Nano 云平台技术白皮书

前言

Nano的诞生，起源于公司内部为了搭建大数据平台开发环境而进行的云计算平台选型。公司有一堆x86服务器，只需要创建出实例，网络连通就行，但是没想到这个简单的要求，却没那么容易搞定。

我们调研了很多现有的云计算和虚拟化平台，发现许多知名产品内部组件繁多，相互之间重重关联，产品体系庞大，概念名词也很多，导致学习困难，配置也极为复杂，难以维护。

另一方面，很多产品还依赖各种第三方软件和库，包括数据库、消息队列等等，版本稍有不对就会产生兼容性问题；搭建一个云平台，需要搭建3、4个额外的依赖软件，不止是引入更多故障点，降低系统稳定性，增加了问题排查难度，而且还暴露了更多可以攻击的安全隐患。

除此之外，Java/Python/Bash等多语言和脚本混杂工作，不仅效率低下，而且一个简单任务牵涉太多代码，对运维开发人员来说也难以学习和维护。

总的来说，现有平台和产品，对于只想使用云主机和虚拟化，但是缺乏足够资源进行研究和维护的团队来说，实在太过笨重，于是，才有了超轻量级的开源云平台Nano。

Nano基于CentOS/KVM虚拟化技术，使用服务器集群构建计算资源池并提供云主机实例管理服务。

Nano最大可能采用智能化和自动化手段替代繁琐易出错的手工操作，在简单易用的基础上，提供强大而稳定的云管理平台，产品，在解放运维人员的同时，提高集群的资源利用率、可用性和可靠性。

对于支持Intel VT-d或者AMD-v的普通x86服务器，管理员只需要下载并安装Nano部署包，即可在三分钟内将其转换为云管理平台，并且开始创建云主机实例。

Nano使用MIT许可，无论自用、修改或者商用都无限制

欢迎访问官方网站 https://nanos.cloud/ 或者加入官方QQ群 Nano Cloud(**819161083**)了解更多内容

作者: Akumas (bokuore@github.com)

Git库: https://github.com/project-nano

版本获取地址： https://github.com/project-nano/releases/releases

手册地址: https://nanocloud.readthedocs.io/projects/guide/zh_CN/latest/

特性

Nano有以下特性：

- **紧凑**：所有后台部分使用Golang单一语言开发，全项目仅3万多行代码，不到Openstack的三百分之一，学习研究起来非常简单。平台部署仅需要编译三个模块二进制执行文件即可，最大的模块也只有9MB大小，无需第三方软件或者依赖库即可执行，更不用担心版本兼容问题。模块升级时直接替换即可，无需考虑复杂的外部依赖，极大降低了维护难度和工作量。
- **开箱即用**：Nano自带完整的Web管理门户以及云主机监控页面，无需安装额外组件。整个系统以最小化配置为目标，从网络发现、组网到设备选择，尽量使用智能化自动配置，将需要人工配置和选择的可能降到最小。
- **可靠**：Nano使用All or Nothing的事务模式管理所有处理任务，即要么任务彻底成功，要么失败回撤，释放所有资源并且还原状态，彻底避免出现资源泄露或者状态不一致。同时Nano没有使用缓慢而且易出错的数据库存储状态，集群实时检测所有节点状态，同时在内存中自动同步实例数据，确保所有状态和操作及时且有效。
- **易扩展**：Nano所有功能都由REST API接口调用，同时实现了应用与资源服务的分离，用户可以便捷地集成到现有系统，或者直接开发自己应用。基于goroutine的业务逻辑抽象，可以让用户快速扩展业务功能，而无需了解复杂的后台消息驱动机制。

差异

Nano作为一个从零开始设计的全新平台，与市面上现有产品相比，有不少的差异。

开发语言

市场主流产品使用Python或者Java开发，资源消耗大，执行效率低，而且执行时需要配置各种运行环境和依赖库，稍有不慎就会产生运行错误或者兼容性问题，这对于需要长时间运行和维护的生产级平台是一个很大的隐患。有的产品甚至还会使用Java/Python/Bash等多语言和脚本混合协同工作，不仅效率低下，逻辑复杂，学习痛苦，对于问题定位和排查也带来极大的困难。

Nano采用谷歌专为后台服务全新设计的语言Go作为唯一的开发语言，依靠Go丰富的功能类库，无需借助任何外部脚本即可实现所有管理功能；利用Goroutine特性，可以在兼顾业务开发效率和并发处理速度的同时，最大程度利用服务器性能，降低资源消耗。最重要的一点，Go编译生成二进制可执行文件，不仅尺寸小，执行效率高，而且无需依赖任何运行环境和运行库即可执行，彻底杜绝传统产品依赖关联和版本兼容的问题，不仅部署和升级更加简单，基础运行环境也可以根据用户需要任意升级和调整，而无需担心干扰Nano模块的正常工作。

自动组网

传统产品搭建网络集群时，涉及服务端、客户端、数据库、消息队列等多种网络组件和服务，每个集群节点少则几个，多则几十个端口，需要在每个节点上选择合适的服务端口，同时还要在所有关联的节点上正确配置好相应的访问地址。当集群节点扩大时，节点之间相互依赖的配置数量成几何倍数增长，不仅工作量巨大，而且稍有不慎就会导致通讯失联或者服务故障。再加上生产系统日常维护产生的设备迁移和地址变更，网络关联关系的管理维护对任何运维人员都是无法回避的困难挑战。

Nano基于组播协议，集群模块可以自动完成工作网络发现、通讯地址选择和关联通讯链路建立等一系列工作，无需任何人工干预；当设备迁移和地址变更时，也能自动更新并且关联应用模块也能自动完成同步。这种模式彻底杜绝人工配置低效而且易出错的问题，在灵活多变的网络环境下，也能保障服务集群的可靠和稳定。

