阿里云 | OpenAnolis 龙蜥社区

云原生节点管理最佳实践白皮书

BEST PRACTICES FOR CLOUD-NATIVE NODE MANAGEMENT WHITEPAPER

引言

背景介绍

随着云计算和云原生技术的广泛应用，越来越多的应用程序在诞生初期就成为云的原著民。在云原生的浪潮下， Kubernetes 集群在规模和数量上快速增长，进而使得在大规模集群中，节点管理的重要性日益突出。在这样的环境下，高效的节点管理成为确保集群稳定性、性能优化和资源利用率最大化的关键因素。

计算节点作为云原生架构的关键组成部分，节点的管理直接影响着整个云原生应用的成本和稳定性。然而，现有的基础架构管理方法更偏向于传统的虚拟机管理理念，缺乏对工作负载的深度感知，无法适应大规模集群的节点管理要求。

面对大规模的节点管理的场景，越来越多的人愿意尝试云原生式的节点管理模式。云原生节点管理是基于云原生理念，使用专为此目的设计的操作系统底座 ContainerOS 和配套基础设施，提供的一种有效的节点管理方案。这种新的管理方案旨在优化云上环境的大规模节点的管理成本，并同时提供更佳的弹性、灵活性、稳定性和安全性。

节点管理现状和面临的挑战

计算节点是云原生架构的基石，承载着工作负载和集群核心组件，对整个系统的可用性和性能至关重要。有效的节点管理能够确保节点的稳定性、弹性和安全性。在云原生环境下，传统的节点管理方式面临着以下挑战。

挑战 1：大规模节点的自动化部署和扩容

Kubernetes 提供了弹性的部署环境，可以迅速扩展 Pod 副本以适应业务压力的迅速增长。为此，在 Kubernetes 集群中需要预留一定的计算资源来支持 Pod 的横向扩展，这预留的标准就是集群预警水位。

预警水位的高低直接影响了集群使用成本，如果水位过低，就会因为机器的闲置而导致资源的浪费。在云上环境中，依托于云厂商云主机（如阿里云 ECS 等）的弹性，使得 Kubernetes 集群可以采用较高的预警水位，在业务高峰期提前扩容 Kubernetes 节点以支持更多的工作负载。

但是，Kubernetes 节点的扩容过程往往需要花费数分钟的时间，大规模的节点扩容甚至可能需要十几分钟，时间敏感的业务可能会因瞬时容量不足导致业务损失。

挑战 2：节点状态的实时监控和故障恢复

当集群的规模足够庞大时，集群中节点在运行过程中出现故障会成为常态，例如网络抖动、异常重启、底层硬件故障等。而且，对于分布式系统来说，由于爆炸半径各有大小，如何实时监控节点状态，快速响应故障情况以避免故障扩大，成为新的挑战。

同时，节点监控本身也需要消耗资源，例如 cgroup 的采集、proc 系统的采集等。在密集部署工作负载的情况下，这种资源消耗会更加严重。如何以更低的成本监控节点的健康状况成为高密度容器部署所需要考虑的首要因素之一。

挑战 3：大规模节点的运维自动化

在大规模集群中，即使是常规的运维操作也会变得充满变数，包括操作系统的升级、安全补丁的应用、软件包的管理、kubelet 或 containerd 的自定义配置等。为了保证将集群内的所有节点安全、平稳地更新到一致的状态，不仅需要具备大规模节点变更的能力，还需要具备变更操作的审计和回滚能力。

在运维操作中，若由于错误而导致节点状态不一致，即部分节点的配置与预期不符，甚至同时存在多个版本的节点，不仅会大幅增加下次运维操作失败的风险，还可能使得相同的业务副本在部分节点上出现非预期行为，进而引入业务的稳定性风险。

本白皮书的目的和范围

本白皮书的目的是探索和总结云原生节点管理的新范式，重点介绍面向云原生场景设计和优化的 ContainerOS 及其在云原生节点管理中的关键角色。我们将深入了解 ContainerOS 及其配套基础设施的能力和特点，阐述为大规模集群管理场景进行的优化和云原生节点管理方案。

本白皮书的范围将涵盖云原生节点管理的核心概念和关键技术，并结合行业最佳实践，提供降低节点管理成本，提高稳定性和安全性的可行方案和具体建议。我们希望通过本白皮书，引起读者对云原生节点管理的关注，并为他们提供全面的理解和应用指南。

目录页

一、云原生节点管理概述 7

1. 云原生节点管理的定义 7

2. 理解 Kubernetes 节点管理成本 8

3. 降低节点管理成本的重要性 10

二、ContainerOS 概述 12

1. 传统操作系统在云原生场景面临的问题 12

2. ContainerOS 的设计原则 13

3. ContainerOS 在云原生节点管理中的角色 14

三、ContainerOS 特性介绍 17

1. 专注于容器化应用 17

2. 安全提升 18

3. 原子升级与镜像版本化 19

四、节点的生命周期 22

1. 千节点扩容的弹性 22

2. 节点运维监控工具 23

3. 节点声明式配置 25

4. 节点故障自愈 28

五、阿里云最佳实践和客户案例 31

1. 在阿里云容器服务中使用 ContainerOS 实现极速扩容 31

2. ContainerOS 助力阿里云 ECI 极致弹性 34

3. 蚂蚁安全科技镜像加速实践 35

六、尾声 39

1. 云原生节点管理的基本逻辑 39

2. 未来节点管理的发展趋势 39

云原生节点管理概述

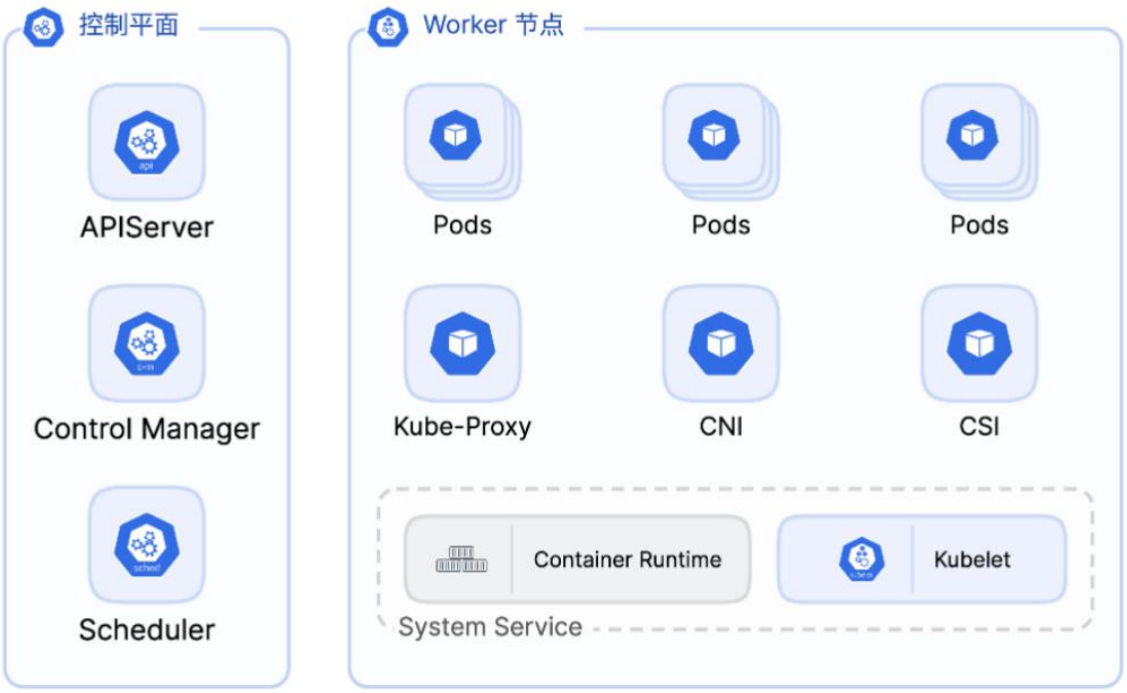
Overview Of The Cloud Native Node Management

一、云原生节点管理概述

Kubernetes 是开源的容器编排平台，用于自动化部署、扩展和管理容器化应用程序。Kubernetes 的基本架构由几个核心组件组成。

首先是控制平面，它是集群的控制中心，负责管理整个集群的状态和配置。控制平面包括三个组件：API Server 提供集群的 API 接口，Scheduler 负责调度工作负载到合适的节点上运行，Controller Manager 处理集群中的各种控制器任务。

其次是 Worker 节点，它是集群中的工作节点，负责运行和管理容器化应用程序。Worker 节点核心包括两个组件：Kubelet 是节点上的代理服务，与 Master 节点通信并管理容器的生命周期，Container Runtime 负责运行容器。



1. 云原生节点管理的定义

Worker 节点（简称节点）是构建云原生应用平台的基础，承载着管理容器生命周期和物理资源的重要任务。通常情况下，节点有以下具体的职责：

- **提供容器运行环境**：节点使用容器运行时来处理容器的创建、启动、停止和销毁。通过全生命周期管理，使容器能够始终保持在期望状态。

- **合理分配资源**：节点负责为工作负载分配所需的运行资源。包括计算资源（如 CPU 和内存）、持久化存储和网络资源等。通过合理的资源分配，节点不仅能够满足容器的运行需求，更能保证不同容器间的资源隔离。

