White Paper of Nano - a lightweight open-sourced cloud platform

White Paper of Nano - a lightweight open-sourced cloud platform

- **introduction**
- Features
- Differences
  - Golang
  - Auto Networking
  - No Database
  - Asynchronous Message Driven
  - Transaction Process
  - Network Model
  - Rich API
  - Divestiture of Resources and Business
- Design Essentials
  - Basic Working Principle
  - Communication and Networking
    - Discovery and Networking
    - Module Communication
  - Resource Model
    - Computing Resource Pool
    - Storage Pool
    - Address Pool
    - Images
    - User and Resource Visibility
  - Task Process
    - Session Management
    - Message Scheduling
  - Data and Status
    - Instance Status Synchronization
    - Node Status Synchronization
    - Data Storage
  - Data Security
    - Image Transport
    - Monitoring Security
    - Firewalld and Selinux
- Thanks

introduction

The birth of Nano originated from a project require a cloud computing platform to build big data development environment.

We investigated many cloud computing and virtualization platforms. Many products have lots of internal components which tight coupling with each other, also bundle with many complex concepts and configuration makes it difficult to learn and maintain.

Most of them also rely on a variety of third-party software and libraries, including databases, message queue, and so on. To build a cloud platform, you need installing 3 or 4 additional software, which makes the system more vulnerable, less stability, and harder to locate problems.

Some of them even mixed using multiple languages and scripts such as Java/Python/Bash, which is not only inefficient but also involves too much code for a simple task, making it hard to understand for the DevOps.

In generally speaking, the current products are too heavy for teams that only need simple virtualization, but lack resources for research and maintenance, so that why we created Nano.

Nano is a virtualization management software based on CentOS and KVM, which manages cloud instances backing by server clusters as resource pools.

It replaces manual operations with automated processes, provides a powerful platform while keeping things simple.

You can turn any server enable Intel VT-d or AMD-v technology into an IaaS platform, and begin deploying virtual machines in 3-minutes with a tiny Nano installer.

Nano uses MIT license, which is free for modification, personal or commercial use.

Features

- **Compact**: Around 30,000 lines of Golang code, less than 1/300 of OpenStack. Only three binaries to deploy, 9MB in maximum, no third-party software or dependency required, and easily replaced when upgrading.
- **Out-Of-Box**: Bundled with rich functions like the Web portal, guest monitor, instance clone, and failover. Automated configure processes including from module discovery, networking to device selection.
- **Reliable**: "All or Nothing" transaction mode, rollback when an error occurs, release all resources and resume state. Detect the status of every node in the cluster in realtime, automatically synchronizes the instance data for precise.
- **Expandable**: All functions backed by REST API. Divide resource services and the business layer for customization and integration. Goroutine-based abstraction enables rapid development neglecting the complex backend mechanism.

Differences

Golang

Nano developed by Golang, a new language designed by Google for backend service, all management functions implements without any external script using the rich libraries. Using Goroutine, we can speed up business development while maximizing server performance.

Most importantly, Go compiles to tiny binary, which is small in size and efficient in execution, no runtime lib or dependency required. The standalone binary can install on any environments, and easier to deploy and upgrade.

Auto Networking

Based on multicast protocol, modules automated discovery each other and establish communication without manual configuration.

When the device migrated or the address changed, it will update and synchorize associated modules automatically, Which ensures a reliable cluster in an unstable network environment.

No Database

Many products use traditional RMDB such as MySQL to store configuration and exchange data. The database is running on a slow disk, it also vulnerable to software failures or program bug which produce wrong data causing state abnormal or allocate fail. In that case, only the most experienced administrators could locate the problems in the backend database.

Besides, the database is a complex system, involving daily maintenance, backup, security, performance tuning, etc, it is too heavy and risky for a small team without DBA expert. Not a few production systems cracked by default database account or the database vulnerability not fixed in time.

Instead, Nano using dispatched JSON files to store configuration and data. All data collected, synchronized in real-time, and process in memory. All the status keep updated. In any case, you can restart modules to resume.

File storage is easy to backup and restore, make the system more robust.

