可扩展性和性能

概览 评估业务集群的资源 评估 **global**

使用管理器策略优化 **Pod** 性能 提升大规模集群的 **Kubernetes** 稳 磁盘配置

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。此区域汇集了规模评估、磁盘配置、资源管理器策略以及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。	评估全局集群
配置 CPU、内存和拓扑管理器策略。	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
配置磁盘相关的性能设置。	磁盘配置
规划基础设施节点或自定义角色节点。	基础设施节点

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

评估业务集群的资源

目录

推荐的控制平面实践

控制平面节点规格

主要版本的 ACP 已测试集群上限

示例

推荐的控制平面实践

使用以下实践来规划 ACP 中业务集群控制平面的资源。

> **INFO**
>
> 以下要求基于安装了 Core 软件包的最低 ACP 部署。安装 Extensions 后，实际要求可能会更高。请阅读特定于 Extension 的指导，了解其他资源要求。

控制平面节点规格

控制平面规格需求会随集群的节点数量、节点类型以及运行的对象数量和类型而变化。以下建议源自集群密度测试，这些测试用于评估负载下的控制平面行为。测试会在指定的一组命名空间中部署以下对象：

- 6 个部署，其中的 Pod 仅运行 sleep 进程
- 6 个服务
- 6 个指向上述服务的 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 ConfigMap

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

上表数据基于安装在公有云虚拟机上的 ACP 环境，其中控制平面节点配置为 16 个 vCPU 和 128 GB RAM，工作节点配置为 8 个 vCPU 和 32 GB RAM。

在有三个控制平面节点的大型高密度集群中，若有一个节点下线——无论是由于电力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主动关机——剩余两个节点都必须承担额外工作。此时，存活的控制平面机器上的 CPU 和内存使用量可能会显著上升。
升级期间也会出现这种情况，因为更新控制平面 Operator 时，通常会依次封锁、驱逐并重启控制平面节点。这种顺序维护会将负载集中到仍在运行的节点上。
为降低级联故障的风险，请确保控制平面服务器有充足的资源余量：目标是将总体 CPU 和内存利用率保持在约 60% 或以下，以便集群能够应对短暂的负载增长。如有必要，请增加控制平面的 CPU 和 RAM，以防止资源耗尽导致潜在中断。

> 重要说明
>
> 节点规格会因集群中的节点数量和对象数量而异，也取决于是否正在集群中主动创建对象。创建对象期间，与对象处于 **Running** 阶段时相比，控制平面的资源使用活动会更加频繁。

主要版本的 **ACP** 已测试集群上限

> **WARNING**

过度使用节点的物理资源可能会削弱 Kubernetes 所依赖的调度保障。请采用资源请求和限制、QoS 类以及节点调优等控制措施，尽量减少内存交换和其他资源争用。

这些数据基于 灵雀云 的测试配置、方法和调优结果。以下条目列出的是在测试条件下观察到的 ACP 最大值，而非固定上限。 由于 ACP 版本、控制平面工作负载和网络插件的组合无限多，因此这些数值并非适用于所有部署的保证上限，也不一定能在所有维度上同时达到。
规划具有类似特征的部署时，可将这些数值作为参考。\

INFO

增加或减少 ACP 集群中的节点数量时，建议：

- 如有可用区域，请将节点分布到所有可用区域，以提高可用性。
- 每次扩容或缩容不超过 25 到 50 个节点。

缩减大型、高密度集群时，操作可能需要相当长的时间，因为在关闭待移除节点之前，必须迁移或终止这些节点上的工作负载。如果一次需要处理大量资源，耗时可能尤其长。
如果需要驱逐大量对象，API 客户端可能会开始对请求进行速率限制。默认客户端每秒查询数 (QPS) 和突发设置分别为 **50** 和 **100**，且在 ACP 中无法更改这些值。\

上限类型	已测试上限
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
ConfigMap 数量	40,000
服务数量	10,000

