Building Serverless Applications with Python

Serverless架构应用开发
Python实现

[印] 贾莱姆·拉杰·罗希特 著
安 翔 译

- 在Serverless浪潮里了解云计算的技术趋势
- 手把手教你用Python构建无须专人托管的服务器

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Serverless 架构应用开发
Python实现

[印] 贾莱姆·拉杰·罗希特 著
安 翔 译

Building Serverless Applications with Python

人民邮电出版社
北 京

图书在版编目（CIP）数据

Serverless架构应用开发 : Python实现 / (印) 贾莱姆·拉杰·罗希特 (Jalem Raj Rohit) 著 ; 安翔译. -- 北京 : 人民邮电出版社, 2019.8
（图灵程序设计丛书）
ISBN 978-7-115-51724-1

Ⅰ. ①S… Ⅱ. ①贾… ②安… Ⅲ. ①移动终端－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字(2019)第155655号

内 容 提 要

本书主要基于云架构的Python示例来讲解Serverless的概念。Serverless架构的核心思想是函数即服务。这种架构能合理配置闲置资源，无须专门的运维团队成员来维护和管理服务器，因此能节省很多管理费用。本书分为三个模块：第一个模块解释Serverless架构的基本原理以及AWS lambda函数的作用；第二个模块教你构建、发布并部署应用到生产环境；第三个模块将带领你完成高级主题，例如为应用构建Serverless API。你还将学习如何扩展Serverless应用并处理生产中的分布式Serverless系统。在本书的最后，你将能够使用Serverless框架构建可扩展的高效Python应用程序。

本书适合希望了解云平台中Serverless架构的Python开发人员阅读。

◆ 著　[印] 贾莱姆·拉杰·罗希特
译　安　翔
责任编辑　温　雪
责任印制　周昇亮

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：12
字数：284千字　2019年8月第1版
印数：1－3 500册　2019年8月北京第1次印刷
著作权合同登记号　图字：01-2018-7610号

定价：59.00元

读者服务热线：(010)51095183转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字20170147号

作者简介

贾莱姆·拉杰·罗希特
（Jalem Raj Rohit）

目前在GEP Worldwide担任数据科学家，专注于机器学习、DevOps和产品开发等领域。他为Python、Go和Julia都贡献过开源项目，还在有关Serverless工程和机器学习的技术会议上发表过演讲。

译者简介

安翔

Dell EMC存储软件工程师。InfoQ社区编辑，CSDN译者。著有《物联网Python开发实战》一书，擅长Python与物联网产品开发的结合。

封面设计 | 83720326@qq.com
自在书装设计

微信连接

回复“运维”查看相关书单

微博连接

关注@图灵教育 每日分享IT好书

QQ连接

图灵读者官方群I：218139230
图灵读者官方群II：164939616

图灵社区
iTuring.cn

在线出版，电子书，《码农》杂志，图灵访谈

站在巨人的肩上

Standing on Shoulders of Giants

iTuring.cn

版权声明

前　言

Serverless 是一个全新的计算机工程领域，它让开发人员专注于编写代码和部署基础设施，而不必在服务器的维护上浪费精力。本书主要基于云架构的 Python 示例来讲解 Serverless 的概念。

读者对象

本书主要为那些想要了解云平台（比如 Azure 和 AWS）上 Serverless 架构的 Python 开发人员而写。因此，要想更好地阅读本书，了解基本的 Python 编程知识是必不可少的。

本书内容

第 1 章，Serverless 范式，介绍微服务和 Serverless 架构的基本概念，并明确列出 Serverless 架构的优缺点。

第 2 章，在 AWS 中构建 Serverless 应用程序，详细介绍 AWS Lambda 的概念、工作原理以及组件，并具体解释 Lambda 的安全性、用户控制和版本控制。

第 3 章，构建 Serverless 架构，进一步介绍 AWS Lambda 中的各种触发器以及它们与函数集成的方法。通过阅读这一章，读者将了解每个触发器的事件结构，还将学会根据使用的触发器类型修改 Lambda 函数。

第 4 章，部署 Serverless API，带领读者探索 AWS API 网关，并利用 API 网关和 Lambda 构建出高效且可扩展的 Serverless API。此外，这一章还将展示如何通过添加身份验证改进 API，以及如何通过限制请求等方法设置用户级别的控制。

第 5 章，日志与监控，介绍 Serverless 应用程序中日志和监控的概念，这仍然是该领域一个尚未解决的问题。这一章带领读者在 AWS 环境中，用 Python 通过自定义指标和日志构建日志和监控系统；此外，还将详细介绍在 Python 中对 Lambda 函数进行日志记录和监控的最佳实践。