Asynchronous Message Driven

Most of the current products use the RPC (Remote Procedure Call) to communicate. Although the RPC is simple to use, it usually executes synchronously, which means it must wait for a response or timeout before it continues to process.When there is an exception on the network or server, it often leads to blocking or slow response, so it is not suitable for the high-performance concurrent task.

Nano uses the asynchronous message commonly used in high-performance distributed systems to communicate and cooperate with the transaction processing module to allow hundreds of thousands of task running in parallel.

Transaction Process

The computer system is not a perfect system that never fails. On the contrary, when the cluster extends to a particular scale, disk damage or network failure is your daily struggle. When something goes wrong, most of the current product is just dropping the job. But the resources already allocated and the data is inconsistent, a manual recover is required.

All tasks of Nano running on the "All or Nothing" transaction mode, rollback when an error occurs, release all resources and resume state. Every task owns a session, which can commit or rollback separately, enable the rapid parallel process.

Network Model

Nano simplifies the network model of virtualization, the instance bridges to the physical network through a host interface by default. The instance obtains the address and network configuration through the physical router as a physical server, then send packets through the physical switch.

The instances are visible in the physical network, and administrators can manage and configure them through existing physical routers and switches as regular devices.

Rich API

Nano designed for customization and integration from the beginning. You can access all service and function via a unified RESTful API, which easy-to-use in most languages and frameworks, and leverage all potential of your application without any backdoor or hack API.

Divestiture of Resources and Business

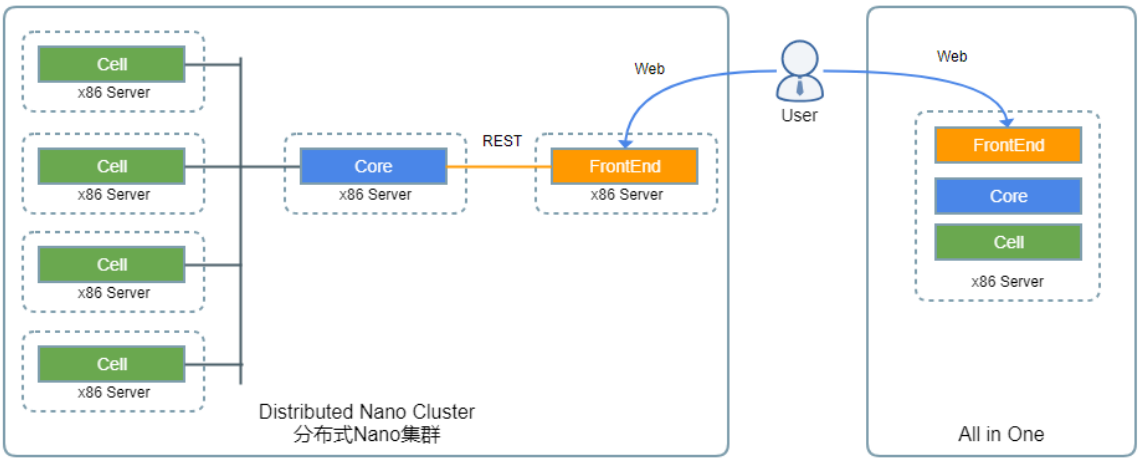
The architecture of Nano divide system into two parts, the frontend implement business workflow and the backend service provides resources management to support the frontend without concern the logic of the application.

You can build any frontend application as your wish, and no need to change the backend service or models.

