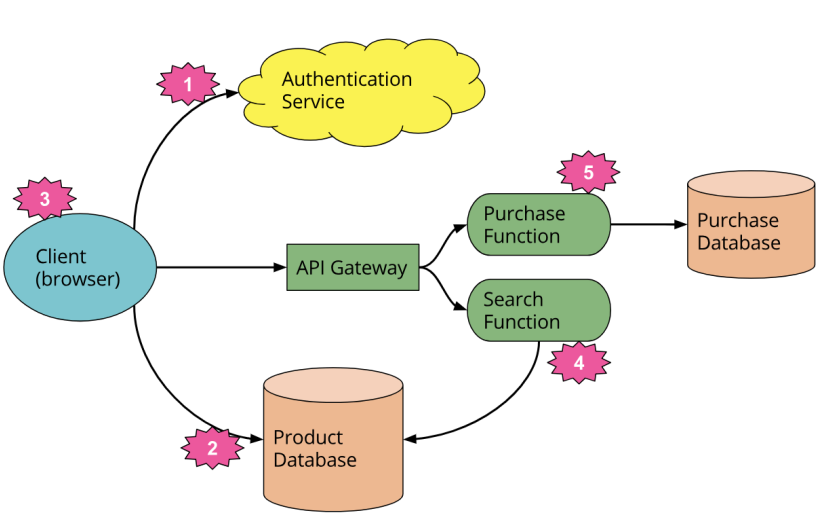
1 传统体系结构和无服务器体系结构

- 传统的体系结构: 一套典型的系统设计项目包括逻辑设计、技术设计和软件体系结构。逻辑设计通常定义的是系统的功能和用途，而技术设计则通常定义的是系统概念。
- 无服务器体系结构：着你迁移至无服务器体系结构，将更为关注系统的用途，而不是系统的概念. 在平台即服务 (PaaS) 环境中，你将重心放在函数属性上，而将配置详细信息交由平台处理。输入的配置详细信息将保存在模板中，并且所有对话都将以功能和集成为重心，无需讨论 RAM、CPU 和磁盘的最佳容量。