第 6 章，扩展 Serverless 架构，讨论如何使用多个第三方工具扩展 Serverless 架构以应对高负载，并且介绍如何利用现成的 Python 模块来保证安全性，以及提供日志和监控功能。

第 7 章，AWS Lambda 的安全性，教读者利用 AWS 自带的安全功能部署安全的 Serverless 应用程序。这涉及严格控制应用程序可以访问的组件，以及有权访问和操作应用程序的用户。读者还将了解 AWS 虚拟私有云和子网，以便理解可以在 AWS Lambda 中遵循的安全功能和最佳实践。

第 8 章，使用 SAM 部署 Lambda 函数，介绍如何通过 Serverless 应用程序模型将 Lambda 函数部署为基础设施即代码。这是一种编写和部署 Lambda 函数的新方法，使得与其他 IaaS 服务（比如 CloudFormation）集成变得更简单。

第 9 章，微软 Azure Functions 简介，带领读者熟悉微软 Azure Functions，并解释该工具的组件及其配置方法。

阅读前提

为了更好地阅读本书，读者应该对 Python 编程语言有基本的了解。如果对云平台也很熟悉，那再好不过了。

排版约定

本书采用如下排版约定。

等宽字体：表示代码、数据库表名、用户输入。例如：“所有的 SAM 都需要元信息，包括 `AWSTemplateFormatVersion` 和 `Transform`。它会让 `CloudFormation` 知道你所编写的是 AWS SAM 代码，并且是一个 Serverless 应用程序。”