- **提供高可用和故障恢复**：节点应具备基本的高可用和故障恢复能力，在可预料的异常发生时，主动干预使工作负载向期望状态靠拢，以提供基础的稳定性和可靠性。

但同时，节点也存在局限性，一方面单节点无法解决非预期的错误，比如容器运行时的异常，节点的恢复手段十分有限。并且由于缺乏全局视角，在集群容量不足时，因单节点的故障导致整个集群的雪崩也时有发生。另一方面，由于宿主节点自身的管理并没有被 Kubernetes 集群标准化，随着集群规模变得庞大时，千奇百怪的手动运维操作，极易使得集群中节点的配置存在差异，加剧环境腐化。

云原生节点管理是指在云上环境中，利用云的弹性、可用性和计量计费等特点，最大化发挥节点的自管理能力，并通过集群化手段弥补单一节点的局限性，构建成本可控、易于管理、敏捷、安全和高可用的集群基础设施。通过云原生节点管理实践，可以有效的进行大规模集群的管理，满足不同工作负载的需求，并确保整个集群的稳定运行。这种管理实践对于构建可靠、可扩展的云原生应用平台至关重要。

2. 理解 Kubernetes 节点管理成本

Kubernetes 是业界云原生应用平台的事实标准，同时也是一个复杂的分布式系统。Kubernetes 的创建者之一，Heptio（VMware）的 Joe Beda 曾表示：

> Kubernetes 是一个复杂的系统，它带来了很多新的抽象，但这并不适合所有问题。我确定，很多人通过更简单的工具实现 Kubernetes 的功能。

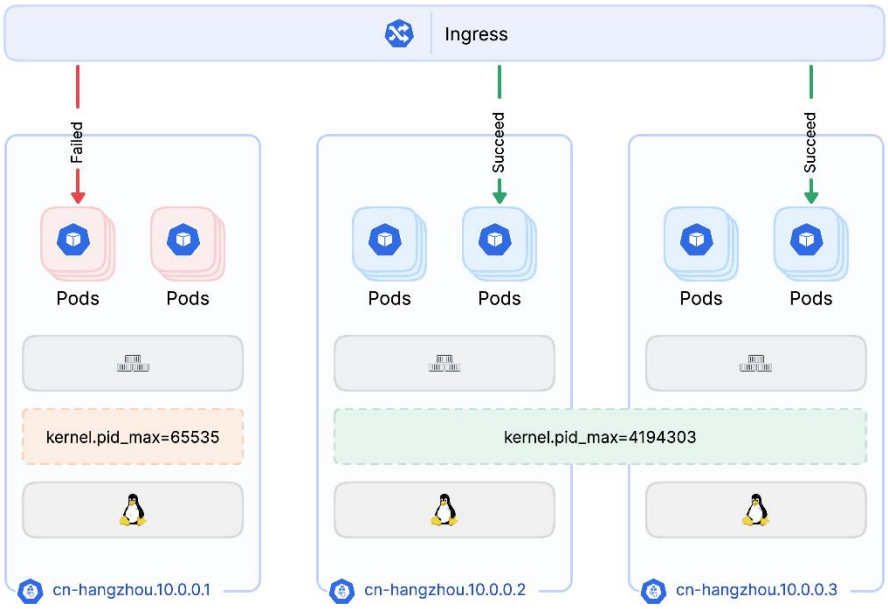
和其他所有的具备生命周期的软件系统一样，集群并不是一成不变的，而是根据业务实际需求动态的调整。无论是在集群内增减部署集，还是根据实际情况对集群节点进行扩缩容。day2 运维操作使得集群的整体状态一直处于变化中。

作为分布式系统，Kubernetes 也天然具备了分布式系统的复杂性和风险，而环境动态变化引入的不一致问题，加剧了故障的可能性和排查难度。所以，随着集群规模的增加，集群的可用性反而会下降，节点管理成本也会不可避免的上升。

所幸 Kubernetes 的使用和运维可以利用系统化和工程化的手段降低复杂度，并提高整体可用性和降低管理成本。但首先我们需要理解什么是节点管理成本，节点管理成本主要涉及硬件成本和运维成本两部分。

硬件成本是指集群所管理的资源成本。在提高部署集规模的同时，对计算资源的需求也会相应增加，为了满足负载需求，需要考虑节点的计算能力、存储空间和网络带宽是否足够，这给容量规划带来了较大的挑战。合理的容量规划可以避免因硬件资源不足而导致的性能问题和系统崩溃，而不合理的容量规划会导致大量资源闲置，产生资源浪费。

运维成本是指在日常的部署和节点运维中，需要投入的资源和人力。一方面，需要确保工作负载符合预期，另一方面，也需要保持节点配置和状态一致，以避免环境腐化。无论是操作系统和集群版本的升级操作，还是系统或容器运行时的配置和调优，大规模的节点管理是一个复杂的任务。需要时刻保持正确且最终一致的部署形态和环境配置，否则很容易造成应用行为异常或引入稳定性风险。



更多的节点数量，也意味着更大的节点故障可能性。当节点发生故障时，需要及时发现并采取相应的措施来恢复服务。这包括诊断故障原因、迁移工作负载以及修复或替换故障节点等。故障处理的复杂性随着节点数目的增加而增高，需要投入更多的时间和人力来保证集群的稳定性。

3. 降低节点管理成本的重要性

降低节点管理成本在构建可持续发展的云原生应用平台中十分重要。在云上环境中，我们可以利用云的标准化和弹性等特点，以通过系统化手段降低集群整体复杂度的方式，获得更可控的管理成本和更稳健的应用平台。采用云原生节点管理实践，可以获得以下益处。

节约计算资源成本：充分利用云上环境的极致弹性特点，对集群内的资源使用状况动态感知，可以根据实际需求进行峰时扩容，低谷时缩容，避免资源浪费。这种灵活的资源调配方式可以有效降低计算资源的开销，降低硬件成本。

提高运维效率：通过大规模自动化和面向终态的配置能力，减少部署和配置的复杂性。自动化的节点管理流程可以减少人工操作的错误和时间消耗，提高运维效率，降低节点运维成本和故障风险，使运维人员能够更专注于核心业务，提高整体运维人效。

提高可用性和安全性：一致的执行环境可以大大降低应用程序出现异常的可能性。通过节点实时监控和节点自愈能力，可以及时发现并解决节点故障，确保应用程序的稳定运行，减少业务中断和损失。

ContainerOS 概述

Overview Of ContainerOS

二、ContainerOS 概述

云原生节点管理是以云原生理念为基础的实践方法论，主要目标是在满足日常运维需求的同时，有效应地应对大规模集群节点管理的挑战。而操作系统作为节点底座，是云原生节点管理的重要组成部分。

1. 传统操作系统在云原生场景面临的问题

Linux 内核诞生至今已三十余年，催生出众多的 Linux 发行版与繁荣的生态。为了适应各种使用场景和各式各样的软硬件环境，传统的 Linux 发行版提供了复杂而完备的功能，包括硬件驱动、软件包、系统库和系统服务等。

然而随着容器技术的出现，业务逐渐容器化，业务的运行依赖已经通过容器镜像实现了自包含。这意味着底层操作系统只需要支持容器运行时即可，不再需要提供大量的额外功能。在云环境中，云厂商的虚拟化技术使得硬件资源的管理变得简单，不再需要操作系统内核提供过多的硬件支持。

因此，传统的 Linux 发行版在云原生场景下存在如下问题。

问题一：体积臃肿

面向通用场景的传统操作系统发行版内置了过多容器场景不会使用的软件包和系统库，提供的多余功能不仅导致镜像体积增大，还会占用多余的 CPU 和内存资源。此外，这些多余的系统服务和软件包还可能引入额外的安全风险，因为它们可能存在未修复的 CVE 漏洞。

问题二：版本零散

传统操作系统以软件包为粒度进行系统的管理，一个操作系统镜像的版本等同于里面所有软件包版本的合集。管理员对操作系统的管理需要细化到每一个软件包的版本，管理复杂度高，随着集群规模的增加，运维工作量往往成倍增加。

问题三：运维方式落后

集群内网络、存储、常规系统资源（如 CPU、内存） 都可以通过 Kubernetes 进行管理，唯独操作系统自身是独立于集群的控制平面的，对操作系统的运维大多通过 ssh 直接登录系统进行操作，即我们常说的「黑屏运维」。运维粒度为单机，运维效率低、难以追溯且容易出错，大规模集群环境下极易导致集群内各个节点状态不一致的情况。而且操作系统自身的状态很难被 Kubernetes 感知并进行协同。

2. ContainerOS 的设计原则

为了应对上述一系列问题，同时给云原生用户带来更好的体验，专为容器负载而设计的容器优化操作系统应运而生，也就是我们通称的 ContainerOS。顾名思义，ContainerOS 聚焦在云上容器场景的功能与业务需求，这样得以摒弃传统操作系统大而全的设计理念和历史包袱。基于如下一些原则，我们设计了一款 ContainerOS。