Design Essentials

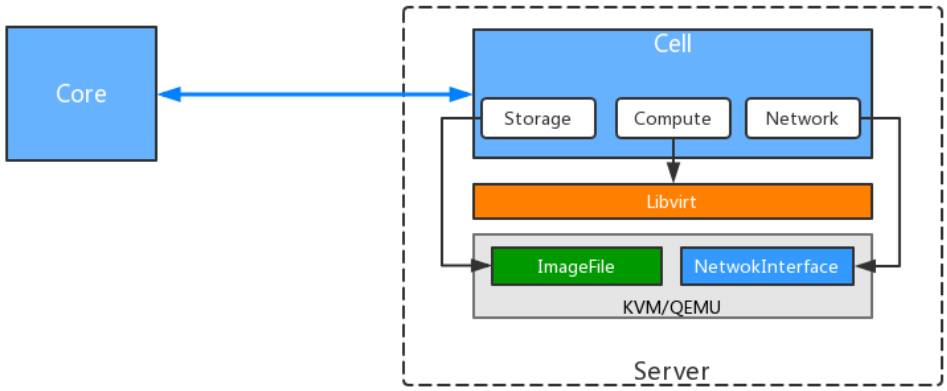
Basic Working Principle

Nano builds one or more x86 servers into a virtual resource pool. When a user issues a creating request, the Core module selects the appropriate host according to the system load of each node in the target pool, then send message to build a new instance, and schedule the subsequent management and recycle.



The Core is the main module responsible for the establishing cluster, monitoring status, scheduling resources, providing API, handling task requests. When establishing a cluster, start the Core first, then other modules could join.

The Cell module runs on every server needs hosting instances. The Cell module base on KVM and Libvirt, it keeps collecting the resource status and synchronizes to the Core module in real-time. It allocates resources and assembles into a new instance for users when received a creating request.



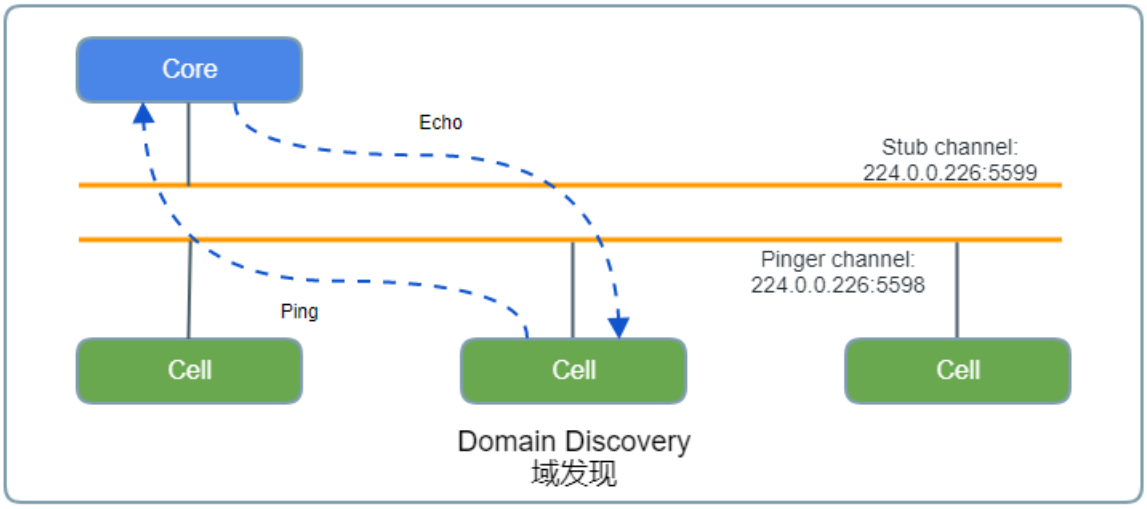
The FrontEnd module implements a Web portal backed by the REST API of the Core. Administrators can manage the whole system in this portal, both physical and virtual resources included. You can also build a series of Web App for their business.

Communication and Networking

The cluster nodes use UDP-based reliable transport protocol for communication. The current version uses JSON for message serialization and Key-Value mechanism to carry parameters which is easy to extend while maintaining backward compatibility.

Discovery and Networking

Each Nano node uses at least one UDP port to communicate, and in a production environment, a server usually has multiple network interfaces. You must have the correct address to establish connections. A Nano module uses the multicast to discover the domain address of the cluster, then chooses an unused port to establish communication.



No manual configure required, start the Core first, and make sure all modules use identical domain parameters.

