美团面试：MySQL为什么 不用 Docker部署？

原创 尼恩架构团队 技术自由圈 2025年01月27日 22:10 中国香港

尼恩说在前面

在45岁老架构师 尼恩的**读者交流群**(50+)中，最近有小伙伴拿到了一线互联网企业如得物、阿里、滴滴、极兔、有赞、希音、百度、网易、美团、蚂蚁、得物的面试资格，遇到很多很重要的相关面试题：

> 问题：MySQL为什么不推荐使用Docker部署?

最近有小伙伴面试美团 问到此 面试题。小伙伴没有系统的去梳理和总结，所以支支吾吾的说了几句，面试官不满意，面试挂了。

所以，尼恩给大家做一下系统化、体系化的梳理，使得大家内力猛增，可以充分展示一下大家雄厚的“技术肌肉”，**让面试官爱到 “不能自已、口水直流”**，然后实现”offer直提”。

当然，这道面试题，以及参考答案，也会收入咱们的 《尼恩Java面试宝典PDF》V175版本，供后面的小伙伴参考，提升大家的 3高 架构、设计、开发水平。

> 《尼恩 架构笔记》《尼恩高并发三部曲》《尼恩Java面试宝典》的PDF，请到文末公号【技术自由圈】获取

MySQL为什么不推荐使用Docker部署

docker可以从远程仓库拉取镜像然后通过镜像快速的部署应用，非常的方便快捷，

但是，为什么 一般公司的 Mysql 不用docker部署，而是部署在 物理机器上呢？

解下来， 45岁老架构师尼恩， 给大家彻底的梳理一下， 让面试官口水直流。

本文目录

第一大问题：DB有状态，不方便扩容

1.1 Docker容器的两大类型： 有状态 、无状态的区分

Docker容器的两大类型： 有状态 、无状态的区分，它们在数据存储、应用场景、管理方式等方面存在明显区别。

有状态容器：

容器在运行过程中， 需要 持久化 和管理数据状态信息的容器。

有状态容器需要对这些数据进行持久化存储，以便在容器重启或迁移后能够恢复到之前的状态，保证应用程序的正常运行。

有状态容器 需要使用数据卷（Volumes）、绑定挂载（Bind Mounts）或网络存储等方式将数据持久化到宿主机或外部存储设备上，确保数据在容器的生命周期之外也能保存。

有状态容器 适用于需要保存和管理数据的应用，如数据库系统、消息队列、文件服务器等。这些应用需要在不同的请求之间保持数据的一致性和完整性，并且能够在容器重启或故障转移后恢复到之前的状态。

有状态容器 扩容 复杂。需要考虑数据的一致性和分布问题，确保新启动的容器能够正确地访问和更新共享数据。

无状态容器：

容器在运行过程中不保存任何数据状态信息 。

无状态容器通常只负责处理输入请求并返回结果，不依赖于之前的运行状态。

它们可以随时被销毁、重启或替换，而不会对应用程序的整体功能产生影响，因为所有需要的信息都通过外部输入或共享资源提供。

无状态容器 一般不需要专门的数据持久化机制，容器内产生的临时数据通常存储在容器的可写层，但这些数据在容器重启或删除后会丢失。

无状态容器 常用于处理无状态的任务，如 Web 应用程序的前端服务、负载均衡器、API 网关等。这些应用可以根据接收到的请求即时生成响应，不需要依赖于内部的状态信息。

有状态容器 扩容 简单。由于容器之间没有状态依赖关系，可以根据负载情况轻松地增加或减少容器实例，实现水平扩展。容器编排工具（如 Kubernetes）可以很方便地对无状态容器进行调度和管理，实现高可用性和弹性伸缩。

1.2 Mysql 是有状态的，不方便扩容

1.3 为什么 MySQL 是有状态的?