无数据库设计

不少主流产品使用MySQL等传统关系型数据库保存系统配置、采集和检测系统状态，但是数据库系统本身基于磁盘读写，响应速度很慢，不仅自身成为一个性能瓶颈，还常常因为软件故障或者程序bug，导致写入错误数据，引发状态异常或者分配错误。由于这种故障非常隐蔽，只有检查后端数据才能发现，只有非常熟悉系统工作原理、表结构设计以及数据库的管理员，才能进行定位和修复，而一旦发生此类故障，对于普通用户来说几乎无解。

除此之外，数据库系统本身就是一个复杂的专业系统，涉及日常维护、数据备份、安全防护、性能调优等多个方面的专业领域知识，不仅在系统中引入了额外的组件，增加了复杂度和学习难度，降低了系统稳定性，还引入了更多的安全隐患导致系统更易被攻击。事实上，有很多平台就是因为数据库默认用户被破解或者未及时修复数据库漏洞而被恶意用户攻破。

这一点上，Nano放弃了传统的关系型数据库存储信息，配置信息使用简单的JSON文件格式存储，状态数据全部实时采集，实时同步，实时处理，而且常驻内存作为热数据处理。系统内的所有状态信息均持续实时更新，即使出现特殊状况，只需要重启模块，就可以自动重新同步所有状态，彻底杜绝传统产品状态和数据不一致的问题，让系统更加健壮和易用，而且纯文件存储，使得系统备份和恢复更加简单。

异步消息驱动

现有大部分产品，主要使用RPC（Remote Procedure Call）协议由主控端调用被控端服务，比如控制远端的资源节点分配虚拟机实例。RPC在网络交互基础上，封装函数接口，可以在本地直接调用远程服务处理任务。但是RPC为了便于使用，通常采用同步执行方式，即发出了请求，必须等待收到响应或者超时，才能继续处理后续指令。当网络或者服务端出现异常的时候，这种模式常常会导致后续指令时间阻塞，所以并不太适宜大量并发任务处理。

Nano的跨节点控制和通讯都使用高性能分布式系统常用的异步消息机制，同时配合Nano自己的消息管理与事务处理机制，在不增加开发复杂度的情况下，能够支持数十万任务请求同时高效地并行处理。

事务处理

计算机系统并不是永不失误的完美系统，相反，当集群达到一定规模之后，磁盘损坏，网络故障几乎就是每天都会遇到的日常现象。当出现这种异常情况的时候，现有大部分产品几乎都缺乏有效机制进行处理，只能简单终止任务执行，但是对于已经申请的资源，既无人管理也不会释放，已经写入的状态数据也不会还原，最终导致资源泄露和状态不一致，只能依靠复杂的手工恢复。

Nano所有处理任务都使用All or Nothing的事务模式执行，要么任务彻底成功，要么失败回撤，释放所有已分配资源并且还原状态，彻底避免出现资源泄露或者状态不一致。Nano的事务机制同样支持高并发处理，每个任务都有自己独立的会话状态管理，可以高效地分别进行提交或者回滚，相互之间没有干扰，确保整个系统的长期稳定和可靠运行。

网络模型

现有许多产品为了能够最大程度满足用户虚拟网络管理方面的需求，引入了虚拟的二层网络、三层网络、虚拟路由（Virtual Router）等诸多概念，还增加了SDN、NFV等技术框架，虽然确实提升了系统的灵活性，但是极大地增加了系统的复杂度和学习难度。引入的众多子系统和框架，不仅让管理员配置可用的虚拟网络系统更加困难，也让网络问题的定位排查变得极为复杂，验证降低了系统的可靠性和稳定性。

Nano简化了实例使用的网络模型，默认配置下，云主机实例直接通过宿主机桥接接入物理网络，实例与物理服务器一样，通过物理路由器获取地址和网络配置，然后通过物理交换机进行二层报文收发。云主机实例在物理网络中可见，管理员可以通过原有物理路由器、交换机进行管理和配置，无需再学习新知识，也无需进行新的设备采购和配置，即可快速上手进行管理。

全功能接口

现有各主流产品，因为种种历史原因，有的没有提供任何API接口，有的虽然提供接口，但是散布在多个不同系统组件里，调用方式不同，工作流程不同，甚至连开发语言都不同。要在此基础上进行二次开发或者系统集成，常常需要整合多个组件的不同接口，还要熟悉各子系统工作原理，甚至要利用到未公开的内部接口，才可能顺利完成。

Nano从一开始就为了二次开发与系统集成而设计，所有平台功能和服务均由统一且唯一的RESTful API接口提供，包括系统自带的Web管理门户也同样调用该接口实现。简单易用的REST风格，可以由各主流开发语言和框架进行调用，非常易于进行二次开发与系统集成。覆盖所有功能的完整REST接口，也确保了用户通过API即可发挥Nano的全部能力，而无需再去寻找或者扩展额外的内部接口。