Module Communication

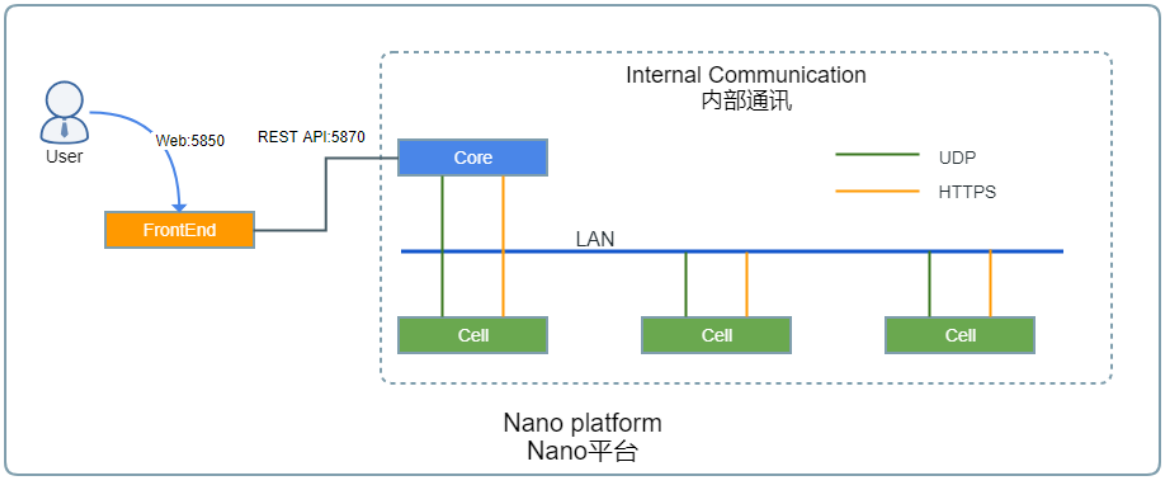
Nano has external and internal network communications.

The external communication is between the cluster and the external module, usually using a fixed address for access, likes the Web portal (TCP 5850 in default) and API service of the Core module(TCP 5870 in default), which can configure on your demand.

Internal communication works between cluster nodes, similar to P2P that every node has a unique name to communicate with each other. the communication network monitoring the node status, when a node joins, it automated establishes connections with the associated nodes and releases when it leaves.

Adapt to the unstable network environment, the communicate address chosen automated by the system without manual configure.

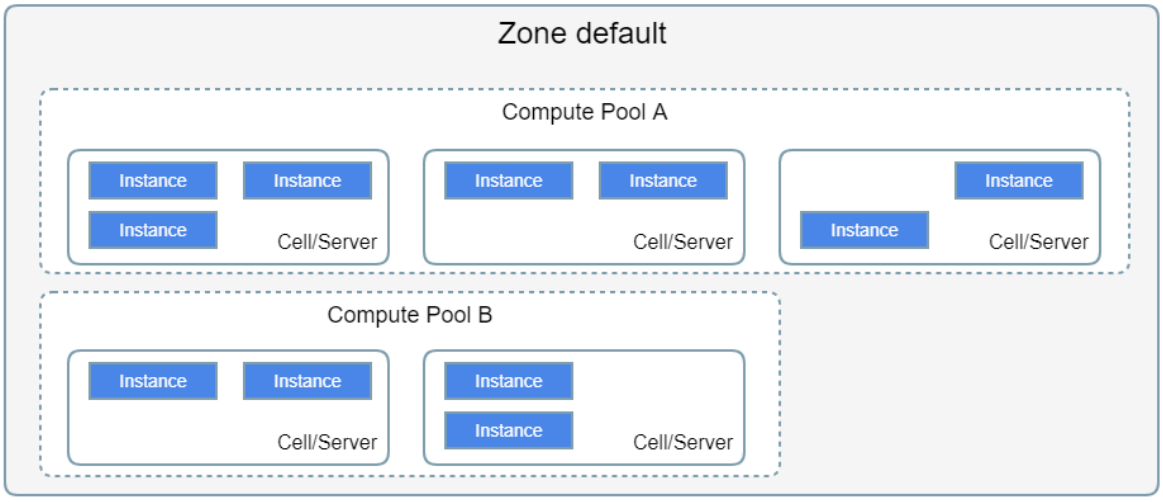
When the server migrated or changed address, the module will update the latest configuration and resume associations, and all subsequent requests switch to the new address for processing.



