可扩展性与性能

概览

使用可扩展性和性能来规划集群规模、资源分配以及对性能敏感的工作负载放置。此部分汇总了规模评估、磁盘配置、资源管理器策略和大型集群稳定性指导。

需求	继续阅读
评估业务集群规模。	评估业务集群
评估 `global` 集群规模。	评估 global 集群
配置 CPU、内存和 topology manager 策略。	资源管理器策略
查看大型集群稳定性指导。	大型集群稳定性
配置与磁盘相关的性能设置。	磁盘配置
规划 infra 节点或自定义角色节点。	节点规划

如需了解经过验证的规模上限、容量规划公式或生产环境加固要求，请参考当前支持矩阵和特定主题的指导。

评估业务集群资源

目录

推荐的控制平面实践

使用这些实践来规划 ACP 中业务集群的控制平面资源。

> **INFO**
>
> 以下要求基于安装了 Core packages 的最小 ACP 配置。安装 Extensions 后，实际需求可能更高。有关额外资源需求，请阅读特定于 Extension 的指导。

控制平面节点规格调整

控制平面规格需求会随着集群的节点数量、节点类型，以及运行的对象数量和类型而变化。以下建议源自 Cluster-density 测试，这些测试用于评估控制平面在负载下的行为。测试会在指定的一组命名空间中部署以下对象：

- 6 个 deployment，其中的 pod 只运行 sleep 进程
- 6 个 service
- 6 个指向前述 service 的 ingress
- 12 个包含 2048 个随机字符串字符的 secret
- 12 个包含 2048 个随机字符串字符的 config map

工作节点数量	**Cluster-density**（命名空间）	**CPU** 核数	内存（**GB**）
24	500	4	16
120	1000	8	32
254	4000	24	128

上表数据基于一个安装了 ACP 的公有云虚拟机环境，使用 16 vCPU 和 128 GB RAM 的控制平面节点，以及 8 vCPU 和 32 GB RAM 的工作节点。

在拥有三个控制平面节点的大型密集集群中，如果有一个节点离线——无论是由于电源或网络故障等意外问题、基础设施问题，还是为了节省成本而有意关闭——其余两个节点都必须处理额外的工作负载。此时，幸存的控制平面机器上的 CPU 和内存使用率可能会显著上升。
在升级期间，你也会看到这种模式，因为控制平面节点通常会在控制平面 Operator 更新时依次被 cordon、drain 和重启。这样的顺序维护会将需求集中到仍处于活动状态的节点上。
为降低级联故障的风险，请为控制平面服务器预留充足的余量：目标是将整体 CPU 和内存使用率控制在约 60% 或更低，以便集群能够吸收瞬时负载增加。如有必要，可增加控制平面的 CPU 和 RAM，以防止因资源耗尽而导致潜在故障。

IMPORTANT NOTES

节点规格会因集群中的节点数量和对象数量而异。它还取决于对象是否正在集群中主动创建。在对象创建期间，与对象处于 **Running** 阶段相比，控制平面的资源使用会更活跃。

ACP 主要版本测试的集群最大值

WARNING

对节点物理资源进行过度分配会削弱 Kubernetes 依赖的调度保证。请应用资源请求和限制、QoS 类以及节点调优等控制措施，以尽量减少内存交换和其他资源争用。

这些数据来自 灵雀云 的测试配置、方法和调优。下面列出的不是固定上限，而是在测试条件下 ACP 观察到的最大值。 由于 ACP 版本、控制平面工作负载和网络插件的组合没有上限，因此这些值并不保证适用于所有部署，也不一定能在所有维度上同时达到。
在规划具有相似特征的部署时，请将其作为参考。\

INFO

在增加或减少 ACP 业务集群中的节点数量时，建议：

- 在可用时，将节点分布到所有可用区，这有助于提高可用性。
- 每次扩容/缩容不超过 25 到 50 个节点。

在对大型、密集部署的集群进行缩容时，该操作可能需要相当长的时间，因为必须先迁移或终止计划移除节点上的工作负载，然后才能关闭这些节点。当一次需要处理大量资源时，这种情况尤其耗时。如果需要逐出的大量对象过多，API 客户端可能会开始对请求进行限流。默认的 client queries-per-second（QPS）和 burst 设置分别为 **50** 和 **100**，并且这些值无法在 ACP 中更改。\

