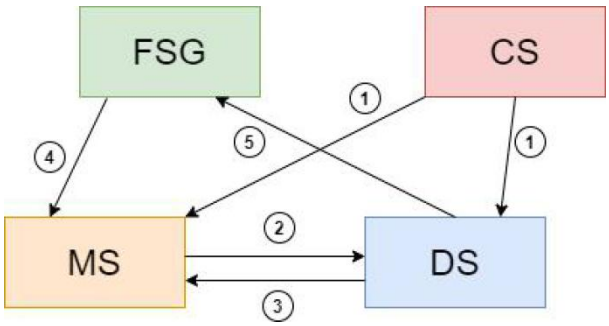
配置管理层的业务逻辑由PHP 结合关系型数据库实现。通过该实现方式管理员可以Web 方式方便的实现对整个存储系统节点配置、安装及负载迁移等。

元数据管理层的业务逻辑由PHP、关系型数据库以及参考流行的API 封装方式实现。通过该实现方式，为用户提供了一个以Web 方式进行注册及访问存储的门户，实现了元数据的管理、承接文件网关层及文件存储层的业务处理。

文件存储层的业务逻辑由PHP 结合流行的API 封装方式实现。通过该方式，将上传、删除、下载、修改进行封装，同时支持了元数据管理层和文件网关层的业务请求，实现方式的文件处理请求。

文件网关底层实现原理是基于一个开源的文件系统二次开发。该文件系统的优势是使

用灵活，能够针对不同的文件操作需求进行文件系统功能上的修改。另外，该文件系统读写可进行控制，便于在数据处理时做一些策略。参考该开源文件系统框架实现方式，对其中关键部分做了功能扩展。如果将FSG 上的文件称为元数据文件，存放于远程DS的文件称为源文件，在FSG 上对元数据文件进行文件操作后，文件操作会转化为一系列的该文件系统底层调用，通过对文件系统底层功能扩展，实现在FSG 上进行文件操作就像操作DS 上文件一样。在一定意义上，若将存储系统中DS 集合抽象为一个存储设备，那么FSG 就是把该文件系统挂载到了这个大容量的存储设备上。CS、MS、DS、FSG 四层服务器之间关系如图。



①表示CS 对所有的MS 和DS 进行管理和配置；②表示MS 对DS 进行管理，存储其相关信息并且转发文件网关层的存储请求。③表示当DS 完成存储后将相关信息发送给MS。④表示FSG 向MS 发送文件请求。⑤表示文件请求数据返回给FSG。

当存储系统容量不够时，可通过CS进行MS节点以及DS节点的动态配置与安装，保证存储系统容量的动态扩展。将FSG 高性能存储设备上的数据进行部分压缩放置远程存储量大但存储性能较低的文件文件存储服务器上，实现了数据分层存储。当用户使用时，动态的从远程提取被压缩部分数据。这样在不损耗数据访问性能的同时节约高性能存储设备容量，降低成本。当存储系统需要引入更加先进数据存储方案时，只需要在文件网关上添加该方案，就可以实现，不需要修改整存储系统逻辑。整个存储系统每层都可单独部署成一个集群，通过该部署方式，避免单点故障，实现整个存储系统的可用性，保证其业务连续最大化。另外，该部署方式也为负载均衡实现提供条件，提高整个存储系统的服务性能。

4. CMCloud 云监控系统实现

CMC 云监控平台用户通过配置管理层对主机，应用集，监控项，动作进行过增删改查，监控层对各种数据进行采集，当发生故障向前端、用户报警。同时平台会首先查询智能库，判断智能库中是否存在解决故障的插件，如果存在解决方案，可下载尝试处理，实现处理脚本的复用，并将处理结果以邮件等形式通知运维人员。如果不存在该插件，该系统将调用大模型的 API，把报警信息传递给大模型，根据得到的处理解决方案依次尝试不同方法，如果故障依然无法解决，用户可以自定义解决方案，成功解决的方案会被智能库收录；CMC 云监控平台采用前后端分离开发的模式进行编写，前端页面的编写采用 Vue 框架以及 element-ui。后端采用了 Django 框架，是一个开放源代码的 Web 应用框架，由 Python 写成。采用了 MTV 的框架模式，即模型 M，视图 V 和模版 T。数据库方面我们采用了关系型数据库 mysql，文件管理系统我们采用了分布式文件管理系统。数据库中保存了主机，触发器，用户等信息，而分布式文件管理系统中则保存用户公共分享以及私密分享的插件（用于解决各种故障）。