The internal HTTPS stream is used to transport encrypted binary data, and UDP is used to transmit control messages. The control message usually contains the names of sender and receiver, which will ensure delivery by the framework of Nano after sent. the UDP messages can also use for rapid internal messaging in a node which has multiple submodules.

Resource Model

Computing Resource Pool



The computing resource pool is the center scheduling unit, composed of one or more servers with Cell nodes installed.

The Cell keeps collecting the status of this node and report to Core. When Core receives a creating request, it will choose a host with lighter load depends on the current status of every server in the pool, to distribute the pressure on the system and prolong the life of the server.

A Nano cluster could configure multiple resource pools, but a Cell node only belongs to one resource pool, the resources isolate by pools. A computing pool could associate with an address pool and a storage pool, instances created in the same pool use the identical storage and address backend, no need to configure individually. For example, if one resource pool dedicates to VIP with SSD backing instances, and the other resource pool serves cost-effective users, new instances bundle with SAS disk.

Lots of new virtualized devices added to KVM/QEMU, some has more features and others providing better performance, but not all operating systems recognize them. Nano optimizes the combination of virtual devices based on the system version, when creating an instance, uses the corresponding configuration of the target version to build the virtual machine to balance compatibility and performance.

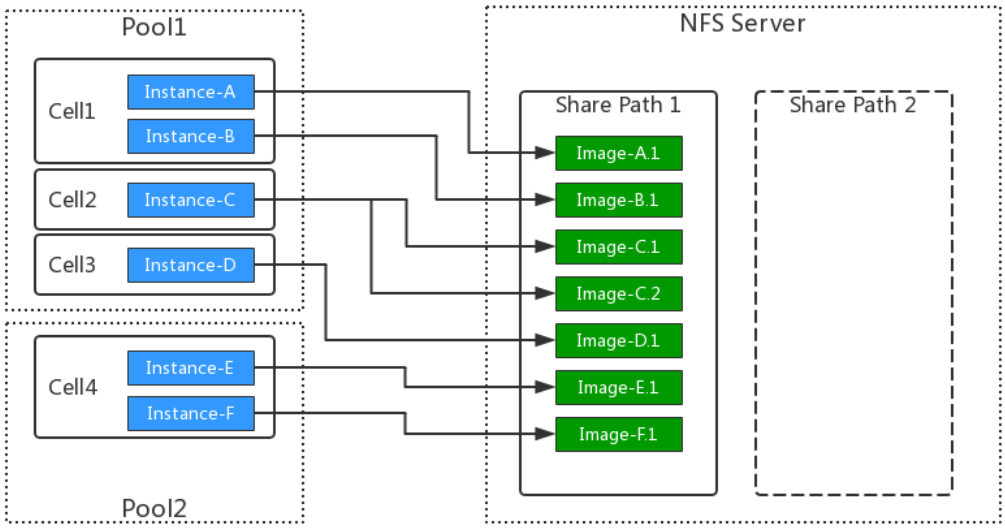
For example, the "CentOS 7" uses VirtIO driver for disk and network, but "Legacy" uses IDE disk and RTL8139 network interface.

Storage Pool

By default, the instance data stored in the local file system of the host, which is cost-effective and fast, but if the server crashes, all instances will shutdown.

The storage pool encapsulates dedicated storage devices, such as NFS servers or FC SAN, as logical storage space. When creating an instance, the Core module allocates virtual volumes from the storage pool, then mount as a storage device.

Because the instance data stores on the backend storage pool, the system could restore instance on other nodes automated when a server crashed. You can also migrate instances between nodes to rebalance system pressure when backing with the storage pool.



A storage pool map to one shared storage path, one storage pool can back multiple computing pools at the same time, but each computing pool can only bound with one storage pool.