1.1 例 1

考虑一个带有服务器端逻辑的传统的三层面向客户端的系统。一个例子是典型的电子商务应用程序 - 在线宠物商店。架构像这样：

在这个架构里，客户端可以相对不用那么智能，绝大多数的逻辑在服务端完成，授权，页面导航，查询，交易等等。在无服务架构里，看起来会是这个样子：

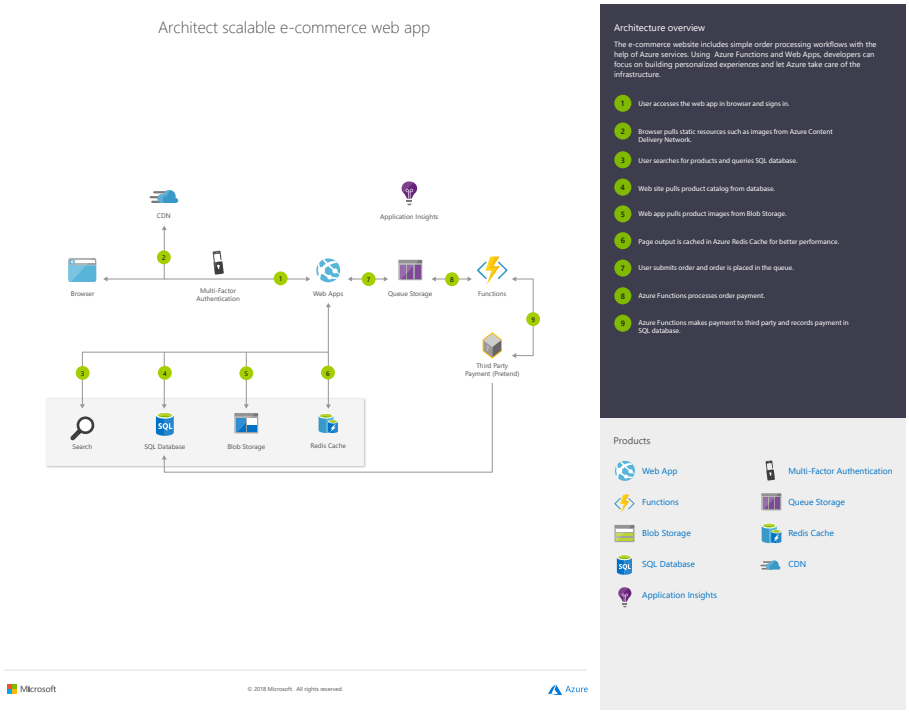


二者对比中，我们可以看到：

- 服务器没了，取而代之的是 Function(Purchase Funciton 和 Search Function). 于是开发者不用再去部署服务器. 在 PaaS 环境中，开发者将定义的 Function(Purchase Funciton 和 Search Function) 上传到 PaaS 提供商，提供商执行配置资源，实例 VM，管理流程等所需的一切。

1.2 例 2

微软官方的电子商务的架构, 不用去考虑服务器的部署，只用考虑设计 Funtions 的逻辑:



2 Azure Functions 的架构

Azure Functions 提供无服务器体系结构的无服务器计算组件。如1 所示，Azure Functions 建立在 Azure App Service 和 WebJobs SDK 的基础之上.

图 1: 图 2

3 Azure Function

Azure Functions 允许你运行小段代码（称为“函数”）且不需要担心应用程序基础结构。借助 Azure Functions，云基础结构可以提供应用程序保持规模化运行所需的所有最新状态的服务器。

功能：

Azure Functions 的一些主要功能包括：

- 无服务器应用程序：使用 Functions，可在 Microsoft Azure 上开发无服务器应用程序。
- 语言选择：使用所选的 C#、Java、JavaScript、Python 和 PowerShell 编写函数。
- 自带依赖项：Functions 支持 NuGet 和 NPM，允许你访问你喜欢的库。

- 集成的安全性：使用 OAuth 提供程序（如 Azure Active Directory、Facebook、Google、Twitter 和 Microsoft 帐户）保护 HTTP 触发的函数。
- 简化的集成：轻松与 Azure 服务和软件即服务 (SaaS) 产品/服务进行集成。
- 灵活开发：直接在门户中编写函数代码，或者通过 GitHub、Azure DevOps Services 和其他受支持的开发工具设置持续集成和部署代码。
- 有状态无服务器体系结构：使用 Durable Functions 协调无服务器应用程序。
- 开放源代码：Functions 运行时是开源的，可在 GitHub 上找到。

使用 Functions 可以做什么？

Functions 是一个理想的解决方案，用于处理批量数据、集成系统、使用物联网 (IoT) 以及生成简单的 API 和微服务。有一系列模板可帮助你开始使用关键方案，包括：

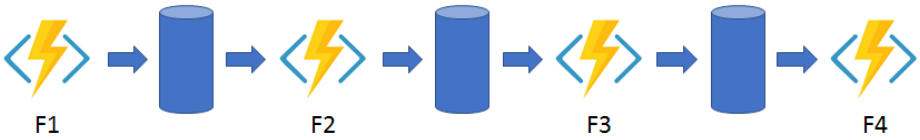
- HTTP：基于 HTTP 请求运行代码
- 计时器：将代码安排在预定义的时间运行
- Azure Cosmos DB：处理新的和修改的 Azure Cosmos DB 文档
- Blob 存储：处理新的和修改的 Azure 存储 blob
- 队列存储：响应 Azure 存储队列消息
- 事件网格：通过订阅和筛选器响应 Azure 事件网格事件
- 事件中心：响应大量 Azure 事件中心事件
- 服务总线队列：通过对服务总线队列消息做出响应连接到其他 Azure 服务或本地服务
- 服务总线主题：通过对服务总线主题消息做出响应连接到其他 Azure 服务或本地服务

4 Azure Durable Function

Durable Functions 是 Azure Functions 的一个扩展，可用于在无服务器计算环境中编写有状态函数。有几种典型的应用模式：

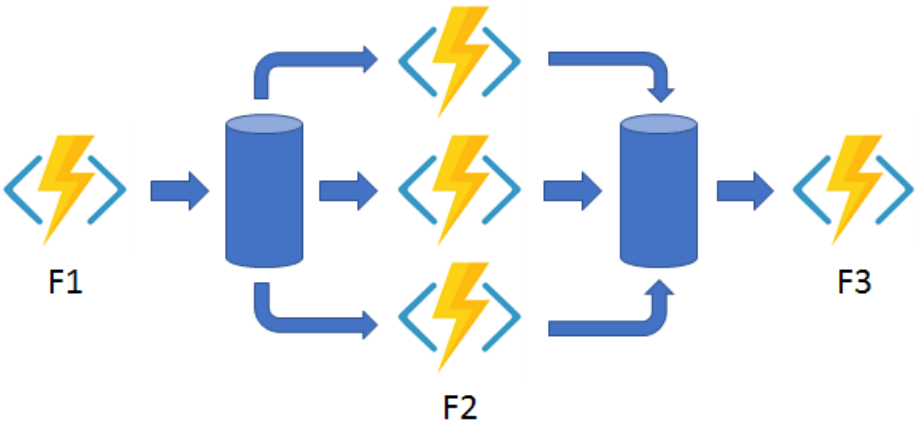
4.1 模式 1：函数链

在函数链接模式中，将按特定顺序执行一系列函数。在此模式中，一个函数的输出将应用到另一函数的输入。



4.2 模式 2：扇出/扇入

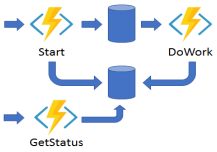
在扇出/扇入模式中，将会并行执行多个函数，然后等待所有函数完成。通常会对这些函数返回的结果执行一些聚合操作。



4.3 模式 3：异步 HTTP API

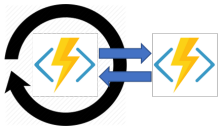
异步 HTTP API 模式解决了使用外部客户端协调长时间运行的操作的状态时出现的问题。实现此模式的一种常用方式是让 HTTP 终结点触发长

时间运行的操作。然后，将客户端重定向到某个状态终结点，客户端可轮询该终结点，以了解操作是何时完成的。



4.4 模式 4：监视

监视模式是指工作流中的灵活重复进程。例如，轮询到满足特定的条件为止。可以使用常规计时器触发器解决简单方案（例如定期清理作业），但该方案的间隔是静态的，并且管理实例生存期会变得复杂。可以使用 Durable Functions 创建灵活的重复间隔、管理任务生存期，以及从单个业务流程创建多个监视进程。监视模式的一个例子是反转前面所述的异步 HTTP API 方案。监视模式不会公开终结点供外部客户端监视长时间运行的操作，而是让长时间运行的监视器使用外部终结点，然后等待某个状态发生更改。

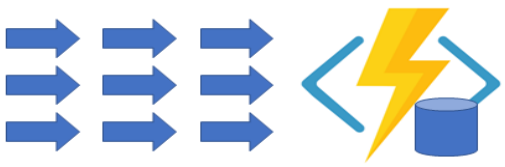


4.5 模式 5：人机交互

许多自动化过程涉及到某种人机交互。自动化过程中涉及的人机交互非常棘手，因为人的可用性和响应能力不如云服务那样高。自动化过程允许使用超时和补偿逻辑来实现这种交互。审批过程就是涉及到人机交互的业务过程的一个例子。例如，某份超出特定金额的开支报表需要经理的审批。如果经理未在 72 小时内审批该开支报表（经理可能正在度假），则会启动上报过程，让其他某人（可能是经理的经理）审批。

4.6 模式 6：聚合器（有状态实体

第六种模式涉及到将一段时间的事件数据聚合到单个可寻址的实体。在此模式中，要聚合的数据可来自多个源、可分批传送，也可以分散在较长的时间段。聚合器可能需要在事件数据抵达时对其执行操作，而外部客户端可能需要查询聚合数据。



参考:

1.https://blog.csdn.net/xialingming/article/details/81369624

2.https://docs.microsoft.com/zh-cn/archive/msdn-magazine/2017/february/azure-serverless-architecture-wihazure-functions#%E4%BC%A0%E7%BB%9F%E4%BD%93%E7%B3%BB%E7%BB%93%E6%9E%84%E4%B8%8E%E6%97%A0%E6%9C%8D%E5%8A%A1%E5%99%A8%E4%BD%93%E7%B3%BB%E7%BB%93%E6%9E%84

3.https://docs.microsoft.com/zh-cn/azure/azure-functions/durable/durable-functions-overview?tabs=csharp