最大类型	测试最大值
节点数量	500 [1]
pod 数量 [2]	100,000
每个节点的 pod 数量	250
命名空间数量 [3]	10,000
每个命名空间的 pod 数量 [4]	25,000
secret 数量	40,000
config map 数量	40,000
service 数量	10,000

最大类型	测试最大值
每个命名空间的 service 数量	5,000
每个 service 的后端数量	5,000
每个命名空间的 deployment 数量 [4]	2,000
自定义资源定义（CRD）数量	1,024 [5]

1. ACP 支持超过 500 个节点的集群；此处的数字是建议的最大值。如果你的用例需要更大的集群，请联系技术支持。
2. 所示 pod 数量来自测试环境。实际 pod 容量会因每个原生应用的内存、CPU 和存储需求而异。
3. 在有许多活跃项目的情况下，如果 keyspace 增长过大并超过其空间配额，etcd 性能可能会下降。建议定期对 etcd 进行维护，例如执行碎片整理，以回收存储并避免性能问题。
4. 一些控制循环会在状态变更时遍历命名空间中的所有对象。单个命名空间中某一类型对象数量过多会使这些循环开销很大，并可能降低处理速度。所述限制假设系统具有足够的 CPU、内存和磁盘来满足应用需求。
5. 该测试运行在一个 29 台服务器的集群上（3 个控制平面节点、2 个基础设施节点和 24 个工作节点），并包含 500 个命名空间。ACP 强制执行 1,024 个 CRD 的总上限，包括 ACP 提供的 CRD、集成产品添加的 CRD 以及用户创建的 CRD。创建超过 1,024 个 CRD 可能会导致 **kubectl** 请求被限流。

示例

在一个测试场景中，500 个工作节点（每个节点配备 8 vCPU 和 32 GB 内存）与 ACP 4.1、Kube-OVN 网络插件以及以下工作负载对象一起进行了测试：

- 200 个命名空间，除此之外还有默认命名空间
- 每个节点 60 个 pod；30 个 server pod 和 30 个 client pod（总计 30k）
- 每个 ns 15 个由 server pod 提供后端的 service（总计 3k）
- 每个 ns 20 个 secret（总计 4k）
- 每个 ns 10 个 config map（总计 2k）
- 每个 ns 6 个 network policy，包括 deny-all、allow-from ingress 以及命名空间内规则

以下因素已知会影响集群工作负载扩展，应在扩缩容规划中加以考虑。如需更多指导，请联系技术支持。

- 每个节点的 pod 数量
- 每个 pod 中的容器数量
- 所使用的探针类型（例如 liveness/readiness、exec/http）
- network policy 数量
- 项目或命名空间数量
- service/endpoints 的数量和类型
- shard 数量
- secret 数量
- config map 数量
- API 调用速率，即对集群配置变化速度的估算。
  - 用于统计 5 分钟窗口内 pod 创建请求每秒数的 Prometheus 查询：
    `sum(irate(apiserver_request_count{resource="pods",verb="POST"}[5m]))`
  - 用于统计 5 分钟窗口内所有 API 请求每秒数的 Prometheus 查询：
    `sum(irate(apiserver_request_count{}[5m]))`
- 集群节点的 CPU 资源消耗
- 集群节点的内存资源消耗

评估 global 集群的资源

目录

概述

在规划用于多集群 ACP 部署的 `global` 集群时，请使用这些资源评估实践。

合理的节点规格有助于 `global` 集群管理已注册集群、处理同步流量，并在符合环境性能预期的情况下处理用户 API 和 Web Console 请求。

节点规格

`global` 集群负责：

- 维护集群注册和元数据。
- 处理来自 Web Console 和 CLI 的入站 API 请求。
- 协调与已连接集群之间的同步和心跳消息。
- 管理内部控制器和资源调谐循环。

