Amazon Q Developer 在线生成和助手代码生成的最佳实践

AWS 规范性指导

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS 规范性指导: Amazon Q Developer 在线生成和助手代码生成的最佳实践

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆或者贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

简介 1
目标 1
开发人员工作流程 3
设计和规划 4
编码 4
代码审查 4
集成和部署 5
高级功能 6
Amazon Q 开发者代码转换 6
Amazon Q 开发者自定义设置 6
最佳编码实践 8
入职最佳实践 8
Amazon Q 开发者的先决条件 8
使用 Amazon Q 开发者时的最佳实践 8
Amazon Q 开发者中的数据隐私和内容使用情况 9
代码生成的最佳实践 9
代码推荐的最佳实践 10
代码示例 12
Python 示例 12
生成类和函数 12
文件代码 13
生成算法 14
生成单元测试 17
Java 示例 18
生成类和函数 18
文件代码 21
生成算法 23
生成单元测试 25
聊天示例 26
询问 AWS 服务 26
生成代码 27
生成单元测试 29
解释代码 31
故障排除 34

空代码生成 34
持续评论 35
行内代码生成不正确 36
聊天结果不足 41
FAQs 45
什么是 Amazon Q Developer？ 45
我如何访问亚马逊 Q 开发者？ 45
Amazon Q 开发者支持哪些编程语言？ 45
如何向 Amazon Q Developer 提供背景信息，以便更好地生成代码？ 45
如果使用 Amazon Q Developer 生成内嵌代码不准确，我该怎么办？ 45
如何使用 Amazon Q Developer 聊天功能生成代码和进行故障排除？ 46
使用 Amazon Q 开发者的一些最佳实践？ 46
我能否自定义 Amazon Q Developer 以根据自己的代码生成推荐？ 46
后续步骤 47
资源 48
AWS 博客 48
AWS 文档 48
AWS 工作坊 48
贡献者 49
文档历史记录 50
术语表 51
..... 51
A 51
B 54
C 55
D 58
E 61
F 63
G 64
H 65
我 66
L 68
M 69
O 73
P 75
Q 77

R 78
S 80
T 83
U 84
V 85
W 85
Z 86
..... lxxxvii

Amazon Q Developer 在线生成和助手代码生成的最佳实践

亚马逊 Web Services (贡献者)

2024 年 8 月 () 文档历史记录

传统上，开发人员依靠自己的专业知识、文档和来自各种来源的代码片段来编写和维护代码。尽管这些方法为行业提供了很好的服务，但它们可能很耗时并且容易出现人为错误，从而导致效率低下和潜在的错误。

这是 Amazon Q 开发者介入改善开发者旅程的地方。Amazon Q Developer 是一款功能强大的 AWS 生成式人工智能助手，旨在通过提供智能代码生成和建议来加快代码开发任务。

但是，与任何新技术一样，也可能存在挑战。不切实际的期望、入门困难、对不准确的代码生成进行故障排除以及正确使用 Amazon Q 功能是开发人员可能面临的常见障碍。这份全面的指南解决了这些挑战，提供了现实生活中的场景、详细的最佳实践、故障排除以及专门针对以下方面的实际代码示例 Python 以及 Java，两种最广泛采用的编程语言。

本指南重点介绍如何使用 Amazon Q Developer 来执行代码开发任务，例如：

- 代码完成-在开发人员实时编码时生成行内建议。
- 代码改进和建议 — 讨论软件开发，使用自然语言生成新代码，并改进现有代码。

目标

本指南的目标是为新用户或长期使用 Amazon Q Developer 的开发者提供支持，帮助他们在日常编码任务中成功使用该服务。开发团队经理也可以从阅读本指南中受益。

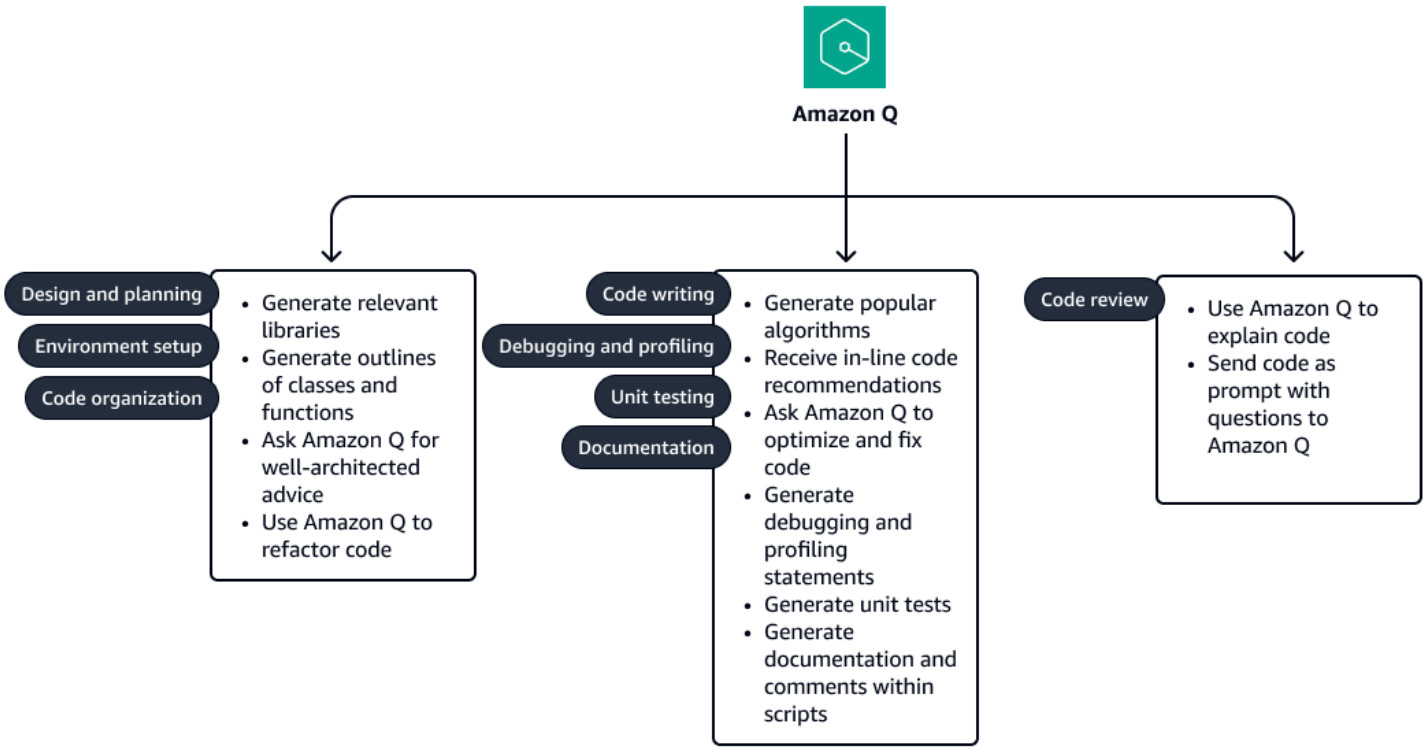
本指南为您提供有关使用 Amazon Q 开发者的以下见解：

- 了解如何有效使用 Amazon Q Developer 进行代码开发
  - 提供将 Amazon Q 开发者集成到开发者工作流程中的最佳实践。
  - 提供 step-by-step 指导和成功生成代码的示例，并提供建议。
- 缓解常见挑战，提高开发者使用 Amazon Q Developer 的清晰度
  - 提供策略和见解，以满足开发者的期望，克服与代码生成准确性和性能相关的障碍。
- 提供故障排除和错误处理
  - 为开发者提供 Amazon Q Developer 代码生成疑难解答指南，以解决不准确的结果或意外行为。