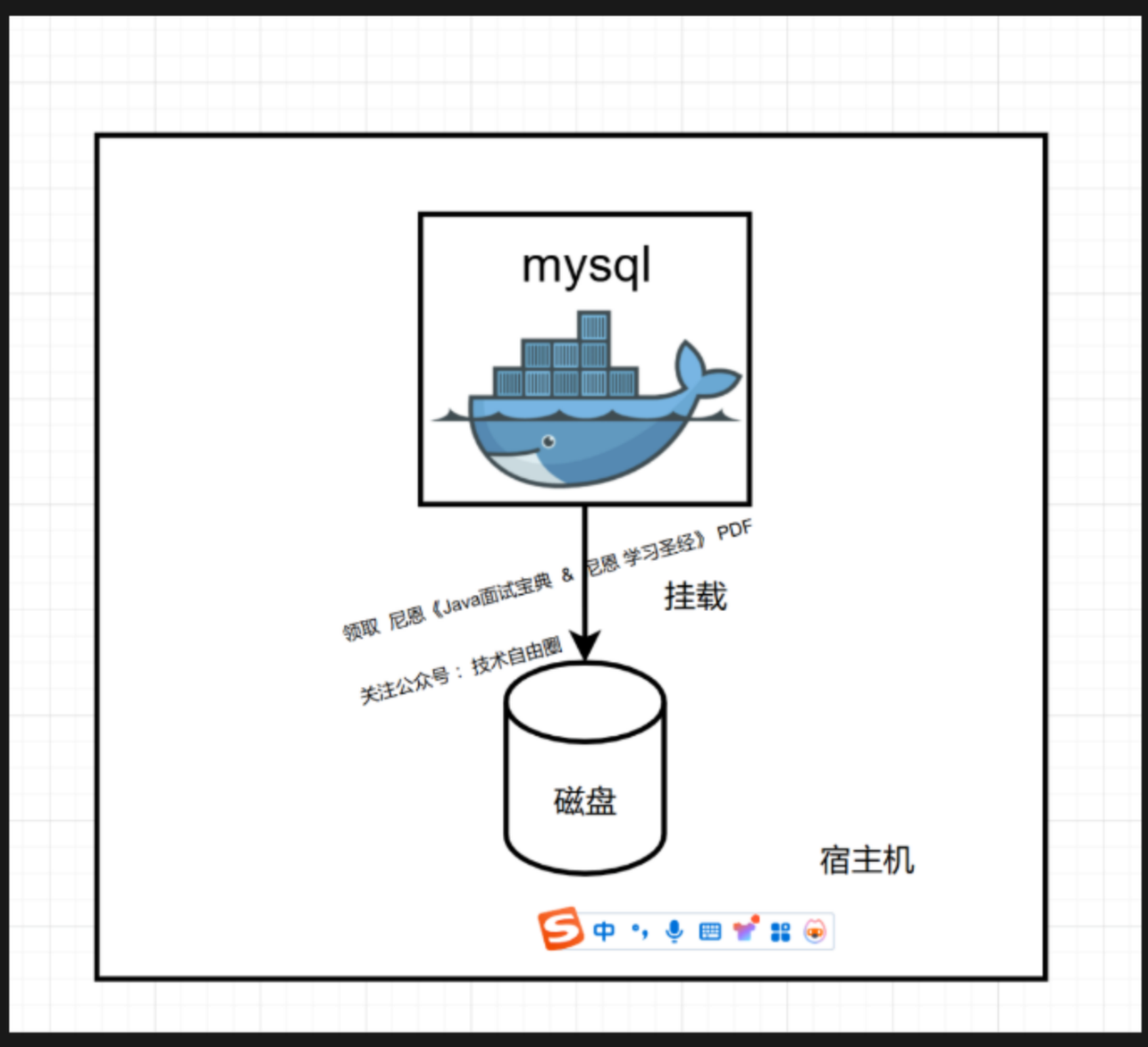
1. **数据持久化**：MySQL 需要将数据存储在磁盘上，以便在容器重启后数据仍然存在。
2. **配置文件**：MySQL 的配置文件（如 `my.cnf`）也需要持久化，以便在容器重启后仍然生效。
3. **日志文件**：MySQL 的日志文件（如错误日志、二进制日志等）也需要持久化，以便在需要时进行故障排查和数据恢复。

如何在 Docker 中实现 MySQL 的数据持久化？

Mysql是用来存储的数据，docker部署Mysql之后, 数据文件 不会存储在容器内部，因为容器关闭之后，内部数据就丢失了。

所以， 为了确保 MySQL 数据在 Docker 容器关闭后不会丢失，需要将数据文件、配置文件和日志文件挂载到宿主机上

数据文件 需要挂载到容器外部，也就是 宿主机器上，如下图所示：



Docker部署MySQL并实现数据持久化， 以下是实现步骤：

第一步，在宿主机上，创建存储目录：

在宿主机上创建目录来存储MySQL的数据、日志和配置文件：

```
mkdir -p /data/mysql/{data,logs,conf}
```

第2步，拉取MySQL镜像： 通过Docker Hub拉取MySQL的官方镜像：

```
docker pull mysql:latest
```

第3步，配置MySQL： 在 `/data/mysql/conf` 目录下，创建一个名为 `my.cnf` 的配置文件，用于设置MySQL的字符集、排序规则等参数：

```
[mysqld]
character-set-server=utf8mb4
collation-server=utf8mb4_unicode_ci
datadir=/var/lib/mysql
log-error=/var/log/mysqld.log
```

第4步，启动MySQL容器：

使用以下命令启动MySQL容器，并将其宿主机目录挂载到容器内部：

```
docker run -d \
  --name mysql-server \
  -p 3306:3306 \
  -e MYSQL_ROOT_PASSWORD=your_password \
  -v /data/mysql/data:/var/lib/mysql \
  -v /data/mysql/logs:/var/log/mysqld \
  -v /data/mysql/conf/my.cnf:/etc/mysql/my.cnf \
  mysql:latest
```