Address Pool

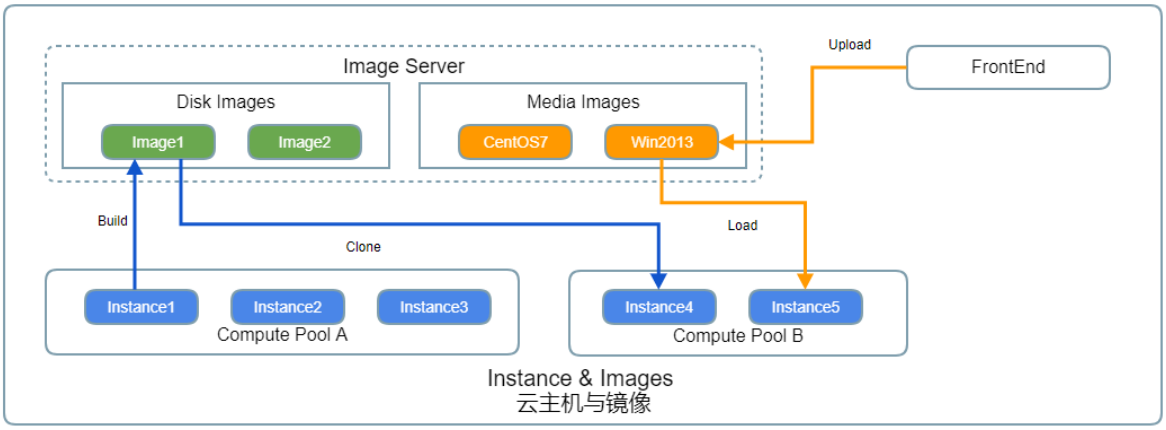
By default, the instances of Nano obtain addresses from the physical network through bridged networks. But for users want to manage the instance IP more accurate, the address pool can be useful.

An address pool consists of one or more available address segments. When creating an instance, a new IP allocates from available segments in the attached address pool. When deleting the instance, the IP returned to the address pool and available for another request.

The Cell implements a tiny DHCP service, which assigns the allocated IP and the associated gateway, DNS, and other information to the local instances.

Images

A clean instance without operation system and application is useless, but you can fast deploy a usable one using the media and disk image.



The media image represents a DVD data in ISO format, which can load into instance working as a physical CD for installing OS or software. It usually is used to customize a template instance.

The disk image represents the disk data of a virtual machine, which can be used to fast clone with the identical OS and software as the original one in a short time. You can build a disk image from any instance.

User and Resource Visibility

Nano provides privilege management over users, user groups, and roles by the FrontEnd.

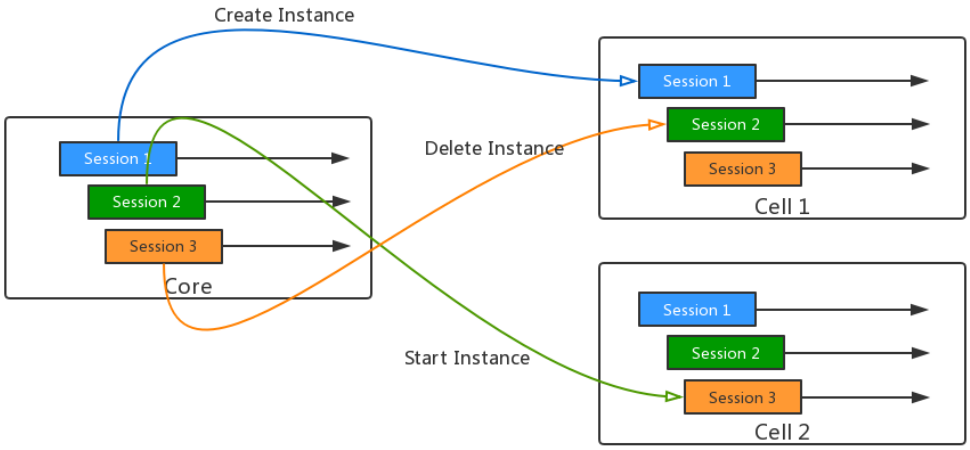
By default, resources such as instance and image are only visible to their owner. However, the administrator could configure group resource visibility to allow access to resources created by other users of the same group, which enables sharing and collaboration.