资源与业务剥离

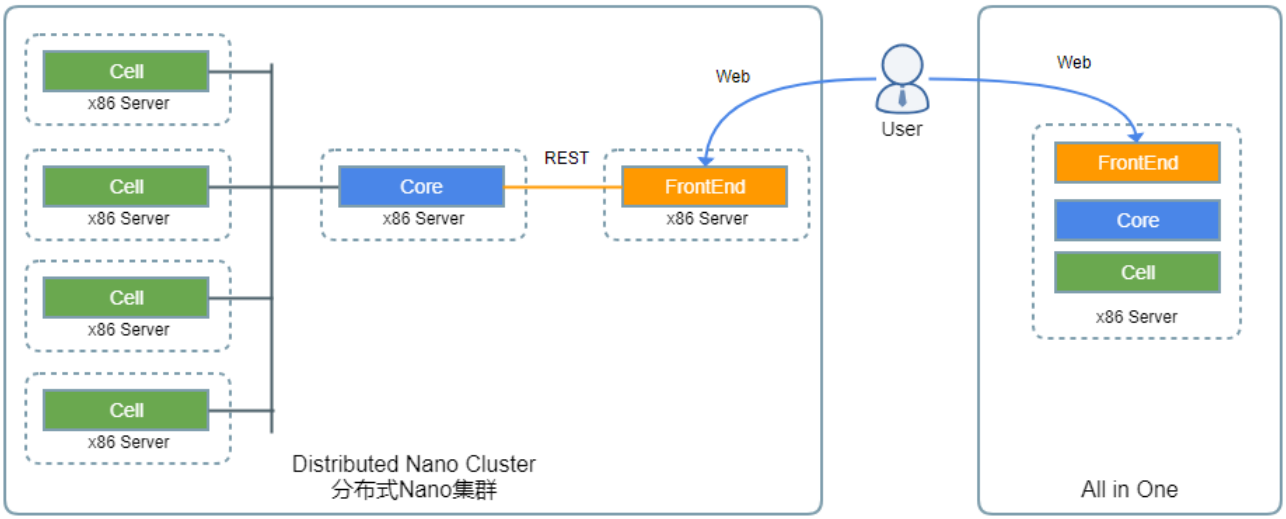
不同的客户有不同的应用场景，不同的需求衍生出不同的管理策略和业务逻辑设计，各种客户的需求差别之大，导致任何一个产品都不可能满足所有用户的需求场景。Nano的架构设计中，通过FrontEnd与Core模块的分离，将系统分隔为管理业务逻辑的前端与提供资源服务的后端资源集群，前端根据不同的客户场景实现各自的业务逻辑与资源分配策略，后端资源集群根据前端业务请求提供实例调度、镜像管理等资源服务，而不关注业务逻辑。这种架构，保障用户不担心影响后端资源管理调度的情况下，灵活根据自己业务需求实现自己的管理前端，实现业务与资源的分离。

设计要点

本章节描述Nano架构中的设计要点

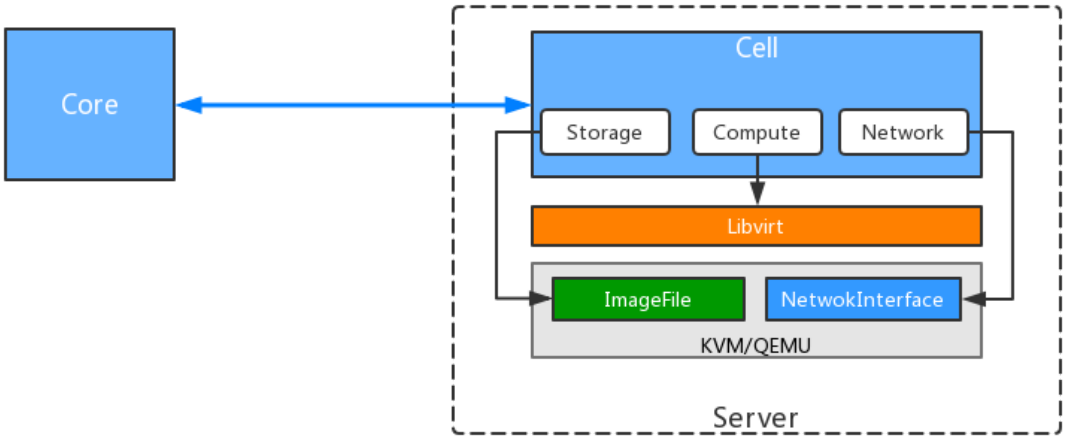
基本工作原理

Nano将一台或者多台x86服务器构建成虚拟资源池，当用户通过Web门户或者REST接口发出创建云主机请求时，Core模块根据资源池内各节点的负载情况，选择合适的承载节点，通过消息控制该承载节点所安装的Cell模块进行实例创建，并负责后续的管理调度以及资源回收等工作。



系统中Core是最核心的主控模块，它负责集群的建立与管理、资源池状态监控、云主机与镜像资源的分配调度、REST接口服务以及任务请求的处理与分发。启动集群时，Core模块需要最先启动，其他模块才能正常进入服务状态。

Cell模块运行在每个可以部署云主机实例的服务器节点上，Cell模块基于KVM与Libvirt工作，启动后持续采集节点资源状况，并实时同步到Core模块，另一方面，收到Core发来的实例创建指令时，分配相应的网络、存储和计算资源，并组装成云主机实例供用户使用。



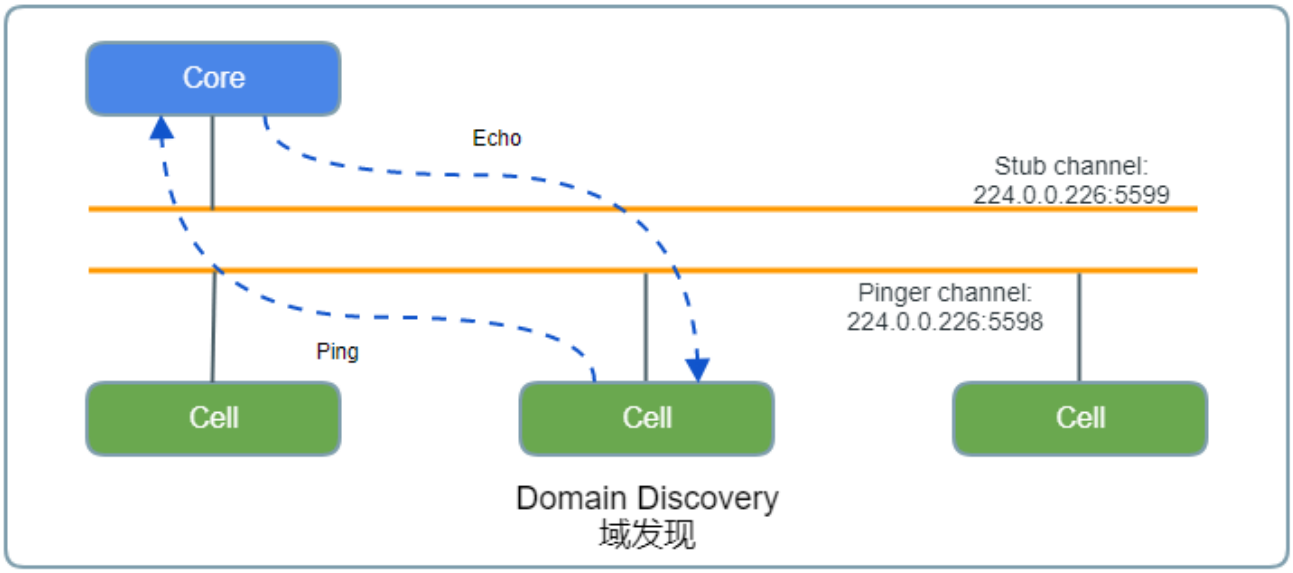
FrontEnd模块是基于Core模块API接口的Web管理门户，它提供了资源池、云主机和镜像等资源的管理调度，也实现了基本的用户管理、安全校验、日志审计等功能，用户可以在FrontEnd上实现对整个平台集群的管理，也可以根据自己业务需求实现一套完全不同的Web应用。