  - 提供特定于以下内容的真实示例和场景 Python 以及 Java.
- 优化工作流程和生产力
  - 使用 Amazon Q 开发人员优化代码开发工作流程。
  - 讨论提高开发人员工作效率的策略。

使用 Amazon Q Developer

开发人员遵循标准工作流程，包括需求收集、设计和规划、编码、测试、代码审查和部署等阶段。本节重点介绍如何使用 Amazon Q 开发人员功能来优化关键开发步骤。



上图显示了 Amazon Q Developer 如何在代码开发阶段加速和简化以下常见任务：

- 设计和规划 | 环境设置 | 代码组织
  - 生成相关库
  - 生成类和函数的轮廓
  - 向 Amazon Q 询问精心设计的建议
  - 使用 Amazon Q in Conder
- 代码编写 | 调试和分析 | 单元测试 | 文档
  - 生成常用算法
  - 接收行内代码推荐
  - 让 Amazon Q 优化和修复代码
  - 生成调试和分析语句
  - 生成单元测试

  - 在脚本中生成文档和注释
- 代码审查
  - 让 Amazon Q 解释一下代码
  - 如有问题，请按提示将代码发送至 Amazon Q

设计和规划

收集业务和技术需求后，开发人员设计新的代码库或扩展现有的代码库。在此阶段，Amazon Q 开发者可以协助开发者完成以下任务：

- 生成相关的库以及类和函数大纲，以获得架构良好的建议。
- 为工程、兼容性和架构设计查询提供指导。

编码

编码过程使用 Amazon Q Developer 通过以下方式加快开发速度：

- 环境设置- AWS Toolkit 在您的集成开发环境中安装 (IDE)（例如，VS Code 或 IntelliJ）。然后，使用 Amazon Q 生成库或根据您的项目目标接收设置建议。有关更多详细信息，请参阅 Amazon Q 开发人员入职的最佳实践。
- 代码组织-重构代码或从 Amazon Q 获取与您的项目目标一致的组织建议。
- 代码编写- 在开发时使用内嵌建议生成代码，或者使用您IDE的 Amazon Q 聊天面板让 Amazon Q 生成代码。有关更多详细信息，请参阅 Amazon Q 开发人员生成代码的最佳实践。
- 调试和分析- 生成分析命令，或使用 Amazon Q 选项（例如 “修复” 和 “解释”）来调试问题。
- 单元测试-在聊天会话期间向 Amazon Q 提供代码作为提示，并请求生成适用的单元测试。有关更多信息，请参阅 Amazon Q 开发者的代码示例。
- 文档-使用行内建议创建注释和文档字符串，或者使用 “解释” 选项生成代码选择的详细摘要。有关更多信息，请参阅 Amazon Q 开发者的代码示例。

代码审查

审阅者需要先理解开发代码，然后才能将其推广到生产环境。要加快此过程，请使用 Amazon Q 的 “解释” 和 “优化” 选项，或者在聊天会话中将带有自定义提示说明的代码选择发送给 Amazon Q。有关更多信息，请参阅聊天示例。

集成和部署

向 Amazon Q 寻求有关持续集成、交付渠道和特定于您的项目架构的部署最佳实践的指导。

使用这些建议，您可以学习如何有效利用 Amazon Q Developer 功能，在整个开发生命周期中优化工作流程并提高工作效率。

Amazon Q 开发者的高级功能

尽管本指南侧重于在动手编程任务中使用 Amazon Q Developer，但请务必了解其以下高级功能：

- Amazon Q 开发者代码转换
- Amazon Q 开发者自定义设置

Amazon Q 开发者代码转换

用于代码转换的 Amazon Q 开发者代理可以升级文件的代码语言版本，而无需您手动重写代码。它的工作原理是分析您现有的代码文件并自动重写它们以使用该语言的较新版本。例如，如果你在像 IDE 这样的 IDE 中工作，Amazon Q 会转换单个模块 Eclipse。 如果你使用的是 Visual Studio 代码，Amazon Q 可以转换整个项目或工作空间。

如果您想执行常见的代码升级任务，请使用 Amazon Q，例如：

- 更新代码以使用语言版本的新语法。
- 运行单元测试以验证编译和执行是否成功。
- 检查并解决部署问题。

Amazon Q 可以为开发人员节省数天到数月的繁琐重复工作，以升级代码库。

自 2024 年 6 月起，亚马逊 Q 开发者支持升级 Java 代码并且可以转换 Java 8 个新版本的代码，例如 Java 11 或 17。

Amazon Q 开发者自定义设置

凭借其自定义功能，Amazon Q Developer 可以根据公司自己的代码库提供内联建议。该公司将其代码存储库提供给亚马逊简单存储服务 (Amazon S3)，也可以 AWS CodeConnections通过（以前称为 AWS CodeStar Connections）提供。然后，Amazon Q 使用启用安全功能的自定义代码存储库来推荐与该组织中的开发人员相关的编码模式。

使用 Amazon Q 开发者自定义设置时，请注意以下几点：

- 自 2024 年 6 月起，Amazon Q 开发者自定义功能处于预览模式。因此，该功能的可用性和支持可能会受到限制。

- 只有考虑到所提供的代码存储库的质量，自定义行内代码建议才是准确的。我们建议您查看您创建的每个自定义项的评估分数。
- 为了优化性能，我们建议您包含至少 20 个包含给定语言的数据文件，其中所有源文件都大于 10MB。确保您的存储库由可引用的源代码组成，而不是元数据文件（例如，配置文件、属性文件和自述文件）。