Task Process

Session Management

In distributed systems, there are usually a large number of concurrent tasks to process.

The session is the basic unit of task management. Nano allocates a session object for each task to manage the executive state, such as tracks the running phase or records temporal data, it could roll back or submit when necessary.



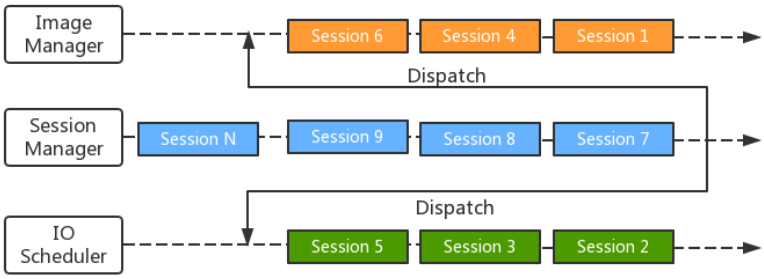
Every node involve in internal communication has its session management mechanism, and each session is usually assigned a unique ID within this node.

A message store the source session ID, when received, the receiver sends the response to the original sender, which enable multiple task multiplex into one connection.

Message Scheduling

Based on the Channel of Golang, Nano divides the processing flow into multiple serial message-driven pipelining. Invoked by received message, throw out immediately after the processing completed, and then process the next one.

This model ensures that each processing pipeline can handle multiple concurrent tasks without waiting for a response or switching context to maximize efficiency.



Data and Status

Most current products depend on a database to store and exchange data. However, the database, which is slow and troublesome, often causes data corrupted or security risk. So Nano made a lot of new attempts in this part.

The following are data management principles of Nano:

- Performance first, accuracy first
- Real-time data in the memory is primary, using persistent data as backup.
- Automated generate when no configuration available
- Simplify the structure and reduce the occupation

Nano designed a variety of strategies to ensure that the core processing module can always get the latest and reliable status. Because all the data are processed and stored in memory, which is much faster than the database, it could easily handle hundreds of thousands of requests, guarantee essential performance for large-scale cluster processing.

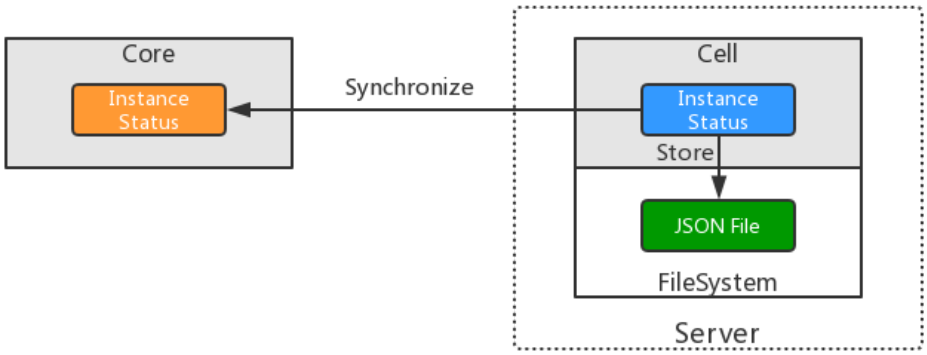
Instance Status Synchronization

There are always three copies of instance status and data in the cluster: two in memory of the Core and Cell modules, other in the configure file on the Cell.

When created a new instance, the Cell store the instance data in memory, then notify to the Core. After the Core created a duplicated data, the Cell saves the configuration into the local file as a backup.

When Cell restarts, it will load all configuration from local files, then compare with running instances, recover or correct the data if necessary. After verifying the data, the Cell notifies all instance status to the Core and enter monitor routine.

The Cell will synchronize any change of instance status to the Core, which allows the Core to handle a lot of requests without communication with the Cell. Faster response and lower footprint.