原则一：小型化与极速弹性

因为 ContainerOS 并不存在通用操作系统的负担，可以专门为容器场景优化，包括容器网络性能优化，资源监控和控制能力优化等，配合更精简的系统设计，用以满足大规模集群中业务 Pod 极速伸缩的诉求。

原则二：安全增强

容器化技术使得应用的运行依赖通过容器镜像实现自包含，这样对底层操作系统的依赖减少。基于此前提，我们可以通过一些相对“激进”的手段来确保操作系统自身的状态处于预期的状态。比如，将根文件系统设计为只读以防非法程序或逃逸容器篡改底层操作系统、受控的运维通道、默认启用 SELinux 强制访问控制等手段，尽可能避免相对开放的云计算平台带来的安全风险。

原则三：镜像原子更新与版本化管理

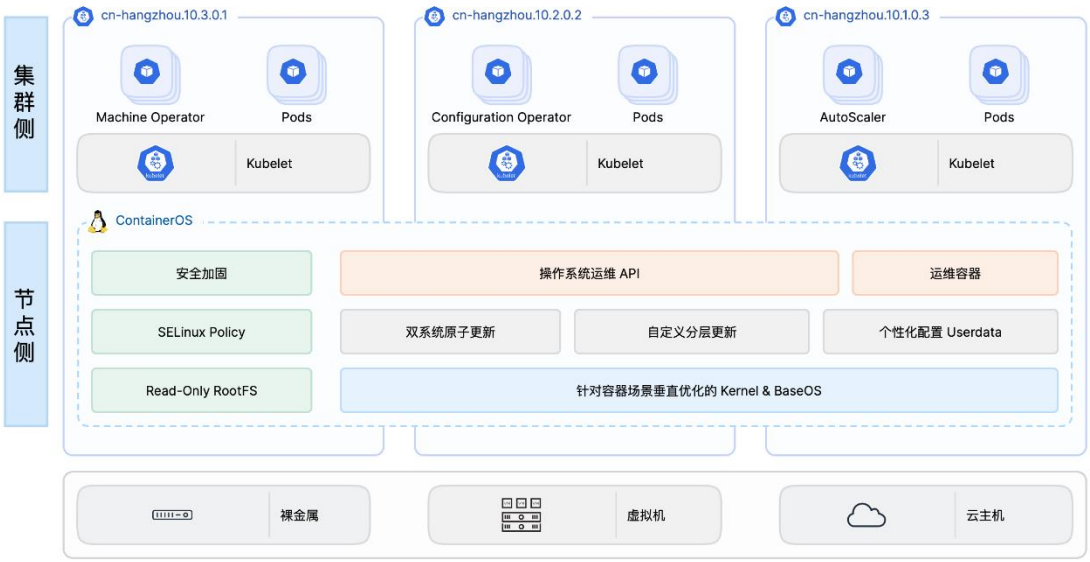
像管理 GIT 代码仓一样管理整个操作系统，任何文件的变更可被记录成为一个新的版本，版本变更过程可记录、可追溯、可回滚。

原则四：云原生场景开箱即用

默认提供云原生场景必备组件，整个系统无需过多配置，用户仅需要关注自身的业务部署情况。

3. ContainerOS 在云原生节点管理中的角色

云原生节点管理提供了集群侧管理和节点侧管理两个维度的理论实践和工具集。这些工具和 ContainerOS 一起构成了可开箱即用的基础设施，帮助用户更好地理解和掌握云原生节点管理的最佳实践。



1） 节点侧管理

节点侧管理的实现主要基于 ContainerOS 以及其内置组件。ContainerOS 为容器化业务进行了内核级的优化，促进容器化业务更快更平稳地运行。

同时操作系统层面提供 API 支持以整个镜像为粒度的原子升级能力和类似于容器镜像的分层更新能力。分层变更是一种基于声明式规则的动态变更机制，它可以动态地变更操作

系统、Kubelet 和容器运行时的版本和配置。这种变更方式采用叠加层的方式进行，可以支持变更回滚和历史版本回溯。

操作系统原子升级能力提供了内核版本和系统软件包的升级能力。ContainerOS 支持新旧两个系统版本同时存在，这意味着节点可以在运行期间准备新版本的操作系统，只需一次重启操作，就能完成系统和软件包的升级，大大减少停机时间。同时支持版本快速回退，符合云原生场景下常用的 RollUp & RollBack 的灰度发布、回滚运维动作。

2） 集群侧管理

在集群侧，多个控制器通过声明式配置的方式相互协作，完成集群内节点的全生命周期管理。

Autoscaler 提供了基于规则的集群扩缩容能力。它根据预设的规则灵活地控制集群的节点数量，以满足不同的业务需求。无论是在业务高峰期还是低谷期， Autoscaler 都可以自动增加或减少节点，提高资源利用率，减少运维工作量，并保持集群的稳定运行。

Machine Operator 支持统筹编排节点任务，基于 ContainerOS 的原子 API 和云平台的能力，对集群内节点统一管理，包括操作系统变更、核心组件升级等。Machine Operator 使用声明式管理，通过定义节点期望版本和状态，支持全自动的分批操作。从而保证集群的一致性和稳定性。

Configuration Operator 实现了对操作系统、Kubelet 和容器运行时配置的统一管理。不仅可以基于声明式配置对节点分批修改，也支持配置的版本管理，使得配置变更可追溯、可回滚。

声明式和自动化的批量节点操作，减少了中间状态，提高了集群的运维效率，保证了集群节点的一致性、安全性和稳定性。

ContainerOS 特性介绍

Introduction to ContainerOS Features

三、ContainerOS 特性介绍

操作系统作为软件与硬件之间的桥梁，一直扮演着重要的作用。尽管随着云原生相关基础设施、应用服务的蓬勃发展，操作系统的概念逐渐被弱化，但它仍然就像空气和水，在整个云原生架构中处于不可或缺的位置。ContainerOS 从一开始的设计上，便是聚焦在容器场景，旨在给容器提供稳定、安全、高效的运行环境。我们不仅从操作节点侧提供各种优化措施和流程，来简化和规范应用操作与部署的方式，更是结合 Kubernetes 控制平面，提供大规模集群下节点管理的最佳实践。

1. 专注于容器化应用

ContainerOS 默认集成 Docker、containerd、Kubernetes 等常规云原生组件，同时最小化运行环境。内核层面，云厂商的虚拟化技术使得云主机内的硬件变得非常简单，我们不需要支持过多的硬件驱动，必备的内核模块构建为 build-in 模式，大幅简化 udev 规则，云主机系统盘基本固定为 virtio-blk 或 NVME 设备，主流根文件系统也相对固定，这样便不需要 initrd 来加载 rootfs，简化内核启动流程。BaseOS 层面，仅保留容器运行所需的软件包与系统服务，剔除不必要的语言包，简化 systemd 服务，软件包数量缩减至 200 以下，相比传统操作系统减少 60%，镜像大小减少 70%。

轻量的操作系统在制作、部署和使用上会带来以下好处：

- 较小的镜像大小：一方面，减少操作系统镜像的存储空间需求，节省存储成本，另一方面，可减少镜像的下载时间，这意味着更快的部署和迁移时间。
- 更快的启动时间：精简的操作系统只加载必需的组件，以阿里云上的 ecs.g7.large（2vCPU，8G 内存规格云主机）为例，ContainerOS 的首次启动时间保持在 3s 以内。
- 更少的资源消耗：更少的系统服务就意味着更低的 CPU 和内存占用，将更多的资源释放给用户。
- 提高安全性：更少的软件包数量与系统服务意味着较少的潜在漏洞和需要修补的安全问题。

使用 ContainerOS 作为节点，在集群化管理时将会拥有更多的可用资源，相同的节点配置下，ContainerOS 可以部署更高密度的 Pod。Kubernetes 还提供了强大的自动化和扩展能力。管理员可以定义自动伸缩策略，根据负载情况自动调整 Pod 的副本数量。ContainerOS 的极速启动可以支持更激进的扩容策略，这意味着即使集群水位不足，在面对高负载业务压力时，Kubernetes 快速的自动扩展的 Pod 副本能够和节点同步横向扩容，而在负载下降时，它又能够自动缩减资源以节省成本。

2. 安全提升

许多行业都有特定的安全标准和法规要求，如 GDPR 和 HIPAA，云计算行业也不例外，因为安全问题导致的业务数据泄漏问题可能招致法律诉讼、罚款和声誉损失。与相对封闭的传统 IT 系统相比，开放的云计算平台其实面临更大的安全风险，操作系统作为基础设施的基石，其安全性就变得尤为重要。为此，我们对 ContainerOS 应用以下设计原则与流程来提升其安全性：