通过使用 Amazon Q 开发者自定义设置，您可以通过以下方式节省时间：

- 使用基于您自己的公司专有代码的建议。
- 提高现有代码库的可重用性。
- 创建可重复的模式，这些模式在整个公司中普遍适用。

Amazon Q 开发者的最佳编码实践

本节讨论使用 Amazon Q 开发人员进行编码的最佳实践。最佳实践包括以下几类：

- 入职-入职时的方法和注意事项
- 代码生成-成功使用代码生成指南
- 代码建议-改进代码的技巧

入职 Amazon Q 开发者的最佳实践

Amazon Q Developer 是一款功能强大的生成式人工智能编码助手，可通过 IDEs 诸如 Visual Studio Code 等热门产品获 JetBrains得 本节重点介绍访问 Amazon Q Developer 并将其引导到您的编码开发环境的最佳实践。

Amazon Q 开发者的先决条件

Amazon Q Developer 作为 AWS Toolkit for Visual Studio Code 和的一部分
提供 AWS Toolkit for JetBrains （例如，IntelliJ 和 PyCharm）。对于 Visual
Studio Code 和 JetBrains IDEs，Amazon Q Developer 支持 Python JavaScript
TypeScript、Java、、、、C#、Go、Rust、PHP、Ruby、Kotlin、C、C++、Shell 脚本、SQL 和 Scala。

有关安装适用 AWS Toolkit 于 Visual Studio 代码和 JetBrains IDE 的详细说明，请参阅《Amazon Q 开发者用户指南》中的 “在 IDE 中安装 Amazon Q 开发者扩展或插件”。

使用 Amazon Q 开发者时的最佳实践

使用 Amazon Q Developer 时的一般最佳实践包括以下内容：

- 提供相关的上下文以获得更准确的响应，例如正在使用的编程语言、框架和工具。将复杂的问题分解为较小的组件。
- 根据提示和问题进行实验和迭代。编程通常涉及尝试不同的方法。
- 在接受代码建议之前，请务必查看它们，并根据需要进行编辑，以确保它们完全符合您的预期。
- 使用自定义功能让 Amazon Q Developer 了解您的内部库 APIs、最佳实践和架构模式，以获得更多相关建议。

Amazon Q 开发者中的数据隐私和内容使用情况

在决定使用 Amazon Q Developer 时，您应该了解您的数据和内容是如何使用的。以下是要点：

- 对于 Amazon Q Developer Pro 用户，您的代码内容不会用于服务改进或模型训练。
- 对于 Amazon Q 开发者免费套餐用户，您可以通过 IDE 设置或 AWS Organizations 政策选择不将您的内容用于服务改进。
- 传输的内容经过加密，任何存储的内容都通过静态加密和访问控制进行保护。有关更多信息，请参阅 Amazon Q 开发者用户指南中的 Amazon Q 开发人员中的数据加密。

使用 Amazon Q 开发者生成代码的最佳实践

Amazon Q Developer 提供自动代码生成、自动完成和自然语言代码建议。以下是使用 Amazon Q Developer 内联编码辅助的最佳实践：

- 提供背景信息以帮助提高回复的准确性

  从现有代码开始，导入库，创建类和函数，或者建立代码框架。这种上下文将有助于显著提高代码生成质量。
- 自然编码

  像强大的自动完成引擎一样使用 Amazon Q 开发者代码生成。像往常一样编码，让 Amazon Q 在您键入或暂停时提供建议。如果无法生成代码，或者您遇到代码问题，请通过在电脑上键入 Alt+C 或在 macOS 上键入 Opt ion +C 来启动 Amazon Q。有关在使用内联建议时可以采取的常见操作的更多信息，请参阅 Amazon Q 开发者用户指南中的使用快捷键。
- 包括与脚本目标相关的导入库

  包括相关的导入库，以帮助 Amazon Q 了解上下文并相应地生成代码。您也可以让 Amazon Q 建议相关的进口声明。
- 保持清晰而有针对性的背景

  让你的脚本专注于特定的目标，并将不同的功能模块化成具有相关上下文的单独脚本。避免嘈杂或令人困惑的上下文。
- 尝试使用提示

  探索不同的提示，推动 Amazon Q 在代码生成方面产生有用的结果。例如，尝试以下方法：

  - 使用标准注释块进行自然语言提示。

  - 创建带有注释的骨架以填充类和函数。
  - 在提示中要具体，提供细节而不是概括。
- 与 Amazon Q 开发者聊天并寻求帮助

  如果 Amazon Q 开发者没有提供准确的建议，请在 IDE 中与 Amazon Q 开发者交谈。它可以提供代码片段或完整的类和函数来启动你的上下文。有关更多信息，请参阅《Amazon Q 开发者用户指南》中的 “与 Amazon Q 开发者聊聊代码”。

Amazon Q 开发者推荐代码的最佳实践

Amazon Q Developer 可以回答开发者问题并评估代码，以提供建议，从代码生成和错误修复到使用自然语言的指导。以下是在 Amazon Q 中使用聊天功能的最佳做法：

- 从头开始生成代码

  对于新项目或需要常规功能（例如，从 Amazon S3 复制文件），请让 Amazon Q Developer 使用自然语言提示生成代码示例。Amazon Q 可以提供公共资源的相关链接，以供进一步验证和调查。
- 寻求编码知识和错误解释

  遇到编码问题或错误消息时，请向 Amazon Q Developer 提供代码块（如果适用，附上错误消息）和您的问题作为提示。这种背景将有助于 Amazon Q 提供准确和相关的回复。
- 改进现有代码

  要修复已知错误或优化代码（例如，降低复杂性），请选择相关的代码块，并将其与您的请求一起发送给 Amazon Q Developer。请具体说明您的提示以获得更好的结果。
- 解释代码功能

  浏览新的代码存储库时，请选择一个代码块或整个脚本，然后将其发送给 Amazon Q Developer 进行解释。缩小选区大小以获得更具体的解释。
- 生成单元测试

  发送代码块作为提示后，让 Amazon Q Developer 生成单元测试。这种方法可以节省与代码覆盖率相关的时间和开发成本，以及 DevOps.
- 寻找 AWS 答案

Amazon Q Developer 是开发人员的宝贵资源， AWS 服务 因为它包含大量与之相关的知识 AWS。无论您是面临特定内容的挑战 AWS 服务，还是遇到特定的错误消息 AWS，还是正在尝试学习新知识 AWS 服务，Amazon Q 通常都会提供相关且有用的信息。

请务必查看 Amazon Q 开发者向您提供的建议。然后，进行必要的编辑并执行测试，以确保代码满足您的预期功能。

Amazon Q 开发者的代码示例

本节提供了真实的示例，说明如何改善您的体验和代码生成，重点是 Python 以及 Java 语言。除了内联示例外，一些场景还展示了如何在编程时使用 Amazon Q Developer 的聊天功能来征求代码推荐的提示。

有关更多信息，请参阅以下主题：

- Python 例子
- Java 例子
- 聊天示例

Python 示例

以下 Python 示例演示了如何开始使用 Amazon Q Developer 生成行内代码建议。这些说明性示例向您展示了如何为自己的项目触发编码助手。以下大多数示例适用于其他语言：