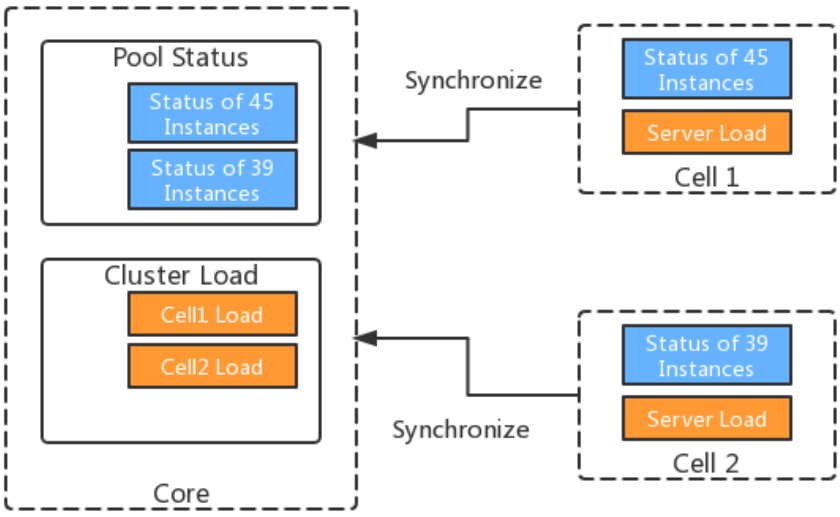


Node Status Synchronization

All Nano modules work on a virtual link layer similar to P2P, the cluster monitor status of every node. When nodes join or leave the domain, the system will update the association, reroute request or invoke failover. The link detection keeps running in the background.

Data synchronization varies by module type. When started, the Core automated synchronize all instance status to the Core, then fetch storage and address pool configuration from the Core. The Core collects loads of physical and virtual resources about every 2 seconds and notifies the Core that it can balance system pressure more precisely.

Furthermore, if you encounter any data or status inconsistent, restarting the module may resolve the problem in most cases.



Data Storage

All data in Nano processed in memory, and the data needs to be persisted will save in a JSON file. JSON can store more complex data than INI, and much smaller than the XML format. Most development languages can easily handle JSON format, including Golang.

Although the data that needs to store varies from module to module, the following are principles when saving files:

1. Configuration data stored in the "config" directory and application data store in the "data" directory which is easy to backup and restore.
2. Divide storage files by resource types, like instance and network, to reduce file size and improve efficiency.

Because all tasks of Nano use the data in memory, the data stored in JSON is more like a backup method. All data change will update to the JSON file eventually, and the next time the module starts, the JSON data will load as initial status.

Data Security

Whether private or public cloud, data security is always a top priority. A lightweight product has less dependence, shorter execution chain, and fewer system components, which are advantages in security protection. Even so, Nano still focuses on data security from the very beginning of the design.

Image Transport

The largest transfer data in Nano comes from image files, whether building a disk image from an instance or uploading a media image. All the file data in Nano transmit via HTTPS and encrypted by TLS to protect from eavesdrop or intercept.

Monitoring Security

VNC is a very convenient method to manage instances. Many products embedded a Web VNC client via WebSocket forwarding to monitor the instance, but it is vulnerable to malicious intruders.

following are common weaknesses:

- VNC is not encrypted or uses the same password
- Unencrypted or hijacked WebSocket
- third-party WebSocket server vulnerability
- VNC port guessing

When Nano create a new instance, it generates a strong password and chooses a random hash VNC port. Then Nano implements a more secure WebSocket service, dynamically creating forwarding channels per request and generating temporary random tokens, eliminating reuse attacks and third-party software vulnerabilities. For user's convenience, any standard through third-party VNC clients still able to access the instances created in Nano.

Firewalld and Selinux

Firewalld and Selinux are new standard security mechanism of the Linux system. They made significant improvements in security, but become very strict with software compliance.

The Nano following the security design specification from the beginning, it is running with Firewalld and Selinux smoothly, do not need lowering your security to work.

Thanks

At first, Nano was just a small product for practicing Golang, but I didn't expect to get so much enthusiastic feedback from our supporters, and it was a surprise that it reached the official launch of 1. 0

Here, I would like to thank all the supporters who concerning and helping the growth of the Nano project for a long time. I also hope that the project can keep going and help more who love cloud computing and virtualization.