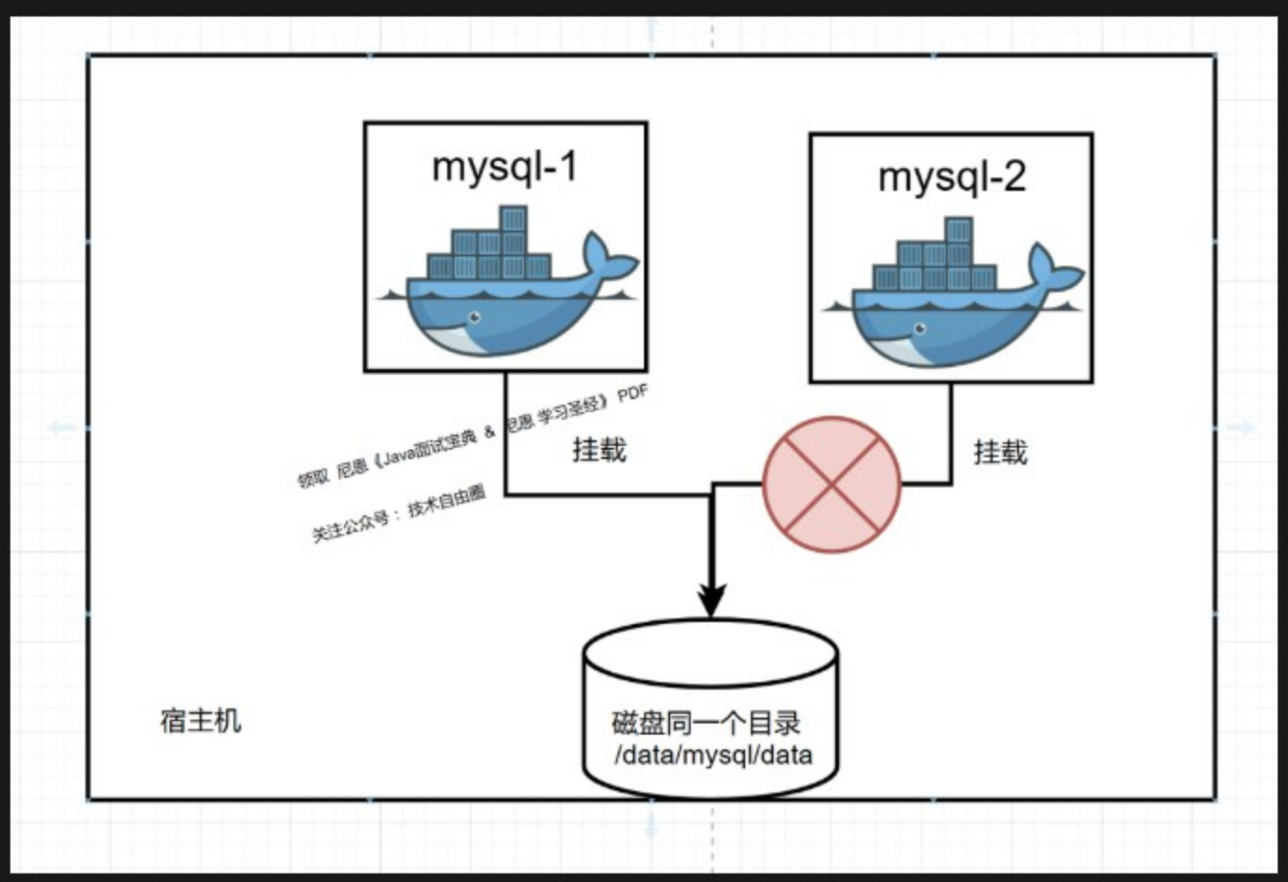
1.4 容器化部署mysql 带来的扩容困境

一个容器的资源，是非常有限的。

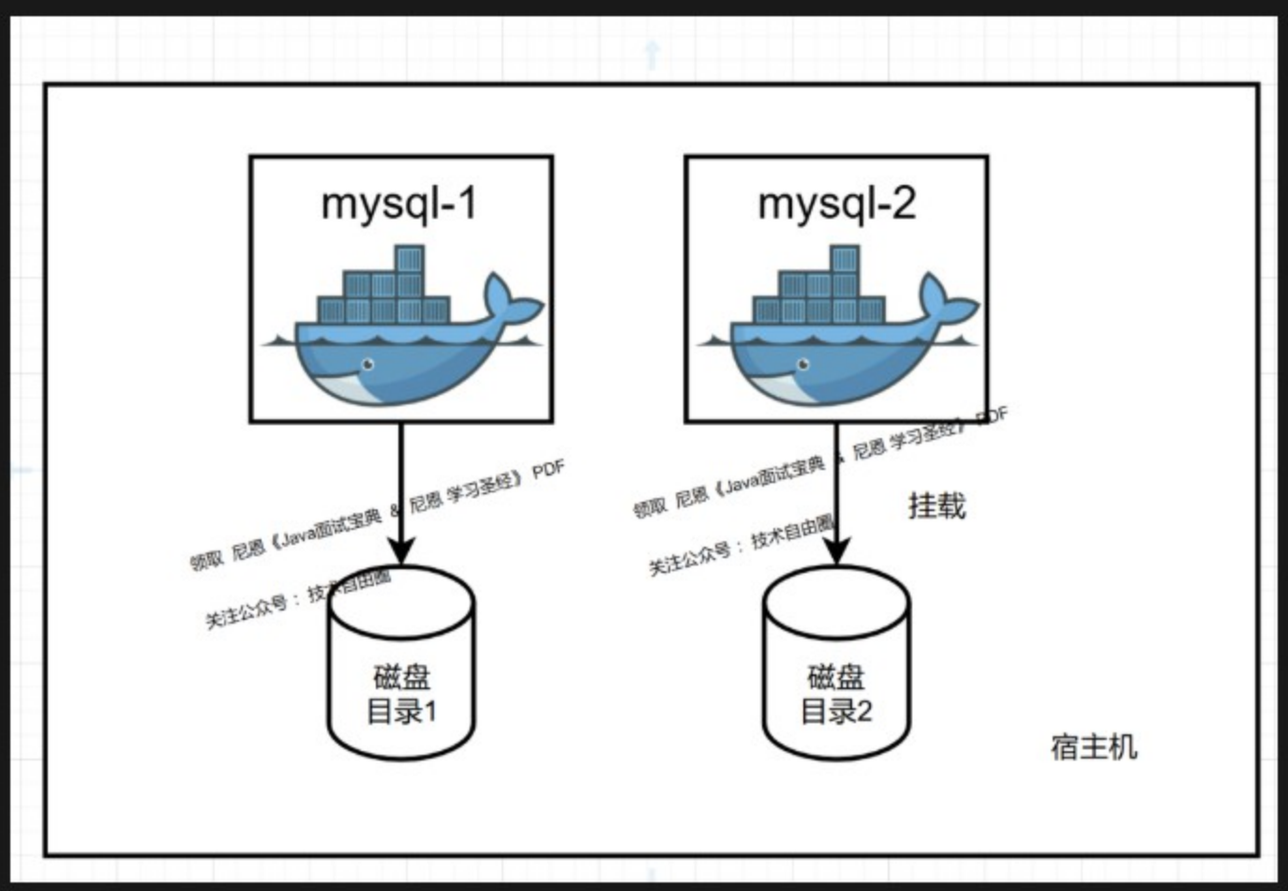
随着业务的持续不断的发展，Mysql难免会出现需要扩容的情况。

但是，如何扩容呢？

扩容的时候就会出现一些困难，如下所示：



此时 Mysql2是无法挂载到文件1上的，因为宿主机的数据文件是容器独占的，无法实现两个容器实例共享一份数据文件。



每个Mysql的容器都挂载了各自的文件上

> 不要将数据储存在容器中，这也是 Docker 官方容器使用技巧中的一条。
>
> 容器随时可以停止、或者删除。
>
> 当容器被rm掉，容器里的数据将会丢失。为了避免数据丢失，用户可以使用数据卷挂载来存储数据。

1.5 使用Docker在本地部署 两个MySQL实例

为了在本地部署两个MySQL实例，需要为每个实例创建独立的数据目录，并确保它们使用不同的端口。以下是详细步骤：

1. 创建2套存储目录

在宿主机上创建两个目录，分别用于存储两个MySQL实例的数据、日志和配置文件：

bash复制

```bash
mkdir -p /data/mysql1/{data,logs,conf}
mkdir -p /data/mysql2/{data,logs,conf}
```

2. 配置2套MySQL

在每个实例的配置目录下，创建一个名为 `my.cnf` 的配置文件，用于设置MySQL的字符集、排序规则等参数：

MySQL 1 配置文件 (`/data/mysql1/conf/my.cnf`):

ini复制

```ini
[mysqld]
character-set-server=utf8mb4
collation-server=utf8mb4_unicode_ci
datadir=/var/lib/mysql
log-error=/var/log/mysqld.log
```

MySQL 2 配置文件 (`/data/mysql2/conf/my.cnf`):

ini复制

```ini
[mysqld]
character-set-server=utf8mb4
collation-server=utf8mb4_unicode_ci
datadir=/var/lib/mysql
log-error=/var/log/mysqld.log
```

3. 启动2套MySQL容器

使用以下命令启动两个MySQL容器，并将各自的宿主机目录挂载到容器内部。注意，每个容器需要使用不同的端口：

启动MySQL 1:

bash复制

```bash
docker run -d \
  --name mysql1 \
  -p 3306:3306 \
  -e MYSQL_ROOT_PASSWORD=your_password1 \
  -v /data/mysql1/data:/var/lib/mysql \
  -v /data/mysql1/logs:/var/log/mysqld \
  -v /data/mysql1/conf/my.cnf:/etc/mysql/my.cnf \
  mysql:latest
```