通讯与组网

Nano集群节点之间使用基于UDP的可靠传输协议进行模块之间的通讯，目前版本使用JSON格式进行消息序列化与反序列化，每个消息报文除了收发者信息之外，还是用Key-Value方式传递消息参数。这种模式可以使用一个类统一进行消息解析和处理，而且能够任意扩展携带参数，同时保持后向兼容。

网络发现与组网

每个Nano集群节点至少需要使用一个UDP端口进行通讯，同时现实环境里一个服务器节点通常有多个网络设备，必须找到正确的地址和端口才能顺利建立连接。为了减少不必要而且容易出错的手工配置，Nano模块启动时，会首先利用组播协议，确定能够与Nano集群通讯的网络地址，然后在本地的工作端口范围内，找出尚未占用的监听端口，启动通讯服务。



整个过程不需要用户配置地址或者端口，只需要首先启动Core模块，作为组播监听和响应端，然后确保各模块使用了相同的组播域参数即可。

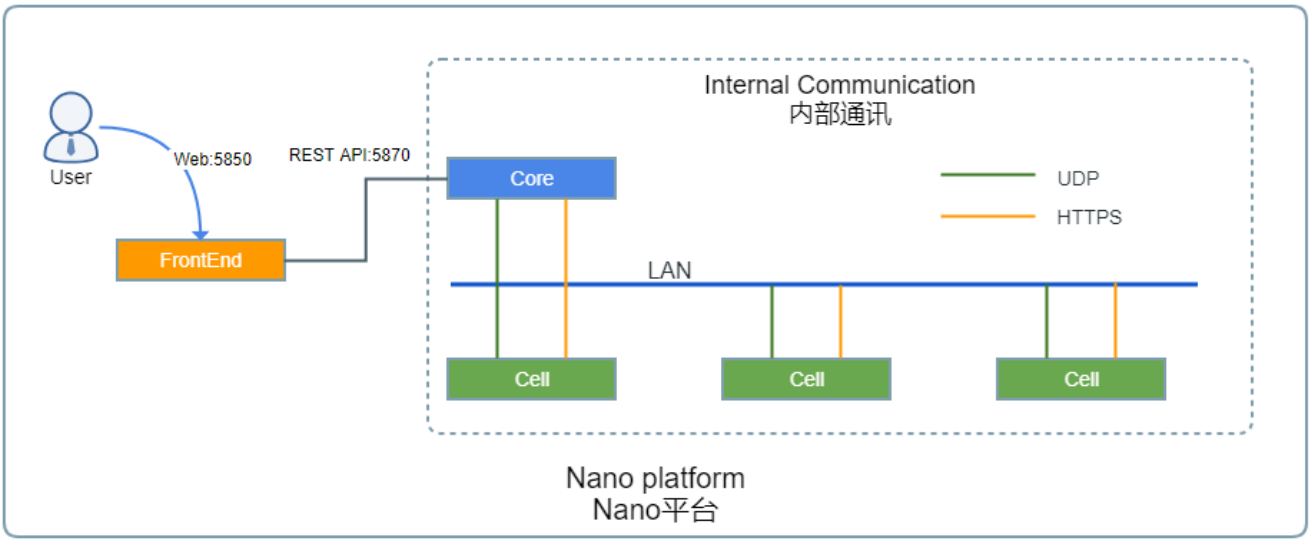
模块通讯

Nano中，通讯分为外部通讯和内部通讯两种。

外部通讯是Nano集群与外部模块之间的通讯，通常使用固定地址和端口，以便外部访问，比如Core模块提供的REST API服务默认使用TCP 5870，FrontEnd提供的Web门户默认端口为TCP 5850。

Nano集群内部节点之间属于内部通讯，Nano的内部通讯组网结构类似P2P，每个节点都会自动生成一个集群内部唯一的标识名称，集群会持续监测节点加入或者离开。节点加入时，会自动与关联节点建立连接，离开时则自动释放已有连接。