代码段的样式如下：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
```

黑体字：表示新术语、重点内容，或者界面上显示的内容。比如，菜单或者对话框中的内容就会用黑体字表示。例如：“要想创建一个函数，需要单击页面右侧橙色的 **Create a function**（创建一个函数）按钮。”

此图标表示警告或者重要信息。

此图标表示提示或者小技巧。

保持联系

我们始终欢迎读者的反馈。

一般反馈：发送邮件至 feedback@packtpub.com 并在邮件主题中注明书名。如果对本书有任何疑问，请发送邮件至 questions@packtpub.com。

勘误：尽管我们已尽全力来保证本书内容的准确性，但错误在所难免。假如你发现书中有错，请告知我们，我们将非常感激。请访问 https://www.packtpub.com/support，单击 Support Errata 选项卡，选择图书，然后输入勘误详情。[①]

反盗版：如果你在网上发现以任何形式复制我们作品的非法行为，请立即将地址或网站名告知我们，我们将非常感谢。请联系 copyright@packtpub.com 提供有盗版嫌疑的链接。

成为作者：如果你是某个领域的专家，并且有兴趣编写图书，请访问 authors.packtpub.com。

评论

请留下你的评论。阅读并使用本书之后，为什么不在购买网站上发表评论呢？其他读者可以参考你的评价来做出购买决定，Packt 可以了解你对我们产品的看法，作者也能看到你对本书的反馈。谢谢！

想了解关于 Packt 的更多信息，请访问 https://www.packtpub.com。

电子书

扫描如下二维码，即可购买本书电子版。

① 本书中文版勘误，请到 http://www.ituring.com.cn/book/2648 查看和提交。——编者注

目　录

第 1 章 Serverless 范式

在阅读本书之前，你可能已经对 Serverless 有所耳闻，比如听说过 Serverless 范式、Serverless 工程以及 Serverless 架构。如今，随着**事件驱动的架构设计**（又称 **Serverless 架构**）盛行，开发人员部署应用程序的方式已经发生了巨大的变化，在数据工程和 Web 开发领域尤甚。

在服务器工作负载的间隙，将闲置资源和服务器置于生产闲置状态的情况并不少见，但这会造成基础设施的极大浪费。如果工作负载之外的其他时间不需要闲置资源怎么办？如果可以在必要时创建资源并在工作完成后将资源销毁呢？

阅读本章内容之后，你将了解 Serverless 架构和函数即服务的工作原理，以及如何将它们构建到现有的软件架构中。你还将了解什么是微服务，并且能够判断 Serverless 和微服务是否适用于你的体系结构，并学会在主流的云平台（比如**亚马逊的 AWS** 以及**微软的 Azure**）上使用 Python 构建 Serverless 应用程序。

本章包括以下内容：

- 了解 Serverless 架构
- 了解微服务
- Serverless 架构不仅仅是实时的
- Serverless 架构的优缺点

1.1 了解 Serverless 架构

Serverless 架构或者 Serverless 工程的核心思想是函数即服务。从技术角度来讲，互联网上最准确的“Serverless 计算”的定义如下：

> “Serverless 计算又称为**函数即服务**（FAAS），它是一种云计算和代码执行模型，其中云提供商管理函数的容器——**平台即服务**（PaaS）——的启动和停止。”

让我们仔细研究上述定义的每个部分，从而更好地理解 Serverless 计算的范式。先来了解“函

数即服务”这个术语。它意味着任何 Serverless 模型都有一个在云平台上运行的函数。这些函数只不过是代码块，它们的执行取决于与之相关联的触发器。下图展示了 AWS Lambda 环境中的所有触发器。

- Amazon S3
- Amazon DynamoDB
- Amazon Kinesis Streams
- Amazon Simple Notification Service
- Amazon Simple Email Service
- Amazon Cognito
- AWS CloudFormation
- Amazon CloudWatch Logs
- Amazon CloudWatch Events
- AWS CodeCommit
- Scheduled Events (powered by Amazon CloudWatch Events)
- AWS Config
- Amazon Alexa
- Amazon Lex
- Amazon API Gateway
- Other Event Sources: Invoking a Lambda Function On Demand
- Sample Events Published by Event Sources

接下来了解一下函数启动和停止的管理机制。只要有触发器触发了某个函数，云平台就会启动一个容器，用来执行该函数。一旦该函数执行成功并返回结果，或者运行超时，那么运行该函数的容器就将被云平台回收或者销毁。在高负载的情况下，或者当两个触发器之间几乎不存在时间间隔时，这种回收机制使得容器能够被重复利用。

再来看看上述“Serverless 计算”定义的后半部分，即函数的容器。这意味着函数在容器中启动和执行。Docker 公司将“容器”的概念发扬光大，它对“容器”的标准定义为：

> “容器映像是一个轻量级、独立且可执行的软件包，其中包含了软件运行所需的一切：代码、运行时、系统工具、系统库、设置。”

容器的作用是将函数的代码、运行时环境等打包到单个部署包中，以实现无缝执行。**部署包**中包含了该函数的主代码文件，以及执行该函数所需的所有非标准库。部署包的创建过程与 Python 虚拟环境的创建过程非常相似。

因此，我们可以明确地指出，对于采用 Serverless 架构的应用程序，其服务器不会一直运行。其好处显而易见，那就是无须专门的运维团队成员来维护和管理服务器。节省出来的人力可以专注于其他事情，比如软件研发等。同时，这也为公司和个人节省了资源和成本。对于那些经常使用 GPU 实例应对高负载的机器学习和数据工程团队来说，好处就更加明显了。因此，按需运行的 Serverless GPU 实例无须开发人员或者运维团队全天候维护，从而能够节省一大笔资金。

1.2 了解微服务

与 Serverless 的概念类似，面向微服务的设计策略最近也非常流行。这种架构设计在 Serverless 的概念出现以前就已经存在很长时间了。就像我们试图通过互联网上的技术定义来理解 Serverless 架构一样，我们也应通过互联网上的技术定义来理解微服务。微服务的技术定义如下：