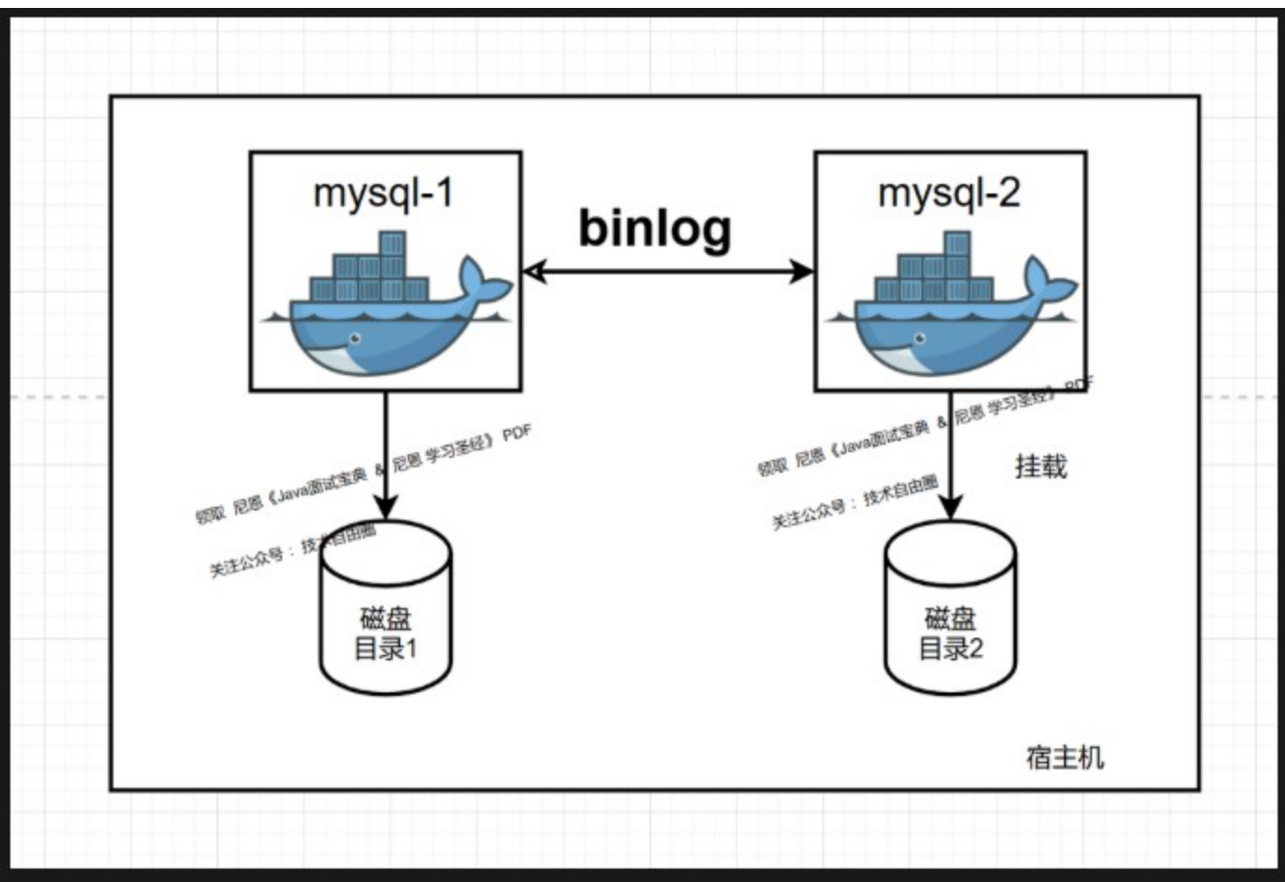
启动MySQL 2:

bash复制

```bash
docker run -d \
  --name mysql2 \
  -p 3307:3306 \
  -e MYSQL_ROOT_PASSWORD=your_password2 \
  -v /data/mysql2/data:/var/lib/mysql \
  -v /data/mysql2/logs:/var/log/mysqld \
  -v /data/mysql2/conf/my.cnf:/etc/mysql/my.cnf \
  mysql:latest
```