- 生成类和函数
- 文件代码
- 生成算法
- 生成单元测试

生成类和函数

要生成全部或部分类实现，请使用代码注释。该评论描述了该班的意图。从简单开始，必要时添加更多细节。生成类后。您可以继续在其骨架中生成函数。

当你写评论时，最好使用该语言的注释结构。In Python，对于单行注释，请使用#。对于多行注释，请使用''' '''或""" """。

输入：

```
'''
DocumentReader class. has a name, children and optional parent.
Includes I/O functions to read and
write document
'''
```

输出：

```
class DocumentReader:
    def __init__(self, name, children, parent=None):
        self.name = name
        self.parent = parent
        self.children = children

    def read(self):
        """Reads current document and returns text"""
        pass

    def write(self, text):
        """Writes text to current document"""
        pass
```

现在有了初始类，可以生成一个带有额外注释的附加函数。

输入:

```
'''
function to read document from local directory path and print the first line of the
 document
'''
```

输出：

```
def read_document(self, path):
        """Reads document from path and prints first line"""
        import os
        if os.path.isfile(path):
            with open(path, 'r') as f:
                print(f.readline())
        else:
            print(f"File not found at {path}")
```

文件代码

依靠内联文档，缩短编写代码的时间。添加 Python 评论中断''' '''。

输入:

```
class S3Utils:
    '''
    <place cursor here>
    '''
    def __init__(self):
        self.base_url = "https://s3"

    def generate_s3_url(self, bucket_name, region):
        return f"{self.base_url}.{region}.amazonaws.com/{bucket_name}"

# Example usage:
if __name__ == "__main__":
    s3_util = S3Utils()
    bucket_name = "my-example-bucket"
    region = "us-west-1"
    s3_url = s3_util.generate_s3_url(bucket_name, region)
    print(f"S3 URL for bucket '{bucket_name}' in region '{region}': {s3_url}")
```

输出：

```
class S3Utils:
    '''
    Class to generate S3 URL from bucket name and region
    '''
    def __init__(self):
        self.base_url = "https://s3"

    def generate_s3_url(self, bucket_name, region):
        return f"{self.base_url}.{region}.amazonaws.com/{bucket_name}"

# Example usage:
if __name__ == "__main__":
    s3_util = S3Utils()
    bucket_name = "my-example-bucket"
    region = "us-west-1"
    s3_url = s3_util.generate_s3_url(bucket_name, region)
    print(f"S3 URL for bucket '{bucket_name}' in region '{region}': {s3_url}")
```

生成算法

Amazon Q Developer 可以生成常用的算法，尤其是与之相关的算法 AWS SDKs。在你的脚本中，添加一个提示来指定你想要的函数。

此示例说明如何生成用于命名值的用户 ID 键的字典，然后使用冒泡排序算法生成排序函数。

输入:

```
class SortList:
    @staticmethod
    def get_user_id_name_map():
        user_id_name_map = {
            1: "Amit",
            2: "Rahul",
            3: "Vijay",
            4: "Raj",
            5: "Kumar"
        }
        return user_id_name_map


    '''
    Sort the list of user id's alphabetically using bubble sort
    '''
```

输出：

```
    @staticmethod
    def sort_user_id_name_map():
        user_id_name_map = SortList.get_user_id_name_map()
        list_of_user_ids = list(user_id_name_map.keys())
        list_of_user_names = list(user_id_name_map.values())
        print("Before sort: ", list_of_user_ids)
        for i in range(len(list_of_user_ids)):
            for j in range(len(list_of_user_ids) - i - 1):
                if list_of_user_names[j] > list_of_user_names[j + 1]:
                    list_of_user_ids[j], list_of_user_ids[j + 1] = list_of_user_ids[j +
 1], list_of_user_ids[j]
                    list_of_user_names[j], list_of_user_names[j + 1] =
 list_of_user_names[j + 1], list_of_user_names[j]
        print("After sort: ", list_of_user_ids)
```

此外，您还可以为标准程序创建提示。在此示例中，用户创建了将字典存储到 Amazon DynamoDB 表的提示。在脚本的开头，用户添加了添加 DynamoDB 库的提示，以帮助为 Amazon Q 开发者提供更多背景信息。

输入:

```
'''
Add libraries for Amazon Dynamo DB
'''
```

输出：

```
'''
Add libraries for Amazon Dynamo DB and JSON.

iMport necessary libraries for working with Amazon DynamoDB and JSON data.
'''

import boto3
import json
from botocore.exceptions import ClientError

def load_data_to_dynamodb(table_name, data_file):
    # Create a DynamoDB resource
    dynamodb = boto3.resource('dynamodb')

    # Get a reference to the table
    table = dynamodb.Table(table_name)

    # Load data from the JSON file
    with open(data_file, 'r') as file:
        data = json.load(file)

    # Iterate over the data and load it to the table
    for item in data:
        try:
            table.put_item(Item=item)
            print(f"Successfully loaded data: {item}")
        except ClientError as e:
            print(f"Error loading data: {e.response['Error']['Message']}")

# Example usage
load_data_to_dynamodb('myTable', 'data.json')
```

现在，用户可以轻松地让 Amazon Q 帮助将字典存储在 DynamoDB 中。

输入:

```
'''
```

```
   function to store the above sorted dict to dynamodb table named users
'''
```

输出：

```
@staticmethod
    def store_user_id_name_map_in_dynamodb():
        user_id_name_map = SortList.get_user_id_name_map()
        list_of_user_ids = list(user_id_name_map.keys())
        list_of_user_names = list(user_id_name_map.values())
        dynamodb = boto3.resource('dynamodb')
        table = dynamodb.Table('users')
        for i in range(len(list_of_user_ids)):
            try:
                response = table.put_item(
                    Item={
                        'user_id': list_of_user_ids[i],
                        'user_name': list_of_user_names[i]
                    }
                )
            except ClientError as e:
                print(e.response['Error']['Message'])
```

生成单元测试

通过内嵌评论，Amazon Q Developer 可以生成单元测试来执行断言和其他测试。在此示例中，用户要求 Amazon Q 生成适用于单元测试的库，以设置测试脚本的上下文。然后，用户要求 Amazon Q 为该函数编写一个测试`sort_user_id_name_map`函数。

输入:

```
'''
Import all the required unit test libraries for python and then generate a unit test
 for sort_user_id_name_map function.
'''
```

输出：

```
import unittest
from unittest.mock import patch
from sort_user_id_name_map import sort_user_id_name_map
```

```
class TestSortUserIdNameMap(unittest.TestCase):
    @patch('sort_user_id_name_map.sort_user_id_name_map')
    def test_sort_user_id_name_map(self, mock_sort):
        mock_sort.return_value = {'id1': 'name1', 'id2': 'name2'}
        result = sort_user_id_name_map({'id2': 'name2', 'id1': 'name1'})
        self.assertEqual(result, {'id1': 'name1', 'id2': 'name2'})

if __name__ == '__main__':
    unittest.main()
```