快速迭代发布 Pipeline

传统操作系统发行版通常具备固定的发布周期，发布流程冗长，周期短则几个月，长则半年或者一年，这导致用户获取到的操作系统版本在部署之时，可能已经有不少软件包版本落后，甚至包含一些安全漏洞。对于 ContainerOS，我们将云原生 Devops 理念引入镜像的制作发布流程中，为了适应云原生场景快速的演进节奏，最快可按天发布新的镜像，尽最大可能确保用户始终可以获取到包含最新漏洞修复的镜像。同时，针对每一个发布的镜像，使用自动化工具和流程来进行持续的安全测试，并确保及时修复发现的问题。

不可变基础设施原则

容器化技术的出现使得应用与其运行依赖被一同打包发布，业务运行所需依赖与配置均由容器镜像提供，这给不可变基础设施的实现提供了技术前提。ContainerOS 的根文件系统在此前提下被设计为只读，恶意软件或攻击者无法对其进行修改或操纵。进一步地，可选择使用 EROFS 这样的只读文件系统，将安全性再提升一个台阶。

强化访问控制与安全审计

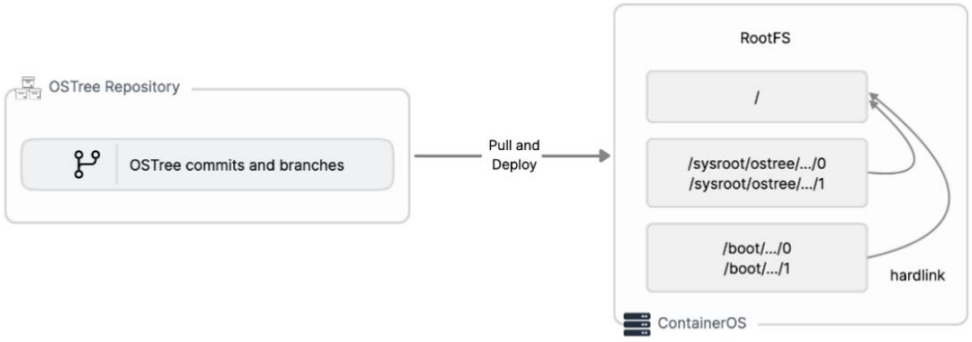
默认启用 SELinux，实施严格的访问控制策略，限制容器对敏感资源的访问权限。使用强密码和密钥管理来保护容器的访问凭证。除此之外，根据行业内的安全测评机构要求，对操作系统进行安全加固配置。这包括关闭不必要的服务、限制特权访问、启用防火墙等。

传统 IT 运维人员习惯于通过 ssh 服务登录系统进行一系列难以追溯的黑屏操作，在 ContainerOS 中，ssh 服务被默认关闭，我们推荐用户通过 API 或运维容器进行操作系统相关的操作，这样一方面可以进行操作过程的记录和审计，另一方面降低误操作带来的安全风险。启用内核 Audit 功能，记录系统中的各种动作和事件，比如系统调用，文件修改，执行程序和系统登入登出等众多与安全相关的事件，并根据记录的信息，给用户推荐相应的安全改进建议。

3. 原子升级与镜像版本化

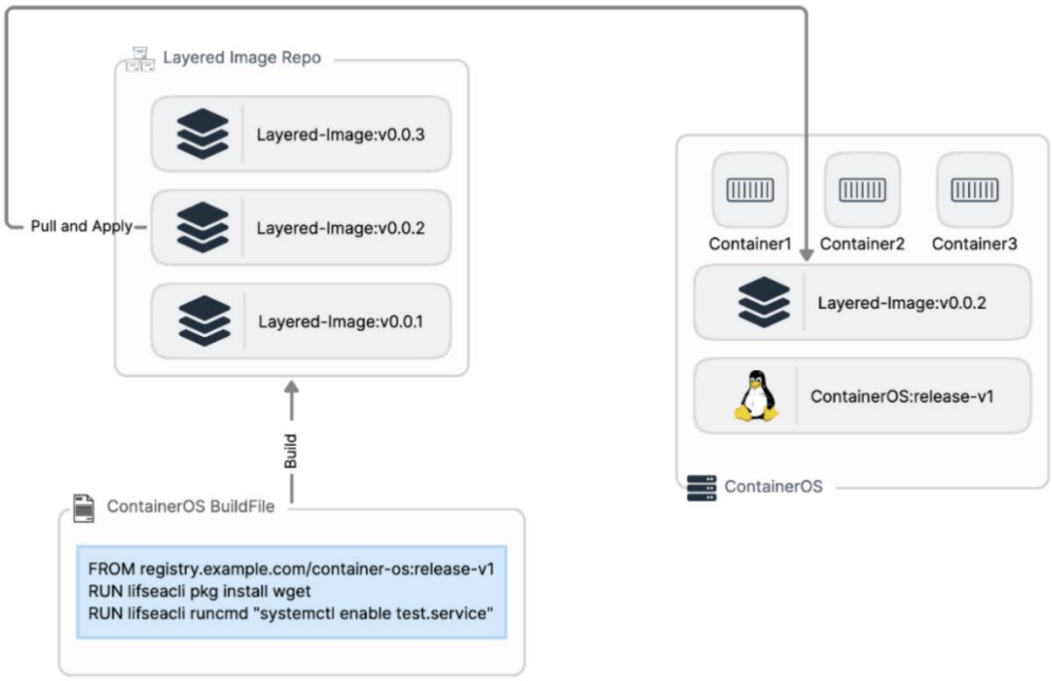
一个拥有众多节点的集群经过长时间的运行和多次的运维操作之后，集群内各个节点的状态将很难保持一致，每一台服务器就像一片独特的雪花一样独一无二，难以复制，这便是非常知名的「雪花服务器」问题。如果集群内的节点状态、配置等存在差异，即使是部署相同的应用程序，或批量下发同样的命令，也很有可能产生不同的结果，甚至，同样的命令只能在部分节点上执行成功，在其余节点上则各自产生不同的错误，当集群规模越大，问题越凸显。大规模集群环境下的节点一致性问题一直是困扰集群运维人员的关键问题之一。

传统操作系统以 rpm 为粒度进行升级，一个操作系统镜像的版本等同于众多软件包与配置的版本合集，这无疑给系统的运维带来极大的困难。相比之下，我们给 ContainerOS 提出镜像版本的概念，借助 OSTree 技术，用户可以像管理 GIT 仓库一样管理操作系统的版本，结合 Machine Operator，节点以整个镜像为粒度进行版本的轮转升级。每个镜像经过内部严格的测试之后才会上线，相较于传统操作系统基于单个 rpm 包的升级带来的不确定性，以镜像为粒度的测试发布更能保证升级后系统的稳定性。运维人员也不必再关注操作系统内部组件的变更，可将更多的精力放在业务连续性保障上。



然而，在实际的业务交付过程中，我们时常发现，部分业务场景可能会对操作系统镜像有一些特殊要求，这些要求包括但不限于修改系统配置、修改启动参数等。同一个 ContainerOS 标准镜像不一定能完全覆盖所有的业务场景。此外，不可变基础设施（只读根文件系统）的设计一定程度上使得运行时修改操作系统变得更加困难。于是，提供一种灵活易用、且能够版本化记录变更的操作系统镜像定制工具就变得尤为重要。

受到容器镜像分层技术（overlayfs）的启发，我们为 ContainerOS 提供分层变更的能力。通过 lifseacli 组件，用户可以采用类似于 Dockerfile 的方式，选择一个官方发布的基础镜像，按照 Dockerfile 的语法，进行一定程度的修改，再构建成为自己的镜像并推送到远端仓库。然后，在已经运行起来的系统中，拉取自定义的镜像层，并应用到当前的 Rootfs 中。



节点的生命周期

The Lifecycle Of A Node

四、节点的生命周期

如果说 Pod 是 Kubernetes 最小的调度单元，那么节点就是集群最小的组成单位。随着工作负载和节点不停的变化，集群始终在变化中动态平衡。从集群的层面看来，节点和节点上的工作负载并没有本质差异，只不过是被控制平面统筹管理的、具备不同能力的资源：同样具备创建、运行、销毁的生命周期，同样可能会因状态异常需要外部控制器介入协调。

有太多对 Kubernetes 的介绍只强调了 day1 的节点初始化，但事实上节点加入集群后还有漫长的变配、升级、日常运维等 day2 操作。节点的日常运维和节点的创建销毁同样重要，就像保持 Pod 的正常运行和成功创建 Pod 同样重要一样。

1. 千节点扩容的弹性

Kubernetes 集群支持以弹性伸缩的方式管理集群节点，当面临突发流量急需工作负载水平扩容时，弹性伸缩可以迅速增加节点数量，确保集群能够快速响应并保持高可用性。通过节点的弹性伸缩，集群可以在短时间内适应不断变化的需求，保证工作负载的正常运行。这种弹性能力使得业务能够轻松应对高峰期的流量，并在需要时自动缩容以降低成本。

快速的弹性伸缩能力还可以提高集群的可靠性和容错能力。当集群中的节点出现故障或不可用时，节点扩容可以快速替换故障节点，确保集群水位足以支撑工作负载的持续运行。新扩容的节点迅速填补故障节点留下的空缺，减少服务中断的风险，减少业务感知，对于关键业务和可用性要求较高的应用场景至关重要。