1.6 如何实现文件1和文件2的数据共享呢？

此时有如下的解决方案：binlog同步



Mysql扩容是因为业务数据量达到瓶颈了，那么大数据量下使用binlog同步，一是影响性能，二是会发生数据延迟。

所以， 这个方案， 还是不合理。

1.7 MySQL 容器的状态问题 总结

MySQL 是一个有状态的服务，因为它需要持久化存储数据。

在使用 Docker 部署 MySQL 时，如果不进行适当的配置，容器关闭后数据会丢失。

因此，需要将数据文件挂载到宿主机上，以确保数据的持久化。

第二大问题：Docker 的资源隔离，不彻底

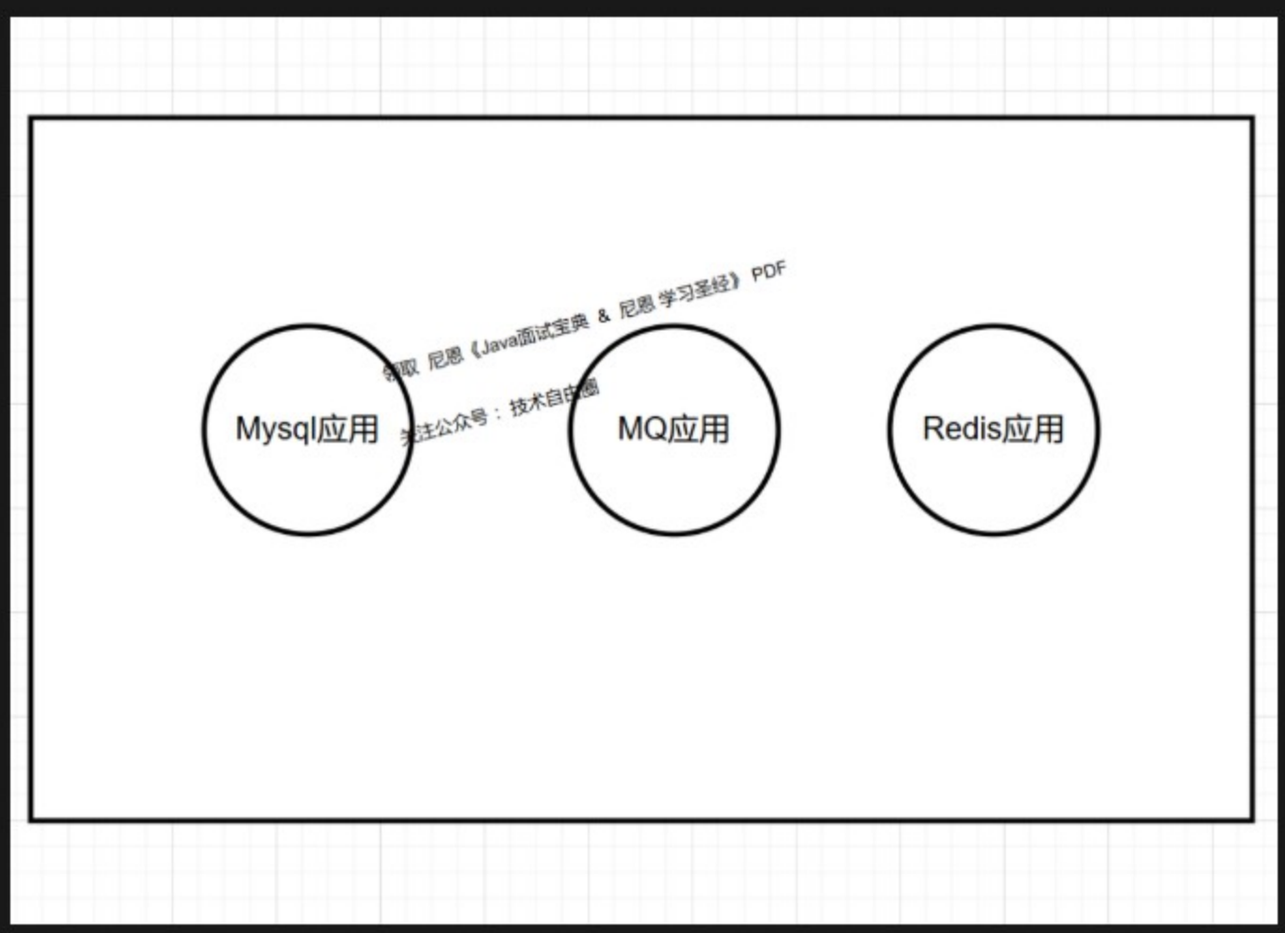
资源隔离方面， Docker 资源隔离不彻底。

Docker是利用Cgroup实现资源限制的：

- Docker只是限制资源消耗的最大值，而不能隔绝其他程序占用自己的资源。
- Docker 并没有真正、彻底的隔离CPU、内存。

假设 Mysql 、Springboot 、Redis 都是docker部署的， 如果其他应用过渡占用物理机资源，将会影响容器里 MySQL 的读写效率。

如下图所示：



假设16G内存，如果Springboot 占用了8G的内存，Redis占4G的内存，Mysql 只剩 4G 了，如果数据量比较大，这个数量是远远不够的。

Mysql内存不够， Mysql无法提供正常的服务，会导致整个上层的应用崩溃。

第三大问题：Docker不适合于 磁盘IO密集型的中间件

Docker 会损耗 磁盘IO性能：

Docker 容器的文件系统是隔离的，容器内的 IO 操作需要通过 Docker 的抽象层进行转发，这会带来一定的性能损耗。

当容器内的应用程序进行大量的小文件读写操作时，每一次的 IO 请求都需要经过 Docker 的文件系统抽象层，增加了系统调用的开销，可能导致 IO 性能下降。

对于大型 MySQL 数据库，往往需要处理大量的数据读写操作，对 IO 性能要求极高。

例如，在进行大规模数据导入导出或复杂的查询操作时，Docker 的 IO 开销可能会导致性能明显下降。

Docker 会损耗网络IO性能：

在容器网络方面，Docker 为容器提供了网络命名空间和虚拟网络设备等隔离机制，这虽然保证了容器网络的独立性和安全性，但也增加了网络 IO 的复杂性和开销。

例如，容器与宿主机或其他容器之间的网络通信需要经过网络地址转换（NAT）和虚拟网络设备的转发，可能会导致网络延迟增加，影响网络 IO 性能。

所以，对于磁盘IO密集型的 应用，其实不适合直接放在 Docker 里面 。

目前，腾讯云的TDSQL（金融分布式数据库）和阿里云的Oceanbase（分布式数据库系统）都直接运行中在物理机器上，并非 Docker 上。

Docker 更适合部署 业务 微服务应用， 而不太适合 部署 磁盘IO密集型 Mysql，尤其是大型 Mysql。

4 Docker的优势

Docker 容器化技术在现代软件开发和部署中扮演着越来越重要的角色，它提供了许多显著的优势，尤其是在自动伸缩、容灾和切换等方面。

4.1. 自动伸缩

水平伸缩

支持水平扩展，即通过增加容器数量来提升整体处理能力，而不是依赖于提升单个服务器的性能。这种弹性扩展能力使得应用能够轻松应对突发的高并发流量，如电商平台的促销活动、社交媒体的热点事件等，确保服务的稳定性和响应速度。

Docker 容器可以根据应用程序的实际负载情况，快速、灵活地增加或减少容器实例数量。

- 当业务流量高峰时，自动增加容器数量来处理更多请求，避免因资源不足导致服务卡顿或崩溃；
- 流量低谷时，自动减少容器数量，释放多余资源，避免资源浪费，从而实现资源的高效利用，降低成本。

使用 Docker Compose 或 Kubernetes 等工具可以轻松地定义服务的副本数量，并根据负载自动调整。

```yaml
version: '3'
services:
  web:
    image: my-web-app
    deploy:
      replicas: 3
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
```

垂直伸缩

垂直伸缩 是 通过增加或减少单个容器的资源（如 CPU、内存）来应对负载变化。

容器可以根据实际需求实时获取或释放资源，如 CPU、内存等。

与传统的虚拟机相比，容器启动和停止速度更快，能够更迅速地适应负载变化，实现资源的动态分配，保证应用始终在最佳性能状态下运行。

Docker 和 Kubernetes 提供了资源限制和请求的配置选项，可以动态调整容器的资源分配。

```
version: '3'
services:
  web:
    image: my-web-app
    deploy:
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
```