此外，在用户与 Amazon Q Developer 之间的聊天中，它会根据用户的代码输入提示建议并生成单元测试。有关更多信息，请参阅聊天示例。

Java 示例

以下 Java 示例演示了用户如何开始生成内嵌代码建议。这些说明性示例向您展示了如何为自己的项目触发编码助手。以下大多数示例适用于其他语言：

- 生成类和函数
- 文件代码
- 生成算法
- 生成单元测试

生成类和函数

要生成全部或部分类实现，请使用代码注释。该评论描述了该班的意图。从简单开始，必要时添加更多细节。生成类后，可以继续在其框架内生成函数。

在撰写评论时，最好使用该语言的注释结构。In Java，这是/** */。

输入:

```
/**
 * DocumentReader class. has a name, children and optional parent.
 * Includes I/O functions to read and
 * write document
 */
```

输出：

```
/**
 * DocumentReader class. has a name, children and optional parent.
 * Includes I/O functions to read and
 * write document
 */
import java.util.ArrayList;
public class DocumentReader {
    private String name;
    private DocumentReader parent;
    private ArrayList<DocumentReader> children;

    /**
     * Constructor
     * @param name name of the document
     * @param parent parent of the document
     */
    public DocumentReader(String name, DocumentReader parent) {
        this.name = name;
        this.parent = parent;
        this.children = new ArrayList<>();
    }

    /**
     * Constructor
     * @param name name of the document
     */
    public DocumentReader(String name) {
        this(name, null);
    }
    /**
     * Getter for name
     * @return name of the document
     */
    public String getName() {
        return name;
    }

    /**
     * Getter for parent
     * @return parent of the document
     */
    public DocumentReader getParent() {
```

```
        return parent;
    }

    /**
     * Getter for children
     * @return children of the document
     */
    public ArrayList<DocumentReader> getChildren() {
        return children;
    }

    /**
     * Adds a child to the document
     * @param child child to be added
     */
    public void addChild(DocumentReader child) {
        children.add(child);
    }

    /**
     * Removes a child from the document
     * @param child child to be removed
     */
    public void removeChild(DocumentReader child) {
        children.remove(child);
    }
}
```

现在有了初始类，可以生成一个带有额外注释的附加函数。

输入:

```
/**
* function to read document from local directory path
* and print first line of document
*/
```

输出：

```
/**
 * function to read document from local directory path
 * and print first line of document
 */
```

```
public void readDocument() {
    try {
        BufferedReader reader = new BufferedReader(new FileReader(name));
        String line = reader.readLine();
        System.out.println(line);
        reader.close();
    } catch (IOException e) {
        System.out.println("Error reading document: " + e.getMessage());
    }
}
```

文件代码

依靠内联文档，缩短编写代码的时间。添加一个 Java 评论中断/* */。

输入:

```
import java.util.function.Function;

public class S3Utils {
    /*
     * <place cursor here>
    */
    public static Function<String, String> generateS3Function(String bucketName, String
 region) {
        return (String key) -> {
            String url = generateS3Url(bucketName, key, region);
            return url;
        };
    }
}
```

如果未进行内联生成，请在将光标放在所需文档的位置后执行 Amazon Q Developer 内联建议。

输出：

```
import java.util.function.Function;

public class S3Utils {
    /*
     * Generate a function that returns the S3 URL for a given key
     * @param bucketName the name of the bucket
```

```
     * @param region the region of the bucket
     * @return the function that generates the S3 URL
    */
    public static Function<String, String> generateS3Function(String bucketName, String
 region) {
        return (String key) -> {
            String url = generateS3Url(bucketName, key, region);
            return url;
        };
    }
}
```

要更简单地解释代码（类、函数或脚本元素），请使用正向字符 (//)，这些字符策略性地放在要添加文档的代码部分之前。

输入:

```
//
public class S3Utils {
    /*
     * Generate a function that returns the S3 URL for a given key
     * @param bucketName the name of the bucket
     * @param region the region of the bucket
     * @return the function that generates the S3 URL
    */
    public static Function<String, String> generateS3Function(String bucketName, String
 region) {
        return (String key) -> {
            String url = generateS3Url(bucketName, key, region);
            return url;
        };
    }
}
```

输出：

```
// This class is used to generate a function that returns the S3 URL for a given key
public class S3Utils {
    /*
     * Generate a function that returns the S3 URL for a given key
     * @param bucketName the name of the bucket
     * @param region the region of the bucket
     * @return the function that generates the S3 URL
```

```
    */
    public static Function<String, String> generateS3Function(String bucketName, String
 region) {
        return (String key) -> {
            String url = generateS3Url(bucketName, key, region);
            return url;
        };
    }
}
```

生成算法

Amazon Q 开发者可以生成常用的算法，尤其是与之相关的算法 AWS SDKs。在你的脚本中，添加一个提示来指定你想要的函数。

此示例生成用户名密钥到命名值的哈希映射，然后使用冒泡排序算法生成排序函数。

输入:

```
import java.util.HashMap;
public class SortList {

    // generate key value pair hash map for
    // user id and name
    public static HashMap<Integer, String> getUserIdNameMap() {
        HashMap<Integer, String> userIdNameMap = new HashMap<Integer, String>();
        userIdNameMap.put(1, "Amit");
        userIdNameMap.put(2, "Rahul");
        userIdNameMap.put(3, "Vijay");
        userIdNameMap.put(4, "Raj");
        userIdNameMap.put(5, "Kumar");
        return userIdNameMap;
    }

    /*
    * Create function to sort UserIdName map by name alphabetically using bubble sort
    */


}
```

输出：

```
/*
* Create function to sort UserIdName map by name alphabetically using bubble sort
*/
public static HashMap<Integer, String> sortUserIdNameMap(HashMap<Integer, String>
 userIdNameMap) {
    int size = userIdNameMap.size();
    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size; j++) {
            if (userIdNameMap.get(i).compareTo(userIdNameMap.get(j)) > 0) {
                String temp = userIdNameMap.get(i);
                userIdNameMap.put(i, userIdNameMap.get(j));
                userIdNameMap.put(j, temp);
            }
        }
    }
    return userIdNameMap;
}
```

此外，用户还可以为标准程序创建提示。在此示例中，用户创建了将哈希映射存储到 Amazon DynamoDB 表的提示。在脚本的开头，用户添加了添加 DynamoDB 的提示 Java 库可帮助为 Amazon Q 开发者提供更多背景信息。

输入:

```
// add libraries for Amazon DynamoDB
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;

     /*
     * Store sorted UserIdName map into Amazon DynamoDB table named users
     */
```