节点扩容重要的指标是扩容速度，一般情况下，使用通用性操作系统的单节点扩容耗时 1-5 分钟不等。但在大量的节点扩容时，扩容速度除了单节点的启动时间，更依赖集群基础服务的性能，比如 API Server、容器镜像服务等，使用传统操作系统的大规模节点扩容时间可能会飙升至十几分钟。

因此我们将千节点扩容作为重要的指标，以衡量集群的横向拓展能力。众所周知，容器的秒级启动使得大规模部署十分的便利，而当集群的千节点扩容具备 1 分钟内完成的能力时，集群的弹性也将具备极大的纵深。使得集群可以采取更激进的容量管理策略，更大密度的部署形态，更高的预警水位线。

2. 节点运维监控工具

和传统操作系统不同，只读根文件系统的设计使得 ContainerOS 仅需很少的运维干预，而面向大规模集群的设计理念，使得 ContainerOS 提供了更为简单的原子运维 API，以供外部控制器批量运维使用。

常规配置变更，包括 sysctl、kubelet、容器运行时的配置，Configuration Operator 根据声明式配置中定义的期望节点配置，自动分批下发，确保目标节点配置一致。

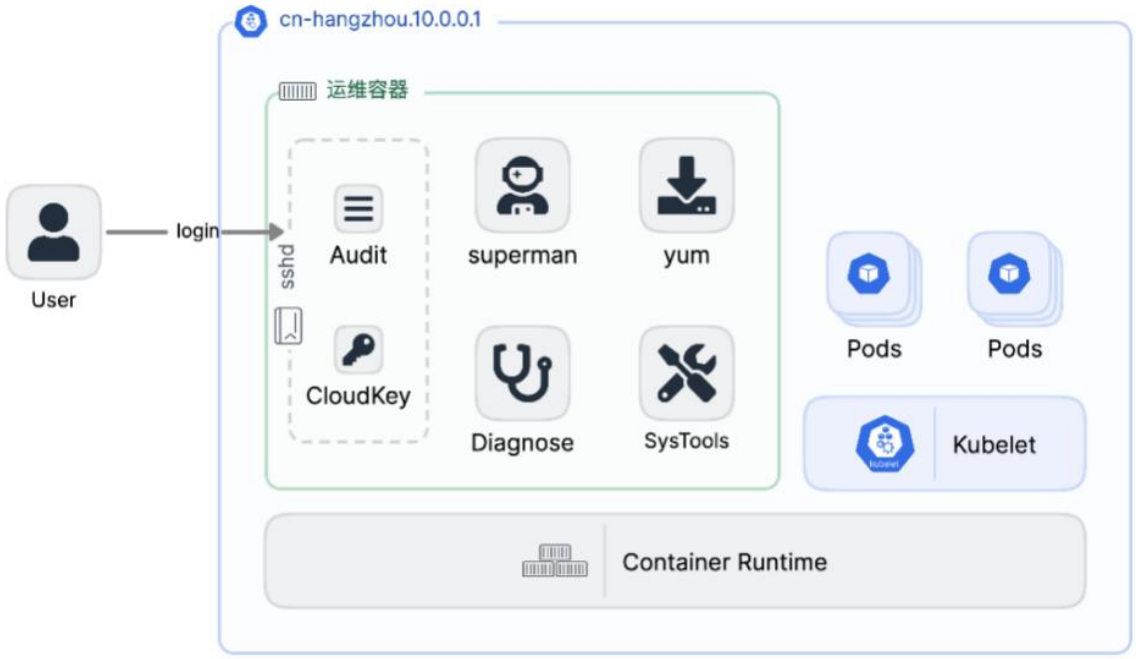
系统软件包的增减、升级操作， ContainerOS 提供了系统分层构建和更新的能力以应对增量和存量节点的更新。Machine Operator 通过统一管理，当包含新版本操作系统镜像就绪时，修改声明式配置的镜像信息。新的分层数据会自动下载到节点中，等待集群进入运维窗口后，对集群中的节点版本分批推平，仅需一次重启，便可使得节点升级到期望的操作系统版本。

1） 运维容器

单节点的登录不应该成为常规的运维手段，为了提高系统的安全性，ContainerOS 原则上不支持直接登录实例进行操作，也不提供 ssh 登录功能。但为了满足用户的非常规的运维需求，ContainerOS 提供了一种新的解决方案：运维容器。

通过启动并登录运维容器，用户可以进行系统的黑屏运维操作。相比主机，运维容器拥有更多的软件包，并且支持使用包管理软件 Yum 安装所需的软件包，以满足用户的调试需求。

在运维容器中，用户可以轻松查看系统进程信息、网络信息、系统配置等关键信息，以便进行系统的监控和管理。将用户的操作隔离在运维容器中，一定程度上得以降低误操作带来的破坏性。



2） 节点监控

节点监控也是 day2 运维中不可缺失的一环。对节点进行实时的监控和管理，有助于及时发现节点的故障或异常情况，并触发相应的预警机制。除此之外，节点监控可以提供节点的性能指标和资源利用情况，帮助管理员了解节点的负载情况、瓶颈和优化空间。ContainerOS 除了支持常规监控组件如 Prometheus、kube-state-metrics、cAdvisor 的部署之外，还提供 ECOS（Economical Cloud Native Operating System） 工具集用于内核特性的友好配置、系统异常分析报告与常规内核监控指标的透出。ECOS 工具集主要分为以下几个部分：

- ECOS Configurator：ECOS Configurator 以更偏向于用户的视角，封装 ContainerOS 内核提供的特性。通过提供一系列稳定的 API，屏蔽不同版本内核之间的配置差异，简化运维人员对内核特性的学习和使用成本。

- ECOS Analyzer：ECOS Analyzer 用于整机运行状态的分析，包括但不限于对磁盘用量、网络状态、内存压力、整机负载状态、kubelet和容器运行时健康状态等常规指标的检查，并提供异常分析诊断结果。

- ECOS Monitor：目前行业内大多数监控组件通过频繁多次调用不同的内核接口来获取

监控指标，在高密度 Pod 部署或压力负载环境下，监控组件本身将会消耗让人难以忽视的系统资源，严重时甚至影响业务 Pod 的运行。ECOS Monitor 旨在以更轻量的形式透出内核的监控指标，技术上通过 eBPF 的形式聚合内核指标。上层应用、管控链路或运维人员仅需调用 ECOS Monitor 提供的少量聚合接口就可以获得常规的监控指标数据。

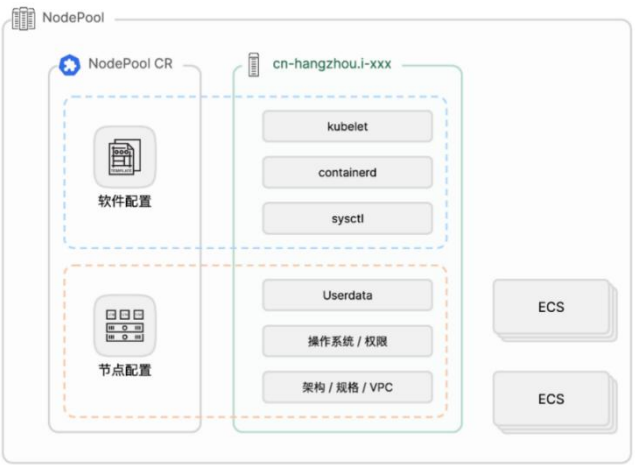
3. 节点声明式配置

声明式配置是一种以声明方式描述期望状态的配置方法，它可以简化节点配置的过程和维护的复杂性。通过定义系统的期望状态，系统可以自动向期望靠拢，保持实现与之一致的状态。

声明式配置也是 Kubernetes 的核心理念之一，它强调通过描述系统所需状态来定义所需的目标状态，而不是编写一系列命令来实现状态转换。在声明式配置中，用户通过 Yaml 文件定义期望的系统状态，然后将这些配置文件提交给 Kubernetes 控制平面。

CRD（Custom Resource Definition）是 Kubernetes 中的一种常见的自定义资源扩展机制，CRD 允许用户定义自己的 API 资源类型，将自定义资源纳入 Kubernetes 控制平面的管理范围内。

我们通过引入节点池自定义资源用于描述一组节点的期望状态，节点池是一个描述一组节点的逻辑概念，在同一个节点池中，节点具备相同的规格和架构，使用相同的基础软件版本和配置。通过定义有限个数的节点池，便可以轻松管理集群中所有的节点。



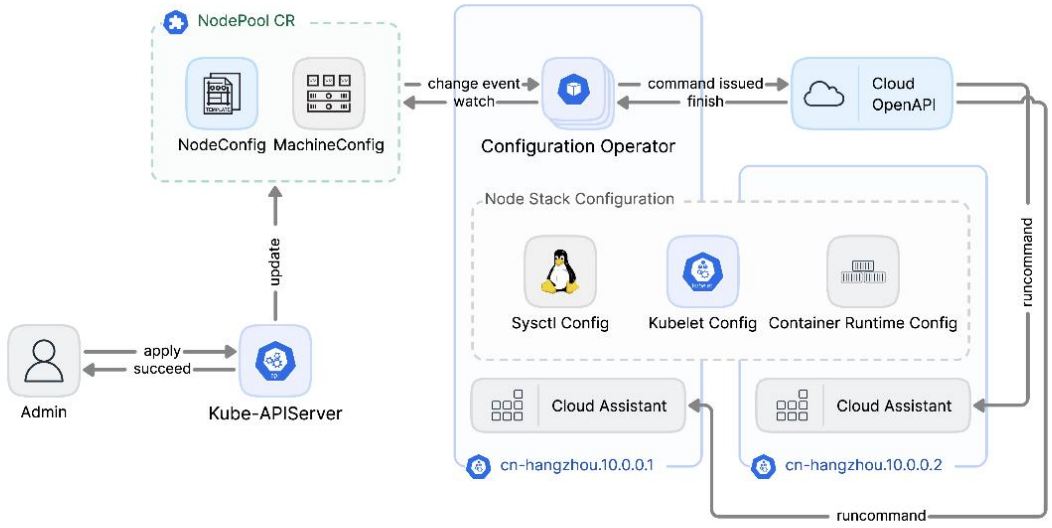
1） Configuration Operator

节点池最重要的两个期望配置项是节点基础软件配置和机器配置。对于云原生场景来说，基础软件配置包含 sysctl、kubelet 和容器运行时的配置。

在大规模集群中，软件配置的难度和复杂度在于配置一致和版本控制。在少数几个节点上执行没有问题的命令，在大规模集群却总会存在非预期的返回。而这种非预期的失败会加剧集群的不一致，导致后续的变配操作更容易因环境脏数据而失败。

对存量节点修改完成后，往往还需要注意新增节点的配置，而常规运维中增量和存量的逻辑是分开管理的，不仅容易导致误配、错配、漏配，更导致配置的版本管理失效，在不同时间创建的节点拥有不同长度的版本树。

Configuration Operator 通过简单的声明式配置，仅需要配置节点池的当前期望，一方面，Operator 会自动对当前的存量节点进行配置轮转，并对新创建的节点使用最新的期望配置。另一方面，配置管理可以直接复用节点池声明式配置的版本管理，从整个集群和节点池维度进行配置变更和版本回滚，使节点的管理不再受节点的生命周期影响。

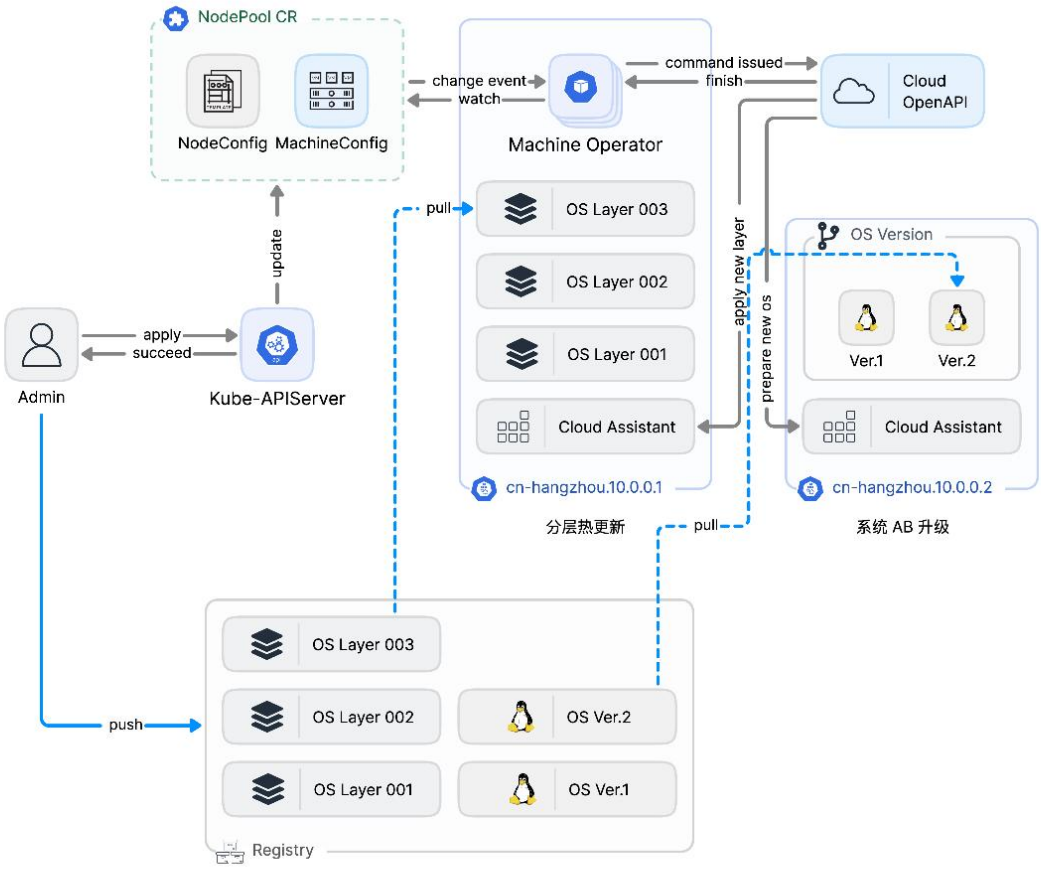


2） Machine Operator

节点机器配置主要包含节点规格、存储、网络、操作系统版本等。当需要对节点配置和系统软件栈变更时，仅需要修改对应的期望配置即可。

当需要更改节点规格、存储、网络时，Machine Operator 会立刻修改新增节点模板，保证增量节点使用新的期望。同时，发起旧节点轮转任务，通过分批轮转操作，将节点池中的节点更新为期望状态。

当需要对集群内的节点增删软件包，升级内核版本时，可以通过构建新的 ContainerOS 镜像并推送到 registry 中来实现。一旦推送完成，可以修改节点池的操作系统配置，不仅使得新的节点可以从 registry 中获取最新版本的镜像信息，同时，存量节点也会感知声明式配置的变化，根据 registry 中记录的分层元数据信息，将需要变更的分层更新到节点中。

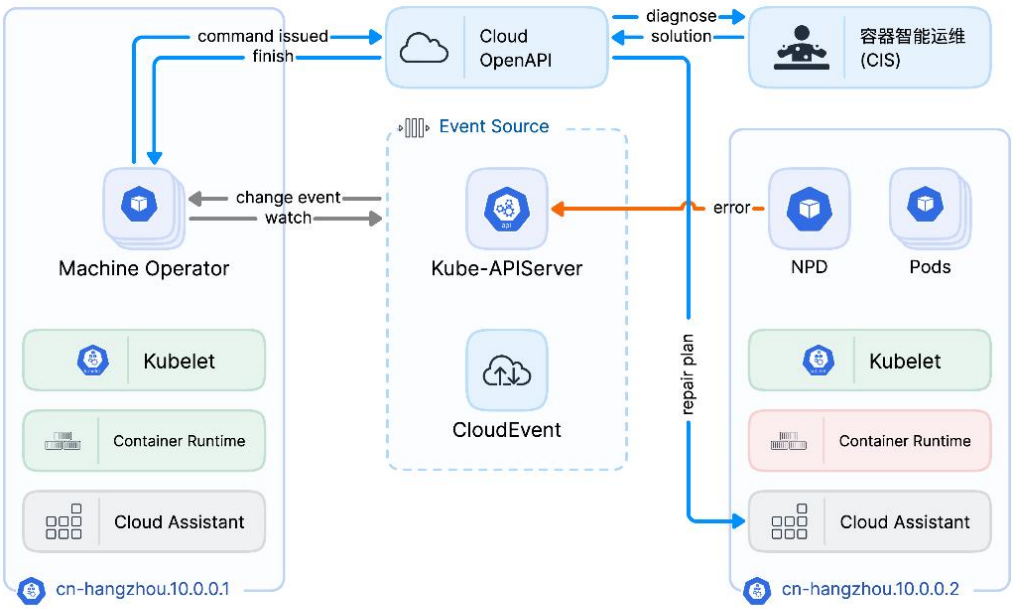


声明式配置管理带来了多个优势，不仅简化节点配置的过程和维护的复杂性。还可以避免黑屏运维带来的风险和错误。同时，声明式配置结合 ContainerOS 分层更新的能力，使得无论是存量节点还是新增节点，集群内的节点始终保持相同的配置，避免因为节点版本差异引发的非预期行为。

另外，声明式配置还提供了可追溯性和版本控制的好处，可以更好的实践 IaC 理念，简化了问题排查和回滚操作。通过声明式配置和 git ops 相结合，追溯每个节点的配置历史和变更变得十分简单，这为系统维护和故障处理提供了便利。

4. 节点故障自愈

节点故障自愈是指在集群节点池中，当节点发生异常时，自动进行节点恢复操作，以保持节点的正常运行状态。节点自愈功能包括问题诊断、恢复决策和恢复任务。节点故障自愈是通过将专家经验自动化，并根据故障特异性指标进行自动排查和故障恢复。



自动化的前提是节点处于统一、可预期的状态。对于容器场景，业务应用通过 Pod 部署在节点中，通过存储卷持久化业务数据，通过 Lifecycle Hook 完成对 Pod 的生命周期管

理，大部分节点上的配置并不会影响工作负载的行为。同时， ContainerOS 不可变的设计理念使得集群内的节点始终在可预期的状态。因此对于 ContainerOS 的节点，可以高度自动化的解决绝大部分的节点异常。

当节点发生故障时，基于事件的控制器会快速感知异常，并根据故障原因，自动执行相应恢复任务。例如，如果 Kubelet 意外停止工作、PLEG 健康检查失败或 PodSandbox 残留，通过执行相应的恢复操作，使得节点恢复正常状态，如重启 Kubelet、清理 PodSandbox、重启 ECS 实例等。

节点自愈的好处不仅可以提高集群的稳定性和可靠性。系统自动的诊断和恢复节点异常状态，还可以减少手动排查和修复故障的时间和精力成本。更及时的发现节点异常情况，并在故障放大前采取相应的恢复操作。减少故障对用户的影响，提高了系统整体的容错能力。

阿里云最佳实践

和客户案例

Alibaba Cloud Best Practices And Customer Case Studies

五、阿里云最佳实践和客户案例

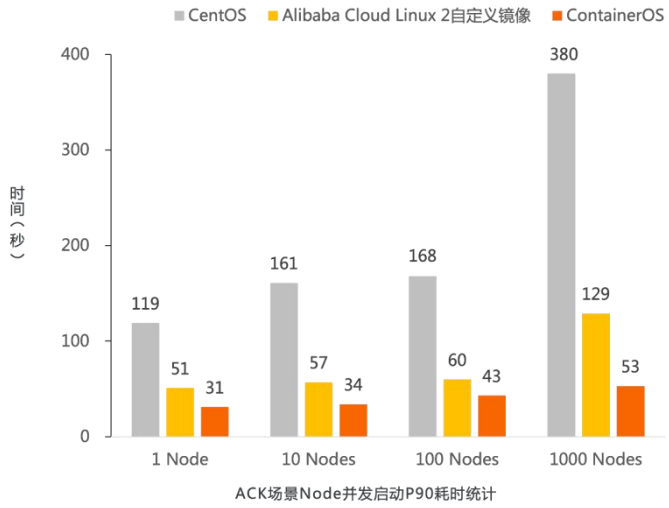
1. 在阿里云容器服务中使用 ContainerOS 实现极速扩容

在阿里云容器服务 ACK 集群中实现弹性扩容，是一项基础、关键且被很多弹性场景的用户所十分看重的能力，尤其是在应对业务突发流量时，如何快速扩容节点、恢复资源水位对业务连续性至关重要。

在节点自动伸缩场景，ACK 通过组件轮询判断集群内资源是否充足，一旦出现不足则自动触发扩容节点。如果在这种情况下，节点扩容速度慢，则会严重影响自动伸缩效果，甚至因为资源水位长期不足而影响用户业务。目前的节点扩容速度在 ACK 节点自动伸缩端到端耗时中占比超过 90%，可以说节点扩容速度的优化的程度决定了自动伸缩的体验。

以某量化公司的扩容场景为例，在其长期提供服务的过程中，经过了上千次百节点级别的扩容活动，平均每次扩容 P90 节点就绪耗时（即单次扩容活动开始至 90% 的节点处于就绪状态）超过 2min，耗时较长，且受网络抖动干扰大，时常出现部分节点就绪时间超长。若能将节点就绪时间稳定收缩在一个范围内，将大幅提高用户在节点扩容场景中的体验。

为了达到上述效果，ContainerOS 基于 ACK 弹性扩容场景进行了端到端优化。通过预置集群管控必备组件的容器镜像以减少节点启动过程中因镜像拉取而带来的耗时，并结合 ACK 管控链路优化（例如调节关键逻辑的检测频率、调整高负载下系统瓶颈中的限流值等），成功将节点扩容时间稳定控制在 1min 以内。



ACK场景Node并发启动P90耗时统计

接下来介绍如何在 ACK 中配置使用 ContainerOS 实现节点极速扩容。

1） 前置条件

- 已在 ACK Pro 创建 Kubernetes 集群，容器运行时为 Containerd，且 Kubernetes 版本为 1.24.6 及以上，节点池类型为托管节点池。
- 集群使用默认的网络插件（Terway）与存储插件（CSI）。

2） 操作步骤

创建 ContainerOS 的节点池

- 登录容器服务管理控制台，在左侧导航栏选择集群。
- 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择节点管理 > 节点池。
- 在节点池页面右上角，单击创建托管节点池。
- 在创建托管节点池对话框，配置操作系统为 ContainerOS 类型，例如 ContainerOS 1.26.3，并按需配置其他选项，然后单击确认配置。

关键组件限流调整

如果您有同时启动大量节点（超过 100 个节点）的业务场景，建议进一步手动配置以下几个优化项以达到更好的弹性效果。部分 API 默认支持的最大连接数为 100，因此同时启动少于 100 个 ECS 节点时无需额外配置。

- KCM（Kube Controller Manager）限流调整
  - 登录容器服务管理控制台，在左侧导航栏选择集群。
  - 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择运维管理 > 组件管理。
  - 在组件管理页面的核心组件页签，定位到 Kube Controller Manager，然后单击卡片右下方的配置。
  - 在参数配置对话框，配置 kubeAPIQPS 为 800、kubeAPIBurst 为 1000，其余选项按需配置，然后单击确定。

说明：基于测试数据，推荐您按照上方数值进行配置。如有其他需求，您也可以按照自身业务场景灵活配置。

- Kube Scheduler 限流调整
  - 登录容器服务管理控制台，在左侧导航栏选择集群。
  - 在集群列表页面，单击目标集群名称，然后在左侧导航栏，选择运维管理 > 组件管理。
  - 在组件管理页面的核心组件页签，定位到 Kube Scheduler，然后单击卡片右下方的配置。
  - 在参数配置对话框，配置 connectionQPS 为 800、connectionBurst 为 1000，其余选项按需配置，然后单击确定。

说明：基于测试数据，推荐您按照上方数值进行配置。如有其他需求，您也可以按照自身业务场景灵活配置。

- APIServer 数量调整

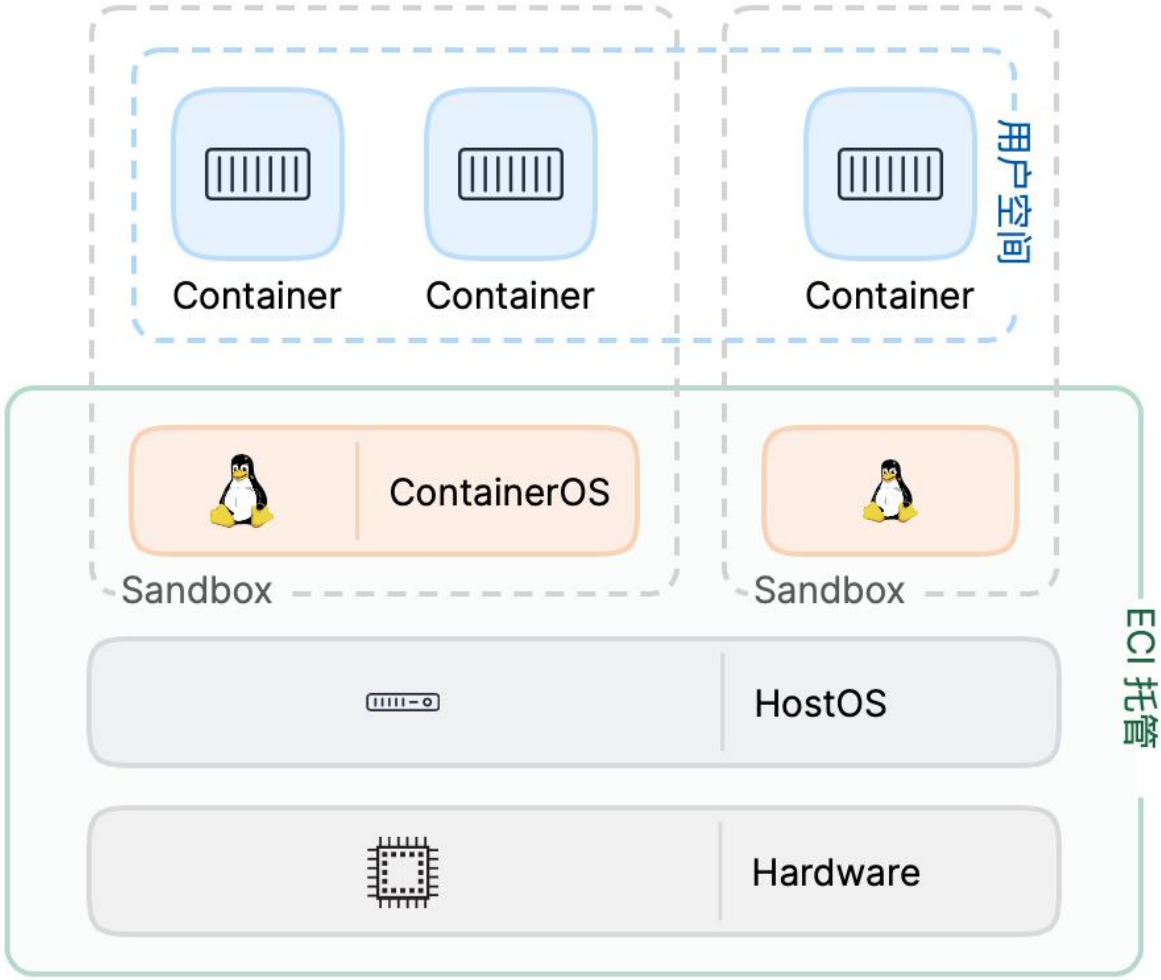
集群内 APIServer 的副本数量根据负载进行弹性伸缩。如果同一时间弹出节点较多，APIServer 会自动进行扩容，一定程度上增加点就绪的耗时。若追求极致扩容时间，您可以提交工单，提前调整 APIServer 的副本数量，优化扩容效果。

2. ContainerOS 助力阿里云 ECI 极致弹性

阿里云弹性容器实例 ECI（Elastic Container Instance）是一款基于轻量级安全沙箱，面向 Serverless 场景的云产品。用户无需管理底层服务器，也无需关心运行过程中的容量规划，只需要提供打包好的容器镜像，即可在云上快速、安全地部署自己的应用，并仅为容器实际运行消耗的资源付费。

随着云原生进入下半场，业界对容器启动速度、资源消耗、稳定性的要求越来越高，而这些也是 ECI 相对普通容器会面临的挑战。在 ECI 中，每个 Pod 之间都是 VM 级别的虚拟化安全隔离（安全沙箱），从安全沙箱创建、调度，到计算、存储、网络资源的初始化，再到应用启动，流程非常长。倘若安全沙箱使用传统操作系统，则单纯安全沙箱自身启动时间就可达分钟级，根本无法适应 ECI 所面临的大规模、突发流量的场景。

除此之外，由于安全沙箱间内核隔离，过大的操作系统本身也会占用额外系统资源，无法达到机器高密度部署要求。因此，轻量、秒级启动的 ContainerOS 成为 ECI 安全沙箱 OS（GuestOS）的不二之选。



使用 ContaienrOS 作为 GuestOS 的 ECI 可以在 6 秒之内弹出 3000 个容器实例，成功支撑了弹性容器实例 ECI 业务每日最高超百万的创建量，通过极致的高密和弹性表现大幅增加业务的核心竞争力。

不仅如此，ContaienrOS 除了提供轻量、极速的运行环境，还提供了快速迭代发布 Pipeline，通过灵活和标准化的制作每个安全沙箱镜像，极大程度上降低 ECI 管控人员在镜像制作维护上的成本。镜像的制作、测试、发布周期可缩短至数小时，足以应对瞬息万变的云原生用户需求。

3. 蚂蚁安全科技镜像加速实践

ZOLOZ 是蚂蚁集团旗下的全球安全风控平台，通过业内领先的生物识别、大数据分析和人工智能技术，为用户和机构提供安全又便捷的安全风控解决方案。ZOLOZ 已为中国、印尼、

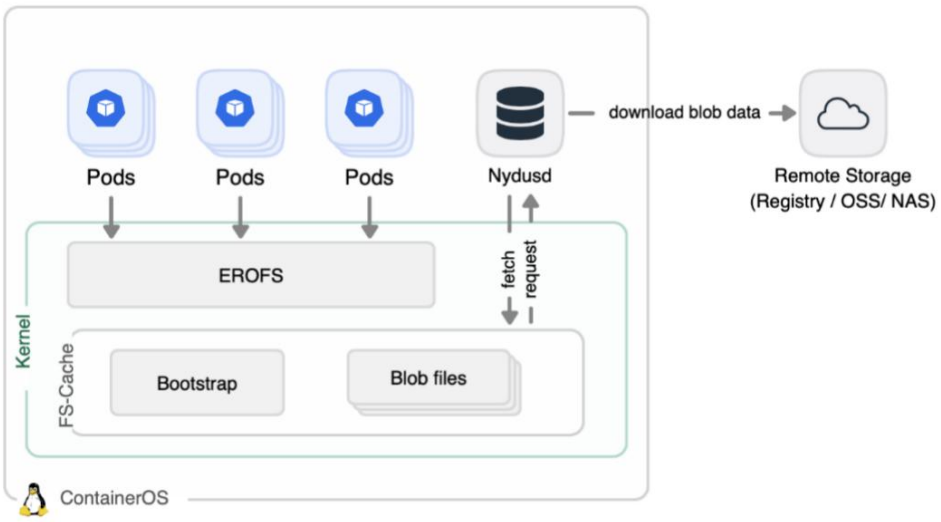
马来西亚、菲律宾等 14 个国家和地区的 70 余家合作伙伴提供数字化转型过程中的安全风控技术支持。目前，已经覆盖金融、保险、证券、信贷、电信、公众服务等领域，累计服务用户超 12 亿。

随着 Kubernetes 和云原生的大爆发，ZOLOZ 应用开始在公有云上进行大规模容器化部署。在公有云上容器化持续推进的当下，ZOLOZ 应用主要遇到如下挑战：

- 集群机器拉起时间长，难以满足流量突增时，弹性自动扩缩容。
- 拉取算法镜像耗时长，在集群扩容大量机器拉取镜像文件会容易导致集群网卡被打满，影响业务正常运行。

针对 ZOLOZ 遇到的实际问题，ContainerOS 结合行业内多种相关解决方案，通过整合 Nydus RAFS （Registry Acceleration File System）能力，支持用户构建 Nydus 格式的镜像。

一方面， ContainerOS 通过简化 OS 启动流程，预置集群管控必备组件的容器镜像以减少节点启动过程中因镜像拉取而带来的耗时，极大地提高了 OS 启动速度，降低了节点扩容时间。另一方面，依托于 Nydus 与内核 EROFS 的优势，用户容器镜像得以进行块级别数据去重，大大降低了镜像的上传和下载数据量，Nydus 镜像提供按需加载能力，容器启动时仅需拉取少部分启动必需的数据，后续容器内业务 IO 请求哪些文件的数据，再从远端 Registry 拉取这些数据，这样避免镜像大量数据拉取阻塞容器的启动，大幅提升容器内业务就绪的时间。



ContainerOS 通过提供标准化、全流程的解决方案，以及更高的安全性和免运维的特点，给 ZOLOZ 整体的业务部署、研发效率、线上稳定性带来了质的飞跃。

尾声

Conclusion

六、尾声

1. 云原生节点管理的基本逻辑

由于云原生技术的指数级增长，越来越多的业务和应用会运行在 Kubernetes 中，Gartner 曾预测，2027 全球范围内将有 90% 的应用以容器形态部署。

而大规模的集群运维使得传统的集中式节点管理方式已经无法满足高效、灵活且可自动化的要求。云原生节点管理的核心目标，是基于云原生的理念，利用针对容器场景进行优化的操作系统和配套基础设施，实现自动化运维，以降低节点管理成本并提升生产效率。

从节点维度上，ContainerOS 作为容器场景优化的操作系统，通过针对性剪裁，不仅减少系统资源的占用，提高节点利用率和安全性。也通过提供系统运维 API 和分层构建、更新的能力，使得节点运维和变更变得更加高效。

从集群维度上，实现自动化运维是降低节点管理成本的关键。自动化运维能够减少运维人员的工作量，依托于操作系统提供的原子 API 和声明式配置，可以快速、准确地配置和部署节点，避免了手动操作可能引发的错误和延误。而节点的一致性和自动化使得自动诊断、节点故障自愈成为可能，这些优化措施共同提高了管理效率、降低人力成本，较少的人力投入足以维护规模庞大集群的正常运行。

2. 未来节点管理的发展趋势

节点管理领域仍然具有广阔的发展前景。随着人工智能、大数据和边缘计算等新技术的发展和应用，节点管理将面临更多的挑战和机遇。

一方面。这些新的技术和业务场景会对 Kubernetes 的使用和运维提出更高的要求，另一方面，这些新技术的发展也会反哺 Kubernetes 等基础设施，为节点管理带来更多的智能化和自动化的可能性。

同时，随着 DevSecOps 的兴起，越来越多的人意识到容器和节点安全的割裂局面，节点管理还将面临更多的安全威胁和风险。未来也将更加注重节点安全和容器安全的统筹管理，包括容器安全加固、自动化安全性测试、审计和合规加固等措施。

云原生节点管理作为一种全新的节点管理范式，是根据当前 Kubernetes 集群的使用局限性提出的系统化和工程化的解决方案。随着技术的不断前进和创新，云原生节点管理始终以云原生理念为指导，以具体的场景和痛点为抓手，面向未来的发展，迎接更广阔的机遇和挑战。

白皮书作者

（以下排名不分先后）

彭媛洪　　陈海波

扫码加入

龙蜥云原生钉钉交流群

畅聊前沿技术

微信扫码关注

龙蜥社区公众号

阿里云开发者“藏经阁”

海量电子手册免费下载