上限类型	已测试上限
每个命名空间的服务数量	5,000
每个服务的后端数量	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；此处列出的是建议的最大值。如果您的使用场景需要更大的集群，请联系技术支持。
2. 此处列出的 Pod 数量是测试环境中的计数。实际 Pod 容量会因各原生应用的内存、CPU 和存储需求而异。
3. 如果有大量活跃项目，etcd 的键空间可能会过大并超出空间配额，进而导致性能下降。建议定期维护 etcd（例如进行碎片整理），以回收存储空间并避免性能问题。
4. 多个控制循环会在状态发生变化时遍历命名空间中的所有对象。在单个命名空间中，某一类型的对象数量过多会增加这些循环的开销，并减慢处理速度。此处给出的上限假设系统拥有足够的 CPU、内存和磁盘资源来满足原生应用的需求。
5. 测试在一个包含 29 台服务器的集群上运行（3 个控制平面节点、2 个基础设施节点和 24 个工作节点），其中有 500 个命名空间。ACP 将自定义资源定义 (CRD) 总数限制为 1,024 个，其中包括由 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 可能导致 **kubectl** 请求遭到限流。

示例

在一个测试场景中，使用 ACP 4.1、Kube-OVN 网络插件以及以下工作负载对象，对 500 个工作节点进行了测试。每个工作节点配备 8 个 vCPU 和 32 GB 内存：

- 除默认值外，另有 200 个命名空间
- 每个节点 60 个 Pod；30 个服务器 Pod 和 30 个客户端 Pod（总计 30k 个）
- 由服务器 Pod 提供后端的服务，每个命名空间 15 个（总计 3k 个）
- 每个命名空间 20 个 Secret（总计 4k 个）
- 每个命名空间 10 个 ConfigMap（总计 2k 个）
- 每个命名空间 6 个网络策略，包括拒绝所有、允许来自 Ingress 的流量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能力，规划扩展时应将其纳入考虑。如需更多指导，请联系技术支持。

- 每个节点的 Pod 数量
- 每个 Pod 的容器数量
- 使用的探针类型（例如，存活性/就绪性、exec/http）
- 网络策略数量
- 项目或命名空间数量
- 服务/端点数量及其类型
- 分片数量
- Secret 数量
- ConfigMap 数量
- API 调用速率，用于估算集群配置变化的速度。
  - 按 5 分钟窗口统计每秒 Pod 创建请求数的 Prometheus 查询：
    `sum(irate(apiserver_request_count{resource="pods",verb="POST"}[5m]))`
  - 按 5 分钟窗口统计每秒所有 API 请求数的 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
节点规格
生产环境基准规格
垂直扩展指南
水平扩展指南
资源验证与监控
总结

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。

使用管理器策略优化 Pod 性能

Kubernetes 集群管理员可以启用并验证 **CPU ManagerPolicy**、**Memory ManagerPolicy** 和 **Topology ManagerPolicy**，以协调延迟敏感型工作负载的 CPU 绑定、NUMA 亲和性和拓扑放置。

目录

范围和先决条件

快速开始：示例 Kubelet 配置

如何计算 `reservedMemory`

应用配置

Alauda OS

非 Alauda OS

验证

关键策略和行为

术语

范围和先决条件

角色和权限

- 需要维护时段访问权限、`kubectl` 管理员权限以及节点的 SSH 访问权限。

工作负载要求

- 要实现专用 CPU 和 NUMA 亲和性，Pod 必须在 **Guaranteed QoS** 类别中运行：requests = limits，且 CPU 必须以完整核心数指定（例如 2、4）。

不包括的内容

- HugePages 不在本指南范围内。如果需要 HugePages 支持，请联系支持团队。

快速开始：示例 **Kubelet** 配置

以下 kubelet 配置可启用 NUMA 感知调度。请根据您的环境调整值。具体应用方式取决于节点操作系统，请参阅应用配置。

```
# ── 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
```

注意：

- `full-pcpus-only: "true"` 可提高延迟一致性。
- `topologyManagerScope: pod` 可确保同一 Pod 中的容器与公共 NUMA 拓扑对齐。
- `reservedMemory` 必须根据 kubelet 配置和驱逐阈值计算（请参阅下一节）。

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（对于 **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

这些 kubelet 设置的应用方式取决于节点操作系统。

Alauda OS

对于运行 Alauda OS 的节点，CPU 和 Memory Manager 策略由 Alauda 技术支持协助应用。请联系 Alauda 支持代表，在这些节点上启用这些策略。

非 Alauda OS

对于运行非 Alauda OS 的节点，请在每个节点上应用配置：

1. 隔离并清空节点

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

2. 停止 **Kubelet** 并清除状态

```
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 DaemonSet 和系统 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"`
- 预留内存总量与 `R_total` 匹配

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

关键策略和行为

CPU ManagerPolicy

- 用途：为 Guaranteed Pod 分配独占的物理 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

- 用途：在 NUMA 节点级别预留并对齐内存
- 配置：`memoryManagerPolicy: "Static"`、`reservedMemory`
- 行为：与 Topology ManagerPolicy 配合使用可实现最佳对齐效果

Topology ManagerPolicy

- 用途：将 CPU、内存和设备分配对齐到同一 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** 绑定：将容器绑定到独占 CPU
- **NUMA** 亲和性：优先使用与 CPU 位于同一 NUMA 节点的内存
- 拓扑对齐：将 CPU、内存和设备集中放置在一个 NUMA 节点上
- **Guaranteed Pod**：requests = limits；CPU 指定为完整核心

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

Footnotes

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

ETCD 最佳实践

验证 etcd 的硬件

使用 fio 进行基准测试

存储容量

INFO

将以下分区挂载到专用磁盘或由 LVM 配置的逻辑卷上，以便日后扩展。

分区	最小大小	建议大小	备注
/var/lib/etcd	10GB	20GB	建议使用专用的高 IO 磁盘来存储 etcd 数据。
/var/lib/containerd/	100GB	150GB	
/cpaas/	global 集群的控制平面节点至少为	200GB	如果预计 infra 节点组件需要更多 /cpaas/ 空间，请规

分区	最小大小	建议大小	备注
	100GB； 其他节点至少为 40GB		划额外空间。 在 Alauda OS 节点上，此分区为 `/var/cpaas/`。
/	50GB	100GB，越高越好。	确保有足够的可用磁盘空间，使使用率保持在 80% 以下。如果使用率超过此阈值，节点上的 Pod 可能会被驱逐。
用于下载和解压安装程序包、扩展等内容的任意位置。	20GB	250GB	实际存储需求将根据计划安装的扩展而有所不同。 如果预计以后会添加更多组件或启用额外功能，请规划额外空间。

INFO

上述分区布局描述的是运行非 Alauda OS 的节点。在节点运行 Alauda OS 的集群中，平台状态存储在 `/var/cpaas` 下，而不是 `/cpaas` 下；节点磁盘则在 provider 的机器配置中声明，而不是手动进行分区。容量建议仍然适用。有关各 provider 的磁盘声明操作步骤，请参阅Alauda OS 上创建工作负载集群。

ETCD 最佳实践

快速存储对 etcd 的可靠运行至关重要。etcd 依赖持久、低延迟的磁盘操作，将提案写入其预写日志 (WAL)。
如果磁盘写入耗时过长，fsync 延迟可能导致成员错过心跳、无法及时提交提案，并出现请求超时或临时的领导者变更。这些问题还可能拖慢 Kubernetes API，降低整个集群的响应能力。
总之，HDD 并非理想选择，不建议使用。如果必须将 HDD 用于 etcd，请选择可用的最快型号（例如 15,000 RPM）。

INFO

遵循以下硬盘使用建议，可让 etcd 获得最佳性能：

- etcd 磁盘优先选择 SSD 或 NVMe。当写入耐久性和稳定性至关重要时，可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，以加快压缩和碎片整理。
  - 优先选择读取带宽强的磁盘，以缩短故障后的恢复时间。
  - 优先选择延迟始终较低的磁盘，以确保快速读写。
- 避免使用 Ceph RADOS Block Device (RBD)、Network File System (NFS) 等分布式块存储系统，以及其他网络连接的后端，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运行 I/O 敏感型活动（例如日志记录）或其他密集的文件系统活动；至少不要让这些活动与 etcd 共用同一底层存储。
- 持续使用 `fio` 等工具进行基准测试，并根据测试结果跟踪集群增长时的性能。更多信息，请参阅磁盘基准测试指南。

验证 **etcd** 的硬件

规格	最低要求	建议	备注
顺序写入 IOPS	50	500（越高越好）	大多数云提供商公布的是*并发* IOPS，而不是*顺序* IOPS。并发 IOPS 值通常约为顺序 IOPS 的 10 倍。
磁盘带宽	10 MB/s	100 MB/s（越高越好）	更高的磁盘带宽可在成员发生故障、需要追赶集群进度时，加快数据恢复速度。
吞吐量（使用 `fdatasync` 进行顺序 8 kB 写入）	每 10 ms 写入 50 次	每 2 ms 写入 500 次	反映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。

请改为在配置与控制平面节点相同的专用 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=$(cat /tmp/fio.out | jq '.jobs[0].sync.lat_ns.percentile["99.00000
0"]')
iops=$(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 th
an the recommended value which is ${fsync} ns > 10 ms, faster disks are r
ecommended 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
```