输出：

```
// add libraries for Amazon DynamoDB
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;

     /*
     * Store sorted UserIdName map into Amazon DynamoDB table named users
     */
```

```
    public static void storeSortedUserIdNameMap(HashMap<Integer, String> userIdNameMap)
 {
        // create Amazon DynamoDB client
        AmazonDynamoDB client = AmazonDynamoDBClientBuilder.defaultClient();

        // store sorted UserIdName map into Amazon DynamoDB table named users
        for (int userId : userIdNameMap.keySet()) {
            String name = userIdNameMap.get(userId);
            client.putItem("users", new HashMap<String, Object>() {{
                put("userId", userId);
                put("name", name);
            }});
        }
    }
```

生成单元测试

通过内嵌评论，Amazon Q Developer 可以生成单元测试来执行断言和其他 JUnit 测试。在此示例中，用户首先要求 Amazon Q 生成适用的库 JUnit 以设置测试脚本的上下文。接下来，用户创建一个公共类，后面是声明符和注释字符串。

输入:

```
// import libraries for java unit tests
import org.junit.Test;
import static org.junit.Assert.*;

public class UnitTests {

    @Test
    /*
     * Create unit test for getUserIdNameMap
     */
}
```

输出：

```
@Test
/*
* Create unit test for getUserIdNameMap
*/
public void testGetUserIdNameMap() {
```

```
    // create a new instance of the UserIdNameMap class
    UserIdNameMap userIdNameMap = new UserIdNameMap();

    // create a new instance of the User class
    User user = new User("John", "Doe");

    // add the user to the map
    userIdNameMap.addUser(user);

    // get the user from the map
    User retrievedUser = userIdNameMap.getUser("John");

    // assert that the retrieved user is the same as the user added to the map
    assertEquals(user, retrievedUser);
}
```

此外，在用户与 Amazon Q Developer 之间的聊天中，它会根据用户的代码输入提示建议并生成单元测试。有关更多信息，请参阅聊天示例。

聊天示例

Amazon Q Developer 聊天可帮助开发者完成各种任务，从编码、测试和升级应用程序，到诊断错误、执行安全扫描和修复以及优化 AWS 资源。本节介绍与 Amazon Q 聊天如何支持开发者完成典型的编程任务，例如：

- 询问 AWS 服务
- 生成代码
- 生成单元测试
- 解释代码

询问 AWS 服务

用户可以向 Amazon Q Developer 提问 AWS 服务，例如寻找合适的服务或了解最佳实践。

聊一聊 AWS 服务

用户：

如果我需要保持较低的成本，哪种容器服务最适合用来运行我的工作负载？

亚马逊 Q：

根据提供的信息，您似乎正在寻找一种经济实惠的容器服务来运行您的工作负载。在这种情况下，我建议考虑将亚马逊弹性容器服务 (Amazon ECS) 与 Fargate 配合使用。[1]

带有 Fargate 的 Amazon ECS 是一款适用于容器的无服务器计算引擎，它允许您运行容器，而无需管理任何底层基础设施。有了 Fargate，你无需配置或管理服务器，Fargate 会为你处理这个问题。这有助于保持较低的成本，因为您只需为容器实际使用的计算资源付费，而不必为闲置的服务器容量付费。[2]

将 Amazon ECS 与 Fargate 配合使用的一些主要好处：

• 无需管理服务器或集群 — Fargate 为您处理基础设施的配置和管理。

• Pay-as-you-go 定价 — 您只需为容器使用的计算资源付费，无需为闲置容量付费。

• 自动扩展 — Fargate 可以根据需求自动向上和向下扩展容器。

• 与其他功能（ AWS 服务 例如负载平衡、日志记录和监控）紧密集成。

提供的资源还提到，ECS 服务在容器调度和放置方面提供了灵活性，这有助于进一步优化成本。

我建议您在 AWS 网站上查看带有 Fargate 的 Amazon ECS 的定价详情，以便根据您的具体工作负载要求更好地了解成本。当然，如果您还有其他问题，请随时告诉我！

源

[1] 哪种 AWS 容器服务最适合您？

aptgetops.tech >--你 aws-container-services-which one-is-right-for

[2] Java 非 HTTP 工作负载的容器选项

stackoverflow.com > 问题 > 43417016

结束关于 Amazon Q 的聊天 AWS 服务

生成代码

用户可以向 Amazon Q 开发者寻求帮助以编写代码片段。

聊聊编写代码

用户：

能否举个例子，说明如何使用列出所有 S3 存储桶？ AWS SDK for Python (Boto)

亚马逊 Q：

当然，以下是如何使用 AWS SDK for Python (Boto)[1] 列出所有 S3 存储桶的示例：

```
import boto3
# Create an S3 client
s3 = boto3.client('s3')

# List all the buckets
response = s3.list_buckets()

# Print the name of each bucket
for bucket in response['Buckets']:
    print(bucket['Name'])
```

代码的工作原理如下：

1. 我们首先导入 boto3 库，这是适用于 Python 的软件开发工具包。
2. 我们使用创建 S3 客户端`boto3.client('s3')`。
3. 我们在 S3 客户端上调用该`list_buckets()`方法，它会返回一个包含账户中所有存储桶列表的响应。
4. 然后，我们遍历响应中的`Buckets`列表并打印出每个存储桶的名称。

这是一个基本示例，但为使用 Amazon S3 AWS SDK for Python (Boto) 提供了许多更高级的功能，例如创建、删除和管理存储桶中的对象，设置存储桶策略等。您可以在 AWS SDK for Python (Boto)) 文档中找到更多示例和文档。[2]

请记住，在使用时 AWS 服务，请务必遵循处理凭证和敏感信息的最佳实践，例如将 AWS 访问密钥存储在环境变量中，而不是在代码中对其进行硬编码。

来源：[1] 使用适用于 Python 的软件开发工具包的亚马逊 S3 示例 (Boto3)-SDK 代码示例 docs.aws.amazon.com > 代码 AWS 库 > latest-1x >... > python-s3.html

[2] 开始使用 SDK 使用亚马逊 S3 存储桶和对象——S AWS D AWS K 代码示例 docs.aws.amazon.com > 代码库 > latest-1x >... > s3_example_s 3_buckets_ _sect... GettingStarted

结束关于编写代码的 Amazon Q 聊天

生成单元测试

用户可以将选定的代码作为提示发送给 Amazon Q Developer，在那里会应用其他说明，例如生成单元测试。有关 step-by-step说明，请参阅 Amazon Q 开发者用户指南中的向 Amazon Q 发送代码。

聊聊单元测试

用户：

为以下函数编写一个包含模拟数据的单元测试：

```
// poll messages from Amazon SQS queue name
   public List<String> pollMessages(String queueName) {
       List<String> messages = new ArrayList<>();
       // poll messages from queue
       ReceiveMessageResult = sqsClient.receiveMessage(new ReceiveMessageRequest().
       for (Message : receiveMessageResult.getMessages()) {
           messages.add(message.getBody());
       }
       return messages;
       }
```