> “微服务又称为**微服务架构**，它是一种架构风格，将应用程序构建为一组松耦合的服务，以实现业务功能。”

与 Serverless 架构一样，规划和设计微服务形式的架构同样有利有弊。我们要熟知微服务架构的优缺点，以便使用时能够扬长避短。在阐述微服务的缺点之前，先来看看它的优点。

微服务能够帮助软件团队保持敏捷和逐步改进。简单来说，由于各个服务之间彼此分离，因此升级和改进一项服务非常容易，并且不会导致其他服务崩溃。例如，在社交网络软件中，如果聊天和订阅功能都是微服务，那么当软件团队尝试升级或者修复聊天服务时，订阅服务完全不会受影响。然而，在大型整体式系统中，很难像微服务那样将功能独立开来。因此，在整体式架构中，即使是一个小组件的修复或者升级都会导致所有服务的停止，而修复所需的时间也往往超出预期。

整体式架构的代码量非常庞大，任何一个小故障都会影响整个系统的运转。微服务则对代码进行了精简和细分，从而极大地提升了开发人员的工作效率。开发人员可以在开销极小甚至为零并且不需要停机的情况下修复和改进服务。容器可以让我们更好地利用微服务，它提供了有效且完整的虚拟操作系统环境，能够隔离进程，并且提供底层硬件资源的专属访问权限。

当然，微服务也有缺点，其中最主要的一点是，它依赖分布式系统。由于各个服务之间是相互独立的，所以架构师需要弄清楚各个服务之间的交互方式，从而构建出功能完整的产品。因此，服务之间的交互方式，以及它们之间数据的传输策略，都是架构师需要认真考虑的问题。分布式系统的主要问题，比如**共识**、**CAP 定理**、**维持共识的稳定性**以及**连接**，都是工程师在构建微服务时需要处理的问题。确保和维护安全性也是分布式系统和微服务中的主要问题。你需要为每个微服务制定单独的安全模式和层级，还需要为服务之间的数据交互制定安全策略。

1.3 Serverless 架构不仅仅是实时的

由于 Serverless 架构以**函数即服务**的形式运行，而函数由一组可用的触发器来触发，所以 Serverless 架构经常被用作实时系统。但是，单纯将 Serverless 架构看作实时系统是一种常见的误解，因为 Serverless 系统除了可用作实时系统之外，也适用于批处理架构。并不是所有的研发团队都需要使用或者拥有实时系统，因此学会将 Serverless 系统用作批处理架构会给研发团队带来更多的可能性。

可以通过如下方式将 Serverless 系统用作批处理架构：

- 触发器的 cron 功能
- 队列

首先来了解一下触发器的 **cron 功能**。我们可以在云平台的 Serverless 系统中设置监控器，将监控器设置为一个普通的 cron 任务，该监控器便会每隔几分钟或者几小时触发一次触发器。这有助于把 Serverless 配置为 cron 批处理任务。在 AWS 环境中，Lambda 可以通过 AWS CloudWatch 作为 cron 触发，为此，可以手动输入时间间隔来设置 cron 任务的频率，也可以在 cron 格式中选择间隔。

Example	Cron expression
Invoke Lambda function every 5 minutes	Copy `rate(5 minutes)`
Invoke Lambda function every hour	Copy `rate(1 hour)`
Invoke Lambda function every seven days	Copy `rate(7 days)`

还可以利用队列来构建 Serverless 批处理架构。让我们设置一个示例数据管道，以此来理解队列的概念。假设我们要构建一个系统来执行以下任务。

(1) 用户或者服务将一些数据发送到数据库或者更简单的数据存储系统中，例如 AWS 的 S3。

(2) 一旦数据存储系统中的文件数量超过 100，就执行一项任务。比如，对文件进行一些分析，例如计算文件的总页数。

该系统可以通过队列来实现。这个例子是一个相对简单的 Serverless 系统。它可以通过以下方式实现。

(1) 用户或者服务将数据上传或者发送到我们选好的数据存储系统中。

(2) 为这个任务配置一个队列。

(3) 为 S3 存储桶或者数据存储系统配置一个事件，这样，只要有数据进来，就向上一步配置的队列中发送消息。

(4) 设置监控系统以监控队列中的消息数量。建议你使用云提供商的监控系统，以便保持系统完全 Serverless 化。

(5) 为监控系统设置告警，并配置告警阈值。例如，一旦队列中的消息数量达到或者超过 100 就触发告警。

(6) 该告警可以扮演 Lambda 函数触发器的角色。Lambda 函数首先接收队列消息，然后使用消息中的文件名来查询数据存储系统，以此对文件进行分析。

(7) 文件由 Lambda 函数分析之后，将被发送到另外一个数据存储系统中存放。

(8) 所有任务完成之后，运行 Lambda 函数的容器或者服务器将自动终止，因此整个传输流程完全 Serverless 化。

1.4 Serverless 的优缺点

至此，我们已经对 Serverless 架构和流程有了大致的了解，知道了如何在现有架构中利用它们，还学会了如何使用微服务来简化架构以及提升开发人员的工作效率。本节将详细列举 Serverless 系统的优缺点，以便软件开发人员和架构师决定何时在现有系统中利用 Serverless 范式。

Serverless 系统的优点如下。

- **降低了运营成本**：部署了 Serverless 系统，服务器就不再需要昼夜不停地运转，因此设备成本得以大幅缩减。当函数被触发时服务器才会启动，而当函数执行完毕时服务器会自动停止，因此用户只需为函数运行的时间段付费。
- **减少了维护工作**：鉴于以上情况，我们不再需要对服务器进行持续不断的监控和维护。由于函数和触发器高度自动化，因此 Serverless 系统几乎不需要维护。
- **提升了开发效率**：由于开发人员不需要操心服务器的维护工作以及宕机情况的发生，因而可以专注于提升软件质量的相关工作，例如扩展和设计功能。

本书后面的章节会展示 Serverless 系统如何改变软件的构建方式。本章旨在帮助架构师判断 Serverless 系统对于他们的架构来说是否是一个好的选择。接下来看看 Serverless 系统的缺点。

Serverless 系统的缺点如下。

- **函数运行有时限**：无论是 AWS Lambda 还是 GCP 云，其函数运行时长的上限都是 5 分钟，这使得计算密集型的任务变得难以运行。为了解决这个问题，可以用 nohup 模式来执行配置工具的脚本。相关内容将在本章后面详细介绍。然而，配置脚本、设置容器以及其他工作也必须在 5 分钟内完成。一旦超过 5 分钟的时限，容器便会自动终止。