4.2. 容灾

每个 Docker 容器都是相对独立的运行环境，一个容器内的故障通常不会影响到其他容器。即使某个容器出现问题，如程序崩溃、内存泄漏等，只会导致该容器自身停止运行，而不会导致整个系统崩溃，提高了系统的稳定性和可靠性。

基于容器的镜像和存储技术，当容器出现故障时，可以快速从镜像中重新创建容器，恢复服务。而且可以通过数据卷等方式将重要数据持久化存储，确保在容器故障或重建时数据不会丢失，实现快速的数据恢复和业务连续性。

Docker的挂载机制，能确保数据在容器重启或故障后仍然存在。通过将数据卷挂载到宿主机或使用外部存储服务（如 AWS EBS、Google Persistent Disk）来实现数据持久化。

```
docker run -d \
  --name mysql-server \
  -p 3306:3306 \
  -e MYSQL_ROOT_PASSWORD=your_password \
  -v /data/mysql/data:/var/lib/mysql \
  mysql:latest
```

自动重启

- **定义**：在容器故障时自动重启容器。
- **实现**：Docker 提供了 `--restart` 选项，可以设置容器在失败时自动重启。
- **示例**：

  bash复制

  ```
  docker run -d \
    --name my-app \
    --restart=always \
    my-app-image
  ```

高可用性

Docker 容器可以很容易地在不同的数据中心或云平台之间进行迁移和部署。在发生自然灾害、网络故障等重大灾难时，可以快速将容器化的应用和数据转移到其他可用区域，实现异地容灾，保障业务的不间断运行。

使用 Kubernetes 等编排工具可以实现服务的高可用性，自动检测故障并重新调度容器。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: my-app-image
```

4.3. 其他优势

开发与生产一致性

确保开发环境和生产环境的一致性，减少"在我的机器上可以运行"的问题。使用 Dockerfile 和 Docker Compose 文件可以确保开发和生产环境使用相同的配置。

```dockerfile
FROM node:14
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
CMD ["npm", "start"]
```

快速部署

快速部署应用程序，减少部署时间。Docker 容器可以在几秒钟内启动，大大缩短了部署时间。

```
docker run -d --name my-app my-app-image
```

隔离性

每个容器都是独立的，互不干扰，提高了系统的稳定性和安全性。Docker 使用命名空间和控制组（cgroups）来隔离容器的资源。