亚马逊 Q：

要为 PollMessages 函数编写带有模拟数据的单元测试，我们可以使用像 Mockito 这样的模拟框架。示例如下：

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
import com.amazonaws.services.sqs.model.ReceiveMessageResult;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Arrays;
import java.util.List;

import static org.junit.jupiter.api.Assertions.assertEquals;
```

```
@ExtendWith(MockitoExtension.class)
public class SQSUtilsTest {

    @Mock
    private AmazonSQS sqsClient;

    @InjectMocks
    private SQSUtils sqsUtils;

    @Test
    public void testPollMessages() {
        // Arrange
        String queueName = "test-queue";
        Message message1 = new Message().withBody("message1");
        Message message2 = new Message().withBody("message2");
        ReceiveMessageResult receiveMessageResult = new
 ReceiveMessageResult().withMessages(message1, message2);
        Mockito.when(sqsClient.receiveMessage(new
 ReceiveMessageRequest().withQueueUrl(queueName)))
                .thenReturn(receiveMessageResult);

        // Act
        List<String> messages = sqsUtils.pollMessages(queueName);

        // Assert
        assertEquals(Arrays.asList("message1", "message2"), messages);
    }
}
```

在此示例中，我们使用 Mockito 来模拟 AmazonSQS 客户端，并设置 ReceiveMessage 方法的预期行为，以便它返回两条正文为 “message1” 和 “message2” 的模拟消息。然后，我们调用 pollMessages 方法并断言返回的消息列表与预期值相匹配。

关键点是：

1. 我们使用 @ ExtendWith (MockitoExtension.class) 注解来启用 Mockito 与的集成。 JUnit
2. 我们使用 @Mock 注解来创建 AmazonSQS 客户端的模拟实例。
3. 我们使用 @ InjectMocks 注解将模拟 AmazonSQS 客户端注入到课堂中。 SQSUtils
4. 在测试方法中，我们使用 mockito.when () .thenReturn () 设置了 receiveMessage 方法的预期行为。
5. 我们调用 pollMessages 方法并断言返回的消息列表与预期值相匹配。

这种方法使我们能够在不实际与 Amazon SQS 服务交互的情况下测试 PollMessages 方法，这可以使测试更快、更可靠。

结束关于单元测试的 Amazon Q 聊天

解释代码

对于代码审查流程，开发人员可以向 Amazon Q Developer 发送代码选择以更好地理解代码。此外，Amazon Q 提供的解释也可以用于文档目的。在以下示例中，Amazon Q 使用函数解释了处理 AWS Lambda 函数，然后提出了文档字符串。

聊一聊解释代码

用户：

解释一下我的代码的以下部分：

```
def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    s3_client = boto3.client('s3')
    table = dynamodb.Table('DYNAMODB_TABLE')
    # Get s3 object from the event
    print(event)
    print(context)
    # get object from event
    bucket_key = event['Records'][0]['s3']['object']['key']
    object_key = unquote_plus(bucket_key)
    fullText = detect_text(object_key, bucket)

    response = s3_client.head_object(Bucket=bucket, Key=object_key)
    print(response)
    print(fullText)
    metadata = response['ResponseMetadata']
    metadata = json.dumps(metadata)

    now = datetime.now()
    current_time = now.strftime("%A %d %B, %Y %H:%M:%S")
    print("Current Time =", current_time + "\n\n")
    reason, outcome = detect_restaurant_closure(fullText, current_time)
    table.put_item(
        Item={
            'id': object_key,
            'creationTime': now.strftime("%Y-%m-%d %H:%M:%S"),
```

```
            'reason': reason,
            'text': fullText,
            'metadata': metadata
        }
    )
```

亚马逊 Q：

提供的代码块似乎是一个写入的 AWS Lambda 函数 Python。 让我解释一下这段代码的关键方面：

1. 该函数名为 lambda_handler，它是 Lambda 函数的入口点。
2. 它使用 boto3 库与 AWS DynamoDB 和 S3 服务进行交互。
3. 该函数从传入的事件数据中检索 S3 对象密钥和存储桶名称。
4. 然后，它会调用 detect_text 函数（未显示在提供的代码中）来提取 S3 对象的全文内容。
5. 该函数还使用 head_object 方法检索有关 S3 对象的元数据。
6. 然后，它提取当前时间并调用 detect_restaurant_closure 函数（也未显示），以根据提取的文本确定餐厅关闭的原因和结果。
7. 最后，该函数将以下信息存储在 DynamoDB 表中：
   - 将 S3 对象密钥作为 id
   - 创建时间
   - 餐厅关闭的原因和结果
   - 提取的 S3 对象文本

总体而言，这个 Lambda 函数似乎是一个更大的系统的一部分，该系统处理 S3 对象，提取文本内容，分析其中的餐厅关闭信息，并将结果存储在 DynamoDB 表中。要完全理解此代码的功能，需要获取 detect_text 和 detect_restaurant_closure 函数的具体细节。

亚马逊 Q 回复已结束

用户：将此解释汇总成此 lambda_handler 函数的文档字符串

Amazon Q：以下是 lambda_handler 函数 [1] 的建议文档字符串：

```
def lambda_handler(event, context):
    """
    AWS Lambda function that processes an S3 object, extracts text content, analyzes
 it, and stores the results.
    Args:
```

```
        event (dict): The event data that triggered the Lambda function, containing
information about the S3 object.
        context (object): The context object provided by AWS Lambda, containing
information about the runtime environment.

    Returns:
        None
    """