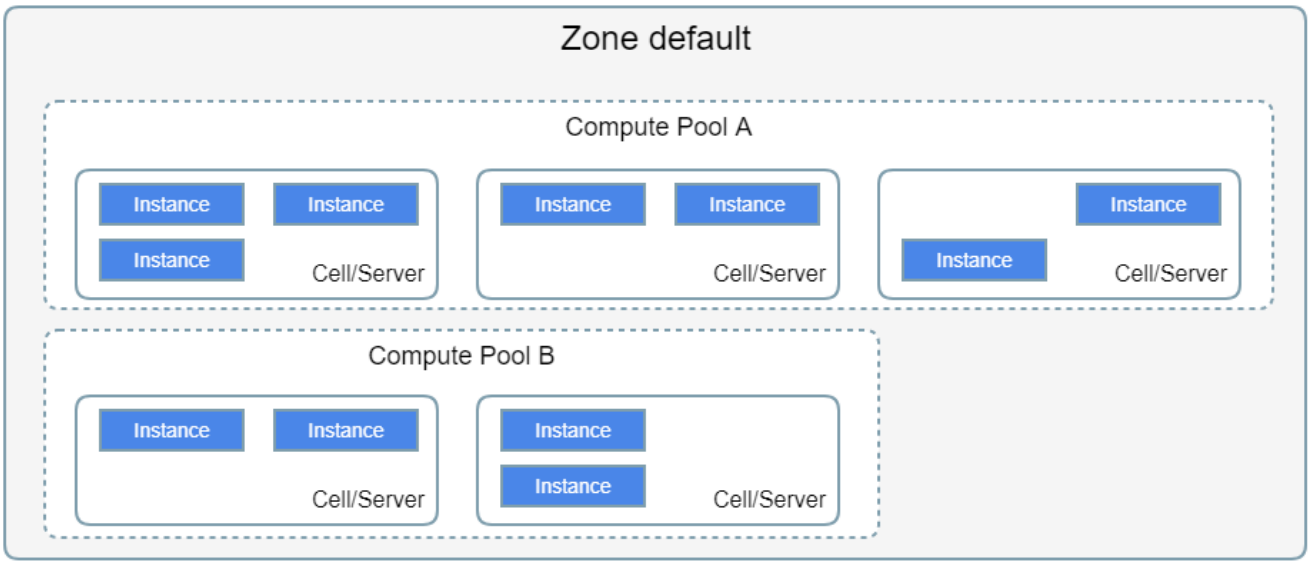
为了适应多变的网络环境，集群节点使用地址与端口均由系统自动选择，无需人工配置即可正常工作；当发生地址变化或者服务器迁移等情况下，节点自动会最新的网络配置同步到集群中并且恢复原有的关联关系，此后的任务请求都将发送到新的服务地址上处理，完成服务切换。



集群内部的HTTPS流用于传输加密的镜像文件数据，UDP则用于传输控制消息。UDP控制消息通常包含发送模块与接收模块名称，模块发送消息后，由Nano的通讯协议栈确保消息达到目的节点。部分模块，比如Core，还包含多个内部逻辑子模块，所以UDP消息除了节点之间的通讯，也可以用于内部子模块之间进行高效地消息传递。

资源模型

资源池



Nano以计算资源池作为核心调度管理单元，一个资源池通常由一个或者多个安装了Cell节点的服务器组成，Cell持续采集本节点负载状况并且持续上报给Core。当Core收到创建云主机请求时，根据实例所需资源大小以及资源池内各节点的负载情况，选择负载较轻的节点进行云主机部署，从而分担系统压力，延长集群使用寿命。

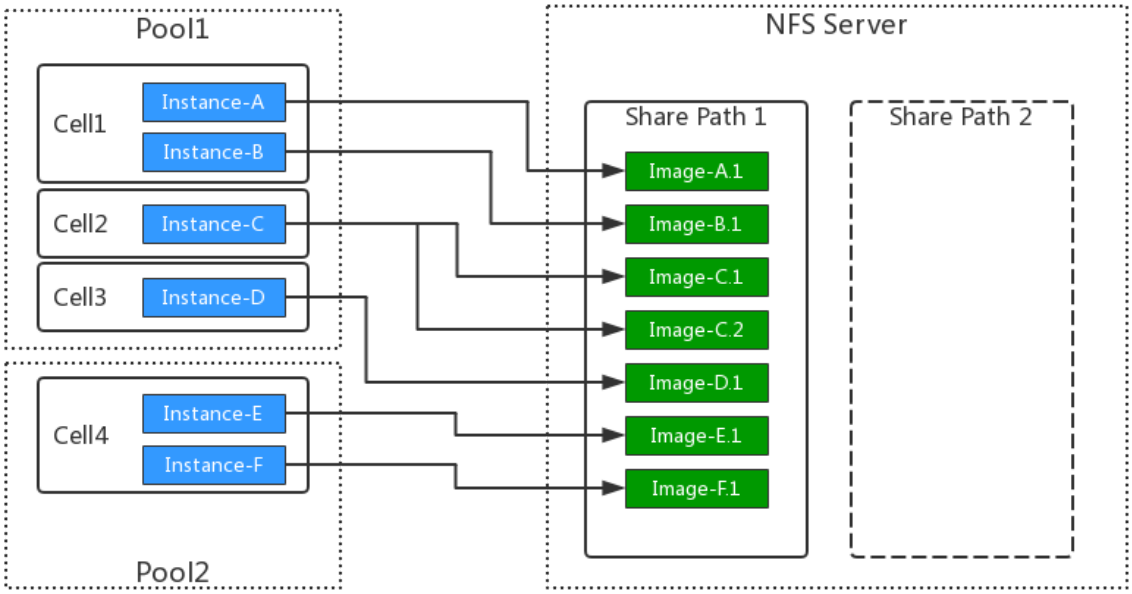
一个Nano集群可以拥有多个资源池，一个Cell节点只能归属一个资源池，通过资源池可以实现不同资源的隔离。每个资源池可以关联不同的地址池与存储池，同一个资源池中创建的实例，都使用相同的后端存储和地址配置，无需再为每个实例单独选择存储介质与绑定地址，这样可以批量创建具有相同特性的云主机实例，比如一个资源池专为VIP服务，生成的实例均使用SSD磁盘，另一个资源池用于高性价比用户，则使用SAS磁盘创建实例。

KVM/QEMU不断增加新的虚拟化设备类型，有的支持更新的特性，有的提供更好的性能，但是不是所有操作系统都能够识别它们。为了平衡兼容性与优化需求，Nano使用系统版本来标记一组特定的虚拟设备组合，比如CentOS 7使用VirtIO磁盘+网络设备，Legacy则使用IDE磁盘+RTL8139网卡。当创建实例时，只要选择系统版本就会使用对应设备来生成虚拟机，在保证兼容的情况下，获得更新的特性和更好的性能。