由于 `global` 集群既要处理管理操作，又要聚合所有已连接集群的数据，因此请根据预期规模和工作负载强度来规划资源分配。

生产环境基准规格

生产规模的规格主要取决于以下因素：

- 受管集群数量
- 同步周期频率
- 并发 **API** 请求率（来自用户或自动化）
- 流式请求数量
- 已安装插件数量

下表提供了经过内部性能测试验证的参考配置。

规模等级	受管集群	节点数	每节点 **CPU**	每节点内存	备
小型	≤ 10	3	8 cores	16 GB	适 环
中型	≤ 50	3	16 cores	32 GB	默 配
大型	≤ 100	3	24 cores	48 GB	支 W 使 步
超大型	≤ 500	6	32 cores	64 GB	需 和 施

WARNING

这些建议仅为通用指导。实际需求取决于集群拓扑、用户并发以及已安装的插件。

垂直扩展指南

当单节点负载增加时（例如，集群数量增加 2 倍或用户并发更高），请按以下方式调整：

参数	扩展建议
CPU	每增加 50 个受管集群，增加 50%
内存	每增加 50 个受管集群，增加 50%

水平扩展指南

当受管集群数量超过 100 个，或持续出现超过 500 ms 的 API 延迟时：

添加节点，以分散请求处理和控制器工作负载。

资源验证与监控

部署后，请持续监控以下指标以验证节点规格：

指标	建议范围
节点 CPU 利用率	峰值负载下 60–75%
节点内存利用率	持续 ≤80%
API 请求延迟	P90 < 500ms
etcd 提交延迟	P99 < 50ms

Node CPU utilization

```
100 * (1 - avg by (instance)(rate(node_cpu_seconds_total{mode="idle"}
[5m])))
```

Node Memory utilization

```
100 * (1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_byt
es))
```

API request latency

```
histogram_quantile(
  0.9,
  sum(rate(apiserver_request_duration_seconds_bucket{verb!="WATCH",re
source!="events"}[5m])) by (le)
)
```

etcd commit latency

```
histogram_quantile(
  0.99,
  sum(rate(etcd_disk_backend_commit_duration_seconds_bucket[5m])) by
(le)
)
```

NOTE

如果持续资源使用率始终超过建议阈值，请在面向用户的性能下降发生之前进行垂直扩展（增加 CPU/内存）或水平扩展（添加节点）。

总结

为 `global` 集群进行规格规划时：

1. 对于中等规模部署（≤50 个集群），从 **3** 个节点 **× 16 cores × 32 GB** 开始。
2. 当请求并发更高或 Web Console 使用强度更大时，进行垂直扩展。
3. 超过 100 个集群后进行水平扩展，以保持 API 响应能力。
4. 每当受管集群数量或同步频率发生显著增长后，重新评估规格。

使用这些实践可使资源规划与 Multi-Cluster 环境的增长保持一致。

使用 Manager Policies 优化 Pod 性能

Kubernetes 集群管理员可以启用并验证 **CPU ManagerPolicy**、**Memory ManagerPolicy** 和 **Topology ManagerPolicy**，以便为低延迟工作负载协调 CPU 绑定、NUMA 亲和性和拓扑放置。

目录

适用范围和前提条件

角色和权限

- 需要维护窗口访问权限、`kubectl` 管理员权限，以及对节点的 SSH 访问权限。

工作负载要求

- 为了实现专用 CPU 和 NUMA 亲和性，Pod 必须运行在 **Guaranteed QoS** 类中：requests = limits，并且 CPU 以完整核心指定（例如 2、4）。

不在范围内

- HugePages 不在范围内。如果你需要 HugePages 支持，请联系你的支持团队。

快速开始：示例 **Kubelet** 配置

以下 kubelet 配置启用了感知 NUMA 的调度。请根据你的环境调整这些值。具体应用方式取决于节点操作系统——请参见应用配置。

```
# ── CPU ManagerPolicy ──
cpuManagerPolicy: "static"              # Options: none | static
cpuManagerPolicyOptions:
  full-pcpus-only: "true"               # Recommended: allocate only full
cores
cpuManagerReconcilePeriod: "5s"
reservedSystemCPUs: ""                  # e.g. "0-1" if reserving specifi
c CPUs for the system

# ── Memory ManagerPolicy ──
memoryManagerPolicy: "Static"           # Options: none | Static
reservedMemory:
  - numaNode: 0
    limits:
      memory: "2048Mi"
  - numaNode: 1
    limits:
      memory: "2048Mi"

# ── Topology ManagerPolicy ──
topologyManagerPolicy: "single-numa-node"     # Options: none | best-effo
rt | restricted | single-numa-node
topologyManagerScope: "pod"                   # Options: container | pod
```