    # function implementation details...
```

这个 docstring 概述了函数的用途、它期望的输入参数以及预期的返回值（在本例中为缺少返回值）。它使用户可以清楚地了解 lambda_handler 函数的作用，而无需深入研究实现细节。

来源：[1] 在 C# 中定义 Lambda 函数处理程序- AWS Lambda https://docs.aws.amazon.com/lambda/latest/dg/csharp-handler.html-: ~: text=Lambda 函数处理程序是、、退出或超时。

关于解释代码的 Amazon Q 聊天已结束

对 Amazon Q 开发者中的代码生成场景进行故障排除

在使用 Amazon Q Developer 时，您可能会遇到以下常见情况，即代码生成和分辨率不准确：

- 空代码生成
- 持续评论
- 行内代码生成不正确
- 聊天结果不足

空代码生成

在开发代码时，您可能会注意到以下问题：

- Amazon Q 不提供建议。
- 您的 IDE 中会出现 “没有来自 Amazon Q 的建议” 消息。

你首先想到的可能是 Amazon Q 无法正常运行。但是，这些问题的根本原因通常与脚本中的上下文或 IDE 中的打开项目有关。

如果 Amazon Q 开发者没有自动提供建议，您可以使用以下快捷方式手动执行 Amazon Q 内嵌建议：

- 电脑-Alt+ C
- macOS-Option+C

有关更多信息，请参阅 Amazon Q 开发者用户指南中的使用快捷键。

在大多数情况下，Amazon Q 会生成建议。当 Amazon Q 返回 “没有来自 Amazon Q 的建议” 消息时，请查看该问题的以下常见解决方案：

- 上下文不足-确保存在适用的库。确保脚本中定义的类和函数与新代码相关。
- 提示模棱两可 — 如果提示令人困惑或不清楚，Amazon Q 在提出内联代码建议时表现不佳。在这些场景中，请在代码和注释中更具体，以获得正确的响应。

如果你陷入困境，请尽你所能继续编程。通常，Amazon Q 会识别出您的意图，从而触发它提供建议。另一种选择是在Amazon Q聊天中发送带有提示说明的代码片段。有关更多信息，请参阅 Amazon Q 开发者用户指南中的向 Amazon Q 发送代码。

持续评论

您可能会使用评论来写提示，但只会收到来自 Amazon Q 的笼统或重复的评论行。

输入:

```
// Kafka test
```

结果：

```
// Kafka test
// Kafka test
// Kafka test
// Kafka test
```

通常，此问题是由于脚本中缺乏上下文或提示说明不佳所致。根据所用语言，我们建议您使用标准的评论格式。在此示例中，用户更改了/* */ 用于评论栏的提示，这一更改使得 Amazon Q 的性能有所提高。

输入:

```
/*
 * Create a unit test to push a message to a Kafka topic
 */
```

结果：

在此结果中，用户按顺序接受了几条建议。

```
/*
 * Create a unit test to push a message to a Kafka topic
 */
import org.junit.Test;
import java.util.Properties;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import static org.junit.Assert.assertEquals;
// create function for a Kafka test
public class KafkaTest {
    @Test
```

```
    public void testKafka() {
        // create Kafka producer
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.put("key.serializer",
 "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer",
 "org.apache.kafka.common.serialization.StringSerializer");
        KafkaProducer<String, String> producer = new KafkaProducer<>(props);
        // create Kafka topic
        String topic = "test";
        // send message to Kafka topic
        producer.send(new ProducerRecord<>(topic, "key", "value"));
        // close Kafka producer
        producer.close();
        // assert that message was sent successfully
        assertEquals("value", KafkaTest.consumer.poll().value());
    }
}
```

行内代码生成不正确

有时，Amazon Q Developer 的内联建议没有用处或不符合提示的目标。有时，行内代码建议可能会提供类似于简单的 Hello World 方法的通用函数。

如果行内代码建议无济于事，请尝试以下最佳实践：

- 查看脚本的上下文。通常，您提供的背景信息越多，您从 Amazon Q 获得的结果就越好。
- 如果上下文变化和/或噪音很大，请对代码进行模块化。将相似的函数分组到同一个脚本中。按目标区分不同的类和函数。
- 从战术上写下您的类和函数名称，以帮助 Amazon Q 提供更好的推荐。

在此示例中，开发人员正在从头开始编写一个脚本，用于将文件上传到 Hadoop。

输入:

```
class  MyFunction{

    // create function to read data locally and upload to HDFS filesystem
```

结果：

```
class  MyFunction{

    // create function to read local data file path and upload to HDFS filesystem
    public void readAndUploadToHDFS(String inputPath, String outputPath) throws
 IOException {
        // read data from local file
        BufferedReader in = new BufferedReader(new FileReader(inputPath));

}
```

尽管 Amazon Q 提供了建议，但它没有达到开发者的预期。为了提高性能，开发人员导入库并更改类名以表示 Hadoop 实用程序。

修复：

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.FileReader;

// import applicable hadoop libraries for IO operations
import org.apache.hadoop.fs.Path;

class  HadoopUtilities{

    // create function to read local data file path and upload to HDFS filesystem
}
```

结果：

在此结果中，开发者按顺序接受了几项建议。

```
class  HadoopUtilities{

    // create function to read local data file path and upload to HDFS filesystem
    public static void uploadToHDFS(String localFilePath, String hdfsFilePath) throws
 IOException {
        // create HDFS file path object
        Path hdfsPath = new Path(hdfsFilePath);
        // create local file path object
        Path localPath = new Path(localFilePath);
```

```
        // create hadoop file system object
        org.apache.hadoop.fs.FileSystem hdfs = org.apache.hadoop.fs.FileSystem.get(new
 org.apache.hadoop.conf.Configuration());
        // upload local file to HDFS
        hdfs.copyFromLocalFile(localPath, hdfsPath);
        // close HDFS file system
        hdfs.close();
        System.out.println("File uploaded successfully!");
}
```

在同一个地方 HadoopUtilities 脚本中，开发者添加评论提示以阅读亚马逊简单队列服务 (Amazon SQS) 主题。

输入:

```
// read message from SQS
```

结果：

```
// read message from SQS
    public static void readMessageFromSQS(String message) {
        System.out.println("Received message from SQS: " + message);
```

Amazon Q 建议打印字符串方法，而不是使用 Java 方法从亚马逊 SQS 读取 SQS 消息。但是，这个建议并不能满足开发者的初始目标。

在这种情况下，最好对代码进行重新模块化。该脚本的重点是 Hadoop 方法。因此，创建专门用于 SQS 操作的实用程序脚本可能是更好的方法。

为 SQS 实用程序创建单独的类可以改进内联建议，该建议可用于 HadoopUtilities 脚本。

输入:

```
// import function for SQS
import com.amazonaws.services.sqs.AmazonSQS;
import java.util.ArrayList;
import java.util.List;


class SQSUtils {
```

```
    // instantiate SQS client
    private AmazonSQS sqsClient;

    // load SQS object
    public SQSUtils(AmazonSQS sqsClient) {
        this.sqsClient = sqsClient;
    }

    // poll message from Amazon SQS given queue name
}
```

结果：

```
// poll message from Amazon SQS given queue name
    public List<String> pollMessages(String queueName) {
        List<String> messages = new ArrayList<>();
        // poll messages from queue
        receiveMessageResult = sqsClient.receiveMessage(new
 ReceiveMessageRequest(queueName));
        for (Message message : receiveMessageResult.getMessages()) {
            messages.add(message.getBody());
        }
        return messages;
```

如果内联建议仍未提供预期结果，请在聊天会话中让 Amazon Q Developer 生成代码。

聊一聊如何生成代码

用户：

生成 Java 函数以读取来自亚马逊 SQS 队列的消息

亚马逊 Q：

以下是用于从 Amazon SQS 队列中读取消息的 Java 函数示例：

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
import com.amazonaws.services.sqs.model.ReceiveMessageResult;
public class SQSMