系统由两个及以上组件组成，组件可独立部署，组件之间互不关联，组件自己独享数据库，系统有自动化测试，具有以上特征的系统我们可以称为微服务

微服务架构主要有以下几个概念

API网关：API网关负责接收所有的API请求，并将请求转发至后端服务器，除此之外，api网关还具有验证权限，均衡负载，短路器等功能

发现服务、服务注册：一个服务要加入微服务系统，它需要向系统注册他的地址和服务名，这样API网关在转发请求时，才可以转发到该服务上

消息总线：微服务之间互不依赖，但微服务之间需要通信，而服务之间的通信则是靠消息进行传播的，所以我们需要一个消息总线

认证服务：认证服务器负责控制微服务的访问，认证服务可以与api网关相连，由api网关控制权限，也可以与各个服务相连，由各个服务自己控制

要实现微服务，我们需要提供系统的物理架构和逻辑架构

物理架构网上可以查到很多方案，我这里给出的简单方案就是搭建先k8s集群，在k8s集群上部署kong，kong为我们提供了api网关和发现服务器，接着我们需要部署消息总线，可以将MQ作为我们的消息总线，接着就是认证服务器，在.net技术中我们可以使用IdentityServer作为认证服务器，这样一个简单的微服务物流架构就搭建出来了

提供了物流架构后就是逻辑架构

逻辑架构是微服务实现的难点，很多公司都没有一个方法去划分微服务，大多情况下都是按照经验和个人的想法去划分，这样出现的一个问题就是服务之间的耦合太严重，为了消除这些耦合引入了大量的冗余，之后又为这些冗余作数据同步，这样的微服务可能还没有单体应用开发的方便

对于逻辑架构我们应该遵循领域驱动设计的原则，我们需要为整个系统进行建模，然后寻找一些关注点去划分我们的上下文，并绘制上下文图，在划分上下文时需要考虑上下文之间的通信的成本，开发人员之间沟通的成本，以及一些特殊的需求需要独立划分上下文，这样我们在绘制好上下文图之后，我们才能从整体去了解各个上下文之间的依赖，并通过调整上下文边界去减少这些依赖，最后我们通过上下文去构建各个微服务

最后我想说的是，引入微服务将会加大开发人员和运维人员的技术要求，如果较多的人员达不到技术要求，则会带来一个糟糕的结果，就是系统会混乱不堪，我的建议是如果系统没有复杂到需要使用微服务去解耦，那么我们没必要使用微服务，我们只需要做好上下文的划分，在需要时，再将系统改造成微服务