说明：

- `full-pcpus-only: "true"` 可提升延迟一致性。
- `topologyManagerScope: pod` 可确保同一 Pod 中的容器对齐到共同的 NUMA 拓扑。
- `reservedMemory` 必须基于 kubelet 配置和驱逐阈值进行计算（见下一节）。

如何计算 `reservedMemory`

公式：

```
R_total = kubeReserved(memory) + systemReserved(memory) + evictionHard(memory.available)
```

所有 NUMA 节点上的 `reservedMemory` 之和必须等于 `R_total`。

步骤（针对 **N** 个 **NUMA** 节点）：

1. 计算 `R_total`（Mi）。
2. 计算商和余数：
   - base = floor(R_total / N)
   - rem = R_total − base × N
3. 分配数值：
   - NUMA 节点 0 = base + rem
   - 其余 NUMA 节点 = base

示例（**2** 个 **NUMA** 节点）：

- kubeReserved=512Mi, systemReserved=512Mi, evictionHard=100Mi → R_total = 1124Mi
- base = 562, rem = 0

```
reservedMemory:
- numaNode: 0
  limits:
    memory: "562Mi"
- numaNode: 1
  limits:
    memory: "562Mi"
```

应用配置

如何应用这些 kubelet 设置取决于节点操作系统。

不可变基础设施

在不可变基础设施上，CPU 和 Memory Manager 策略会在 Alauda 技术支持的协助下应用。请联系你的 Alauda 支持代表，以在不可变节点上启用这些策略。

传统操作系统

在传统操作系统上，请在每个节点上应用配置：

1. 封锁并驱逐

```
kubectl cordon <node>
kubectl drain <node> --ignore-daemonsets --delete-emptydir-data
```

2. 停止 **Kubelet** 并清理状态

```
sudo systemctl stop kubelet
sudo rm -f /var/lib/kubelet/cpu_manager_state
sudo rm -f /var/lib/kubelet/memory_manager_state
```

3. 重启 **Kubelet**

```
sudo systemctl daemon-reload
sudo systemctl start kubelet
```

4. 重新调度 **Pod**

```
kubectl uncordon <node>
```

- 对于 DaemonSets 和 system Pod，请显式重启或删除 Pod。

5. 验证恢复

```
kubectl get nodes
kubectl get pods -A -o wide | grep <node>
```

验证

CPU ManagerPolicy 状态

```
sudo cat /var/lib/kubelet/cpu_manager_state | jq .
```

检查：

- `.policyName` = `"static"`
- `.defaultCpuSet` 列出非独占 CPU
- `.entries` 显示容器到 CPU 的分配

Memory ManagerPolicy 状态

```
sudo cat /var/lib/kubelet/memory_manager_state | jq .
```

检查：

- `.policyName` = `"Static"`
- reserved memory 的总和与 `R_total` 一致

- Guaranteed Pod 按照 `single-numa-node` 策略分配到 NUMA 节点

关键策略和行为

CPU ManagerPolicy

- 目的：为 Guaranteed Pod 分配独占物理 CPU
- 配置：`cpuManagerPolicy: static`，`full-pcpus-only: "true"`
- 行为：仅适用于 Guaranteed Pod；Burstable/BestEffort 不受影响

Memory ManagerPolicy

- 目的：在 NUMA 节点级别预留并对齐内存
- 配置：`memoryManagerPolicy: "Static"`，`reservedMemory`
- 行为：与 Topology ManagerPolicy 配合时，能更好地实现对齐

Topology ManagerPolicy

- 目的：将 CPU、内存和设备分配对齐到单个 NUMA 节点
- 配置：`topologyManagerPolicy: single-numa-node`，`topologyManagerScope: pod`
- 模式：best-effort、restricted、single-numa-node（严格）

术语

- **NUMA** 节点：非一致性内存访问域
- **CPU pinning**：将容器绑定到专用 CPU
- **NUMA affinity**：优先使用与 CPU 相同 NUMA 节点上的内存
- **Topology alignment**：将 CPU、内存和设备共置于一个 NUMA 节点上
- **Guaranteed Pod**：requests = limits；CPU 以完整核心指定

提升大规模集群的 Kubernetes 稳定性

集群运维人员和 SRE 可以使用这些实践来降低控制平面过载、提升可靠性，并限制大规模 Kubernetes 集群中的故障影响范围。

目录

说明

- 对于网络、存储、负载均衡器、日志和监控调优，请使用针对这些能力的组件专用指导。

WARNING

- 在生产环境之前测试配置更改。
- 当风险较高时，避免使用单个超大集群；可以考虑多集群管理以缩小故障影响范围。

问题	描述	优化
etcd 磁盘	在大规模集群中，etcd 对磁盘 I/O 非常敏感。	为 etcd 数据目录使用专用的 NVMe SSD。

问题	描述	优化
etcd DB 大小	大于 ~8 GB 的数据库会显著增加延迟、资源使用和恢复时间。	将 etcd DB 保持在 ≤ 8 GB；删除未使用的对象，并保持频繁更新的对象较小（~≤100 KB）。
etcd 键抖动	键的高读写频率会给 etcd 带来压力。	分析 etcd 指标以查找并减少热点键。
etcd 中每种资源类型的数据大小	每种资源类型的总量过大会使全量列表操作代价高昂，并可能阻塞控制器。	将每种资源类型的总数据量保持在 ≤ 800 MB；清理未使用的 Deployment/Service。[1]
API server LB 带宽和连接数	负载均衡器的带宽或连接数限制可能导致节点变为 NotReady。	提前监控/预配 API server 的负载均衡器。
每个 namespace 中的 Service 数量	过多的 Service 会向 Pod 注入大量环境变量，并减慢启动速度。	将每个 namespace 中的 Service 数量控制在 < 5,000，或设置 `enableServiceLinks: false`。[2]
集群中的 Service 总数	过多的 Service 会增加 kube-proxy 规则并影响性能。	将 Service 总数控制在 < 10,000，并清理未使用的 Service。[1]
CoreDNS	大量 Pod 可能会降低 CoreDNS 性能。	运行 NodeLocal DNSCache (nodelocaldns)。
Pod 更新速率	过高的更新速率会将 Endpoint/EndpointSlice 变更推送到所有节点，并可能引发风暴。	降低 Pod 抖动；对于 RollingUpdate，设置保守的 `maxUnavailable` / `maxSurge`。
ServiceAccount token 挂载	每个 token Secret 都可能创建一个 watch；大量 watch 会加重控制平面负担。	对于不需要 API 访问的 Pod，设置 `automountServiceAccountToken: false`。[3]
对象数量/大小	累积的 ConfigMaps、Secrets、PVCs 等会增加控制平面负载。	限制 ReplicaSet 历史记录（`revisionHistoryLimit`），并为 Jobs/CronJobs 使用 `ttlSecondsAfterFinished`。

问题	描述	优化
Pod requests/limits	requests 与 limits 之间的差距过大，在节点丢失时可能触发级联故障。	在可行的情况下，优先将 requests 设置为等于 limits。
控制器重启与监控	控制器或 API server 重启会触发重新列表，可能使 API server 过载。	监控控制器，设置合适的资源以避免重启，并减少不必要的控制平面操作。

Footnotes

1. https://github.com/kubernetes/community/blob/master/sig-scalability/configs-and-limits/thresholds.md ↗ ↩ ↩[2]
2. https://kubernetes.io/docs/tutorials/services/connect-applications-service/#accessing-the-service ↗ ↩
3. https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/#opt-out-of-api-credential-automounting ↗ ↩

磁盘配置

目录

存储容量

INFO

请将以下分区挂载到独立磁盘或由 LVM 提供的逻辑卷上，以便后续扩容。

partition	Minimum size	Recommended size	Notes
/var/lib/etcd	10GB	20GB	建议为存放 etcd 数据使用专用的高 I/O 磁盘。
/var/lib/containerd/	100GB	150GB	

partition	Minimum size	Recommended size	Notes
/cpaas/	对于 global 集群的控制平面节点，至少 100GB； 对于其他节点，至少 40GB	200GB	如果预计 infra node 组件需要更多 /cpaas/ 空间，请预留额外容量。
/	50GB	100GB，越大越好。	确保有足够的可用磁盘空间，使使用率保持在 80% 以下。如果使用率超过该阈值，节点上的 Pod 可能会被逐出。
用于下载并解压安装程序包、扩展等内容的任意位置。	20GB	250GB	实际存储需求会因计划安装的扩展而异。 如果预计后续要添加更多组件或启用额外功能，请预留额外空间。

推荐的 ETCD 实践

快速存储对于 etcd 可靠运行至关重要。etcd 依赖持久、低延迟的磁盘操作，将提案持久化到其预写日志（WAL）中。
如果磁盘写入耗时过长，fsync 延迟可能会导致成员错过心跳、无法及时提交提案，并出现请求超时或临时领导者切换。这些问题还会拖慢 Kubernetes API，并降低整个集群的响应速度。
总之，HDD 是较差的选择，不建议使用。如果必须在 etcd 中使用 HDD，请选择可用的最快型号（例如 15,000 RPM）。

INFO

以下硬盘实践可为 etcd 提供最佳性能：

- 将 SSD 或 NVMe 作为 etcd 磁盘优先选择。当写入耐久性和稳定性是优先考虑因素时，可以考虑服务器级单层单元（SLC）SSD。避免使用 NAS、SAN 和 HDD。
    - 优先选择写入吞吐量高的磁盘，以加速压缩和碎片整理。

  - 优先选择读取带宽高的磁盘，以缩短故障后的恢复时间。
  - 优先选择读写延迟始终较低的磁盘，以确保快速的读写操作。
- 避免使用 Ceph RADOS Block Device（RBD）、Network File System（NFS）以及其他网络附加后端等分布式块存储系统，因为它们会引入不可预测的延迟。
- 将 etcd 数据保存在专用磁盘或专用逻辑卷上。
  - 不要在控制平面主机上放置 I/O 敏感型（例如日志记录）或其他高强度的文件系统活动，或者至少不要让它们与 etcd 共享同一底层存储。
- 使用 `fio` 等工具持续进行基准测试，并利用结果在集群扩展时跟踪性能。有关详细信息，请参见 磁盘基准测试指南。

验证 **etcd** 的硬件

Specification	Minimum Requirement	Recommended	Notes
Sequential write IOPS	50	500（越大越好）	大多数云提供商公布的是 *并发* IOPS，而不是 *顺序* IOPS。并发 IOPS 的值通常比顺序 IOPS 高约 10 倍。
Disk bandwidth	10 MB/s	100 MB/s（越大越好）	更高的磁盘带宽可在故障成员需要与集群追赶时加快数据恢复。
Throughput (sequential 8 kB write with `fdatasync`)	50 writes per 10 ms	500 writes per 2 ms	反映了在每次写入操作后将数据刷新到磁盘时的持续写入吞吐量。

使用 **fio** 进行基准测试

要测量实际的顺序 IOPS 和吞吐量，请使用磁盘基准测试工具 `fio`：

WARNING

不要在 ACP 集群的任何节点上运行这些测试。

请改为在一台具有与控制平面节点相同配置的专用 VM 上运行测试。\

```
set -e
mkdir -p /var/lib/etcd/

echo "INFO: Running fio"
fio --rw=write --ioengine=sync --fdatasync=1 --directory=/var/lib/etcd --size=100m --bs=8000 --name=etcd_perf --output-format=json --runtime=60 --time_based=1 | tee /tmp/fio.out

# Scrape the fio output for p99 of fsync in ns
fsync=$(cat /tmp/fio.out | jq '.jobs[0].sync.lat_ns.percentile["99.000000"]')
iops=$(cat /tmp/fio.out | jq '.jobs[0].write.iops')
echo "INFO: 99th percentile of fsync is $fsync ns"

# Compare against the recommended value
if [[ $fsync -ge 10000000 ]]; then
    echo "WARN: IOPS is $iops, 99th percentile of the fsync is greater than the recommended value which is ${fsync} ns > 10 ms, faster disks are recommended to host etcd for better performance"
else
    echo "INFO: IOPS is $iops, 99th percentile of the fsync is within the recommended threshold: - 10 ms, the disk can be used to host etcd"
fi
```