```
docker run -d --name my-app1 my-app-image
docker run -d --name my-app2 my-app-image
```

4.4. Docker的优势总结

Docker 容器化技术提供了许多显著的优势，包括自动伸缩、容灾、切换、开发与生产一致性、快速部署和隔离性等。

这些优势使得 Docker 成为现代软件开发和部署中不可或缺的工具。

通过合理使用 Docker 和 Kubernetes 等工具，可以大大提高系统的可扩展性、可靠性和维护性。

5 总结：大型 Mysql 为何不用 docker部署?

大型 MySQL 通常不用 Docker 部署主要是出于性能、管理复杂度、稳定性等多方面因素的考量，以下是具体分析：

5.1 性能方面

IO 性能损耗：

Docker 容器在 IO 操作时，由于存在额外的文件系统抽象层和存储驱动，会带来一定的性能开销。

对于大型 MySQL 数据库，往往需要处理大量的数据读写操作，对 IO 性能要求极高。

例如，在进行大规模数据导入导出或复杂的查询操作时，Docker 的 IO 开销可能会导致性能明显下降。

资源隔离限制：

虽然 Docker 可以进行资源限制和隔离，但在实际运行中，多个容器共享宿主机资源时，可能会出现资源竞争的情况。大型 MySQL 数据库需要稳定且充足的 CPU、内存等资源来保证性能，在 Docker 环境中可能难以精准地满足其资源需求，导致性能波动。

5.2 管理与维护方面

配置管理复杂：

大型 MySQL 数据库通常需要进行复杂的配置优化，如调整缓存参数、线程池大小、日志设置等。

在 Docker 环境下，这些配置可能需要在容器内部和宿主机之间进行协调，增加了配置管理的难度和复杂性。而且，容器的配置文件可能会受到容器镜像和存储驱动的限制，不如直接在物理机或虚拟机上配置灵活。

数据持久化挑战：

对于大型 MySQL 数据库，数据的持久化和可靠性至关重要。

在 Docker 中实现数据持久化需要额外的配置和管理，如使用数据卷、外部存储等。

如果配置不当，可能会导致数据丢失或损坏的风险增加。

而且，在进行数据备份和恢复时，Docker 环境下的操作相对复杂，需要考虑容器的状态、数据卷的挂载等因素。

集群管理困难：

大型 MySQL 通常会采用集群架构来实现高可用性和扩展性，如主从复制、分布式数据库等。

在 Docker 环境下管理 MySQL 集群，需要考虑容器之间的网络通信、数据同步、节点故障恢复等问题，增加了集群管理的难度和复杂性。

5.3 稳定性与可靠性方面

容器技术成熟度：

尽管 Docker 技术已经相当成熟，但与直接在物理机或虚拟机上部署相比，仍然存在一定的稳定性风险。

对于大型 MySQL 数据库这种对稳定性要求极高的应用，任何微小的故障都可能导致严重的后果。例如，容器的运行时环境、存储驱动或网络插件等可能存在的漏洞或兼容性问题，都可能影响 MySQL 的稳定性。

故障排查复杂：

在 Docker 环境中，当大型 MySQL 数据库出现故障时，排查问题的难度相对较大。

需要同时考虑容器本身的问题、宿主机的问题以及两者之间的交互问题。

例如，容器内的 MySQL 进程崩溃，可能是由于容器资源限制、网络问题或 MySQL 本身的故障引起的，排查过程需要涉及多个层面的信息收集和分析，增加了故障定位和解决的时间。

除了mysql，现在互联网的数据库多是share nothing的架构，每一个节点其实都是有状态的，都不太适合使用docker。

6 Share Nothing 架构

Share Nothing 架构是一种分布式计算架构，其中每个节点都是独立的，不共享任何资源（如内存、磁盘等）。这种架构的优点包括高可扩展性、高可用性和低延迟。每个节点都可以独立运行，互不干扰，从而提高了系统的整体性能和可靠性。

6.1 为什么需要 Share Nothing 架构

1. **高可扩展性**：由于每个节点都是独立的，可以通过增加更多的节点来水平扩展系统，而不需要担心资源竞争。
2. **高可用性**：单个节点的故障不会影响其他节点的运行，从而提高了系统的整体可用性。
3. **低延迟**：每个节点都可以独立处理请求，减少了数据传输的延迟，提高了系统的响应速度。

6.2 Share Nothing 架构的应用场景

1. **分布式数据库**：如 MySQL Cluster、Cassandra、CockroachDB 等，这些数据库通过将数据分布在多个节点上，实现了高可用性和高扩展性。
2. **分布式计算框架**：如 Hadoop、Spark 等，这些框架通过将计算任务分布在多个节点上，实现了高效的分布式计算。

3. **微服务架构**：每个微服务可以独立部署和扩展，互不干扰，从而提高了系统的灵活性和可维护性。

6.3 Share Nothing 架构的实现

1. 数据分布

分片（Sharding）：将数据按照一定的规则分成多个片段，每个片段存储在不同的节点上。例如，可以根据用户ID、地理位置等进行分片。

一致性哈希（Consistent Hashing）：通过一致性哈希算法将数据均匀分布到多个节点上，减少节点增减时的数据迁移量。

2. 数据一致性

最终一致性（Eventual Consistency）：

在分布式系统中，数据的一致性可能不是立即的，但最终会达到一致状态。例如，Cassandra 采用最终一致性模型。

强一致性（Strong Consistency）：通过分布式事务协议（如 Paxos、Raft 等）确保数据在多个节点上的一致性。例如，CockroachDB 采用强一致性模型。

3. 故障恢复

副本（Replication）：

每个节点的数据都有多个副本，存储在不同的节点上，以防止单点故障。

自动故障转移（Automatic Failover）：

当某个节点故障时，系统自动将请求重定向到其他副本节点，确保服务的连续性。

6.4 现在互联网的数据库多是share nothing的架构

Share Nothing 架构是一种分布式计算架构，其中每个节点都是独立且自给自足的，系统中不存在单点竞争，节点之间不共享内存或磁盘存储等资源。每个节点都有自己独立的 CPU、内存、存储和网络接口，数据分散存储在多个节点上。

- **Cassandra**：是一款分布式 NoSQL 数据库，具有高度可扩展性和容错性，广泛应用于大规模分布式数据存储和处理场景。
- **MongoDB**：文档型数据库，支持水平扩展，通过数据分片技术实现 Share Nothing 架构，适用于存储和处理非结构化和半结构化数据。
- **Hadoop HDFS**：Hadoop 分布式文件系统，是 Hadoop 生态系统的核心组件之一，采用 Share Nothing 架构实现大规模数据的分布式存储和管理。

7 当然 小型的mysql ，还是可以用docker部署的

凡事不可一概而论。

轻量级或小型数据库，docker还是适合跑的。当docker服务挂掉，会自动启动新容器，而不是继续重启容器服务。

说在最后：有问题找老架构取经

只要按照上面的 尼恩团队梳理的 方案去作答， 你的答案不是 100分，而是 120分。面试官一定是 心满意足， 五体投地。

按照尼恩的梳理，进行 深度回答，可以充分展示一下大家雄厚的 “技术肌肉”，**让面试官爱到 “不能自已、口水直流”**，然后实现"offer直提"。

在面试之前，建议大家系统化的刷一波 5000页《尼恩Java面试宝典PDF》，里边有大量的大厂真题、面试难题、架构难题。

很多小伙伴刷完后， 吊打面试官， 大厂横着走。

在刷题过程中，如果有啥问题，大家可以来 找 40岁老架构师尼恩交流。

另外，如果没有面试机会， 可以找尼恩来改简历、做帮扶。

前段时间，**刚指导一个小伙 暴涨200%（涨2倍），29岁/7年/双非一本 ， 从13K一次涨到 37K ，逆天改命。**

另外，**刚指导一个 40岁大龄，上岸：转架构，收3个外企offer， 机会多了，不焦虑了，逆天改命。**

狠狠卷，实现 “offer自由” 很容易的， 前段时间一个武汉的跟着尼恩卷了2年的小伙伴， 在极度严寒/痛苦被裁的环境下， offer拿到手软， 实现真正的 “offer自由” 。