存储池

默认情况下，Nano中云主机实例的磁盘数据存储在承载Cell节点的本地存储系统中，无需额外配置，性价比也最高，但是当Cell节点出现故障时，使用本地存储的实例就无法继续工作。

存储池将专用存储设备，比如NFS服务器或者FC SAN封装为逻辑存储空间，当创建云主机时，Core模块首先在存储池中分配指定大小的磁盘卷，然后作为存储设备远程挂载到Cell生成的实例上。使用存储池的情况下，由于云主机的磁盘数据保存在存储后端，当承载的Cell节点故障时，系统可以自动在其他Cell节点重新恢复云主机实例。存储池同样可以用于将实例在节点之间迁移，用于分担和调整节点负载以提升集群整体性能和使用寿命。



一个存储池对应一个共享存储路径，一个存储池可以同时为多个计算池提供后端存储，但是每个计算资源池只能绑定一个存储池。

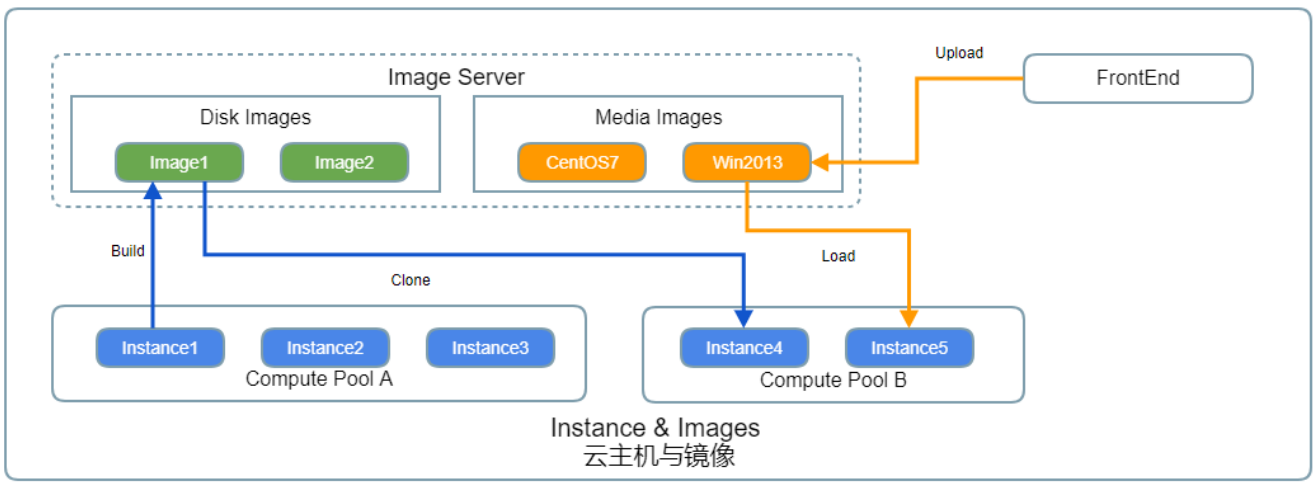
地址池

默认情况下，Nano分配的云主机实例通过桥接网络从现有物理网络设备中获取地址，但是对于希望能够精确控制实例IP的用户，可以使用地址池对网络资源进行更有效地管理。

一个地址池由一个或者多个可用地址段组成，当计算资源池创建实例时，自动从关联的可用地址段内分配新IP。当实例删除时，已分配IP释放回地址池，可以重新由其他实例申请。Cell节点在本地实现了一个微型的DHCP服务，用于将分配IP以及关联的网关、DNS等信息分配给云主机。

镜像

空白云主机并不能满足我们的日常使用要求，我们还需要安装操作系统和应用软件，Nano的光盘镜像和磁盘镜像能够快速部署可用云主机。



光盘镜像保存了ISO格式的光盘数据，上传到平台之后，可以加载到云主机中安装操作系统或者其他系统软件，通常用于定制模板云主机。

磁盘镜像保存云主机的系统盘数据，用户可以通过从任意云主机构建磁盘镜像，创建出的磁盘镜像能够快速复制新的云主机实例，并且直接获得与模板云主机相同的操作系统和预装软件，大幅度提高批量部署示例的效率。

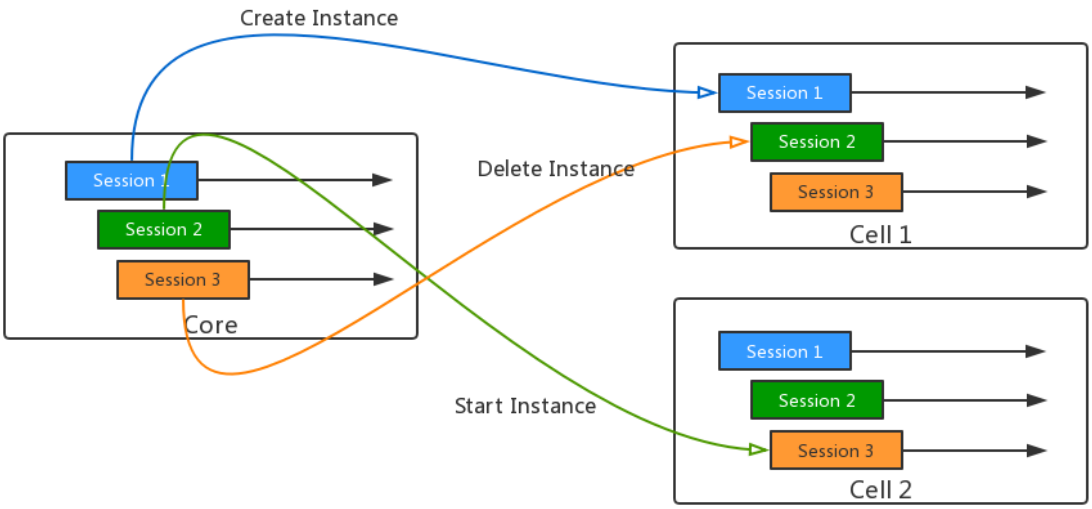
用户与资源可见性

Nano在FrontEnd上提供了用户、用户组和角色三个维度的权限管理，默认情况下，云主机实例或者镜像资源，都只对创建者可见。但是Nano允许通过组资源可见性的配置，允许访问同一用户组下的其他用户创建的资源，用于实现资源共享的效果。

任务处理

会话管理

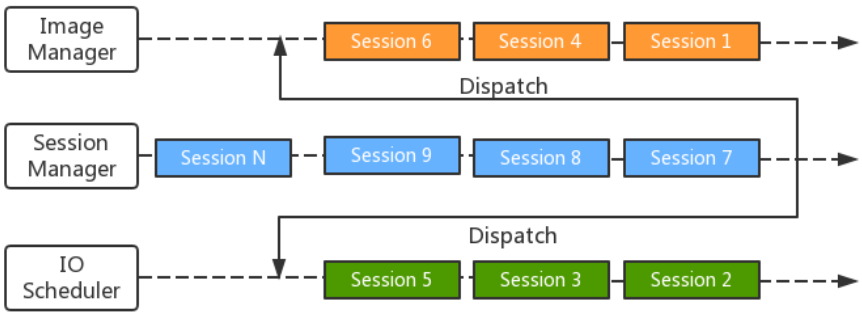
会话（Session）是Nano进行任务管理的基础单元，分布式系统中通常会有大量并发任务同时进行处理，Nano为每个任务请求分配一个独立的会话对象用于管理执行状态，会话对象跟踪任务执行阶段，记录原始请求、中间信息等数据，处理响应消息，必要时可以进行回滚或者提交。配合Nano的消息队列编排和事务管理机制，可以高效地并行处理上万任务。



每个参与内部通讯的Nano节点都有自己的会话管理机制，通常也会为每一个会话分配节点内唯一的ID。发送请求时，在消息中将源会话设置为该ID，接收到请求的模块，将响应消息发回请求模块，并且指定源会话进行接收，则会话管理系统可以保障由原始发出的会话接收到该响应消息，并正确处理。

消息编排

Nano基于Go的Channel特性，将任务处理流程切分成多个串行消息驱动流水线，只有收到消息才触发处理，处理完成后立刻抛出，然后继续处理下一个消息。这种模式确保每个处理流水线能够及时并且高效处理多个并发会话的请求，而不用浪费多余的时间片等待响应或者切换上下文，让效率达到最高。



数据与状态管理

现有主流产品在状态管理与配置存储上，主要依靠后台数据库管理，除了速度慢，管理复杂以外，还存在诸多不尽人意的地方，比如显示状态与实际状态经常不一致，需要手动同步；又比如无法及时检测节点失联；又比如数据库数据异常或者被人攻击等等，因此Nano在这部分环节进行了不少新的尝试。

Nano系统中的数据管理遵循以下原则：

- 性能优先，准确度优先
- 以实时状态为准，实时数据、内存数据为主，持久化数据为辅
- 如无配置，优先自动生成或者读取
- 精简数据结构，减少资源占用

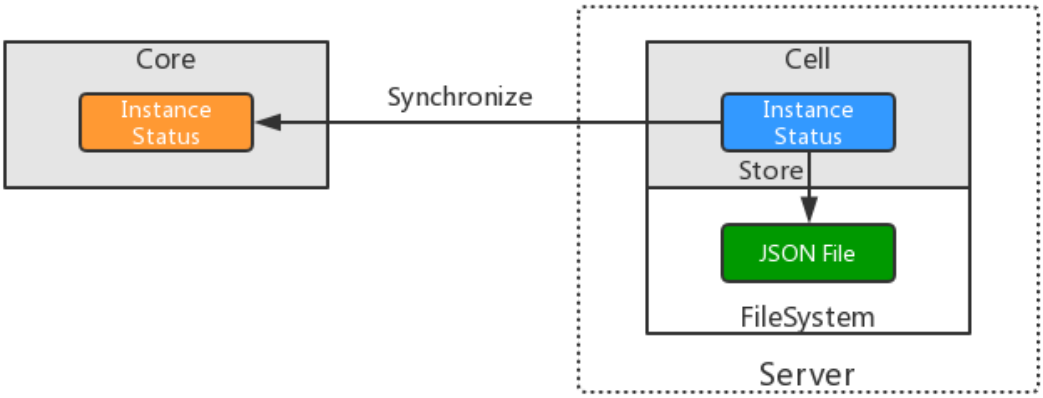
Nano设计了多种策略保障核心处理模块始终能够获取到最新而且真实的节点与资源状态，以此为依据进行处理，不用担心状态不一致或者延迟的问题。而且由于所有数据都在内存中处理交换，不仅状态更加及时和准确，速度也能够大幅度提升，每秒可以轻松完成数十万个请求，为大规模集群处理提供基准保障。

实例状态同步

每一个实例的状态和配置信息，同一时刻在Nano集群系统中至少拥有三个副本，一份在Core节点内存数据中，一份在Cell节点内存数据中，还有一份保存在Cell节点的配置数据文件中。

当Cell处于运行状态时，如果收到Core发来的创建实例请求，Cell生成了本地实例之后，会构建本地内存数据记录该实例状态，同时使用消息将实例信息告知Core，由Core模块在自己内存中构造一份相同的状态数据，最后Cell模块再将该实例配置写入本地配置文件中。此后，Cell会持续检测该实例运行状态，并将变更信息通知Core模块进行同步。由于Core模块拥有一个可靠的准实时状态，所以很多逻辑处理和判断在Core模块即可处理完成，无需再分发到Cell节点执行。

当Cell重新启动时，首先从本地配置文件读取所有实例信息，然后依次与实际运行实例进行比对校验，恢复或者修正数据。校验完成后，通过配置文件生成所有本地实例的状态信息保存到内存，在使用消息将本节点承载所有实例状态同步到Core节点，完成初始化启动，进入持续检测阶段，确保集群中的实例状态真实有效。

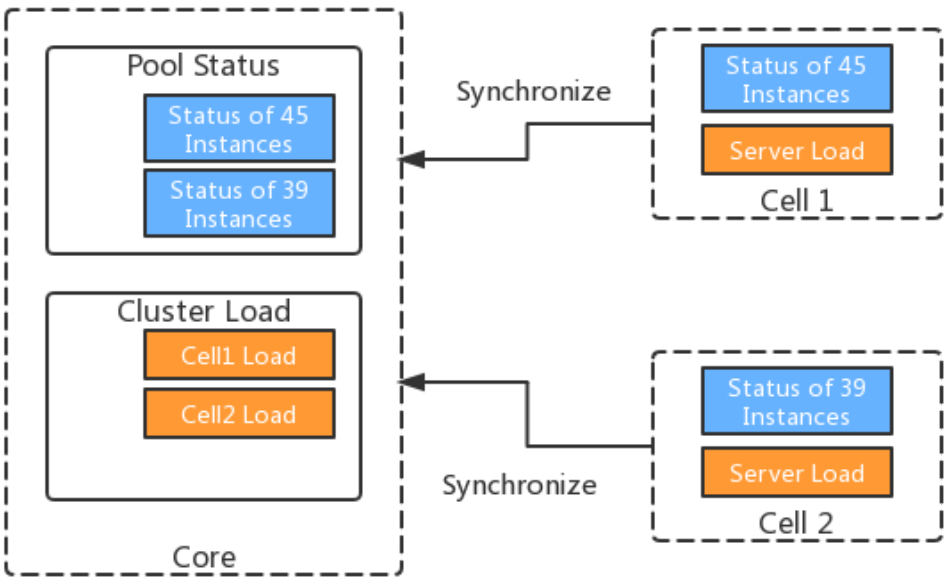


节点状态同步

Nano集群节点的同步分为两个层面。

第一个层面是虚拟链路层，Nano集群形似P2P网络，集群会检测每一个节点加入通讯域或者离开通讯域，从而变更关联配置、触发失联告警、暂时屏蔽节点或者启动故障切换，虚拟链路检测在模块运行中会持续进行。

第二个层面是应用层同步，每个模块的逻辑各自不同，比如Cell节点启动时，会自动向Core同步自己所有承载实例配置，同时从Core获取存储池、地址池等配置信息，用于更新自己本地配置。又比如Cell节点运行时，会以大约2秒的频率采集自己物理资源和虚拟资源的负载情况，通知Core节点，以便在资源池创建实例时，能够根据实际负载情况更有效地进行分发。即使因为特殊异常或者代码bug出现状态不一致，大部分情况下，只要重启模块即可自动恢复。



数据存储

Nano所有数据均在内存中进行处理，对于需要持久化存储的数据，基本上使用JSON格式的文本文件保存。JSON相比INI，能够更灵活地支持复杂数据与对象的存储，相比XML，格式更简单，尺寸更小。而且不仅Go自带JSON读写库，几乎所有主流开发语言都能够有效支持JSON文件格式。

每个模块需要保存的数据各不相同，但是存储数据时基本遵循以下规则：

1. 配置数据存放在config目录下，应用数据存放在data目录下，便于备份和恢复
2. 不同资源类型数据分为不同文件存放，比如实例数据与网络数据，避免文件尺寸过大，降低效率

由于Nano的任务处理都是基于内存数据，所以JSON数据实际上更多是作为数据备份手段来使用。当模块内部数据产生变更时，会根据不同的策略立即或者延迟更新到JSON文件中作为备份，当模块下一次启动时，会首先从JSON中读取初始数据，再继续执行。

数据安全

无论私有云还是公有云，数据和信息安全都是重中之重。作为轻量级平台的Nano，依赖程度更低，执行链条更短，系统组件更少，在安全防护上有天然的优势；即使如此，Nano在设计之初依然把数据安全和系统防护作为重点考虑。

镜像传输

Nano集群中，最大的传输数据来自于镜像文件，无论是从云主机构建磁盘镜像，还是从磁盘镜像生成云主机实例，又或者上传和加载光盘镜像，都涉及到大量的数据传送。Nano所有的镜像数据，均使用内部HTTPS协议传输，使用TLS加密，保障数据不被窃听或者截获。

监控安全

VNC是非常方便的云主机监控和管理手段，不少主流产品常常内置Web版本的VNC客户端配合WebSocket转发实现便捷的云主机监控，但是这种模式也常常被恶意利用，主要的原因有以下几点：

- VNC未加密或者使用统一密码
- WebSocket未加密被盗用或者劫持
- 第三方WebSocket服务端漏洞
- 云主机VNC端口被猜测

Nano为云主机配置VNC服务时，随机生成强密码强制校验，端口随机散列选择，避免端口猜测；然后Nano自己实现了一个更安全的WebSocket服务，每次请求动态创建转发通道，并且生成临时随机令牌，彻底杜绝重用攻击和第三方软件漏洞。在保证安全性的同时，Nano的云主机实例依然可以正常地通过第三方标准VNC客户端进行访问和管理，尊重用户的使用习惯。

Firewalld与Selinux

Firewalld和Selinux是现在linux系统标配的安全防护措施，与原有的手段有相当大的提升，但是对应用开发的合规性要求更高。Nano设计之初就是完全遵循安全规范设计，能够完美配合Firewalld与Selinux正常工作，保障系统安全性，无需降权或者关闭安全服务即可正常使用。

致谢

Nano起初只是个人练习Go语言自娱自乐的小产品，但是没有想到得到了各位支持者的不断测试和热心反馈，不知不觉走到了1.0正式发布，实在是出乎意外。

在此，特别感谢长期以来关心帮助Nano项目成长的各位朋友与同学们，也期望该项目能够不断进步，为更多热爱云计算与虚拟化的同学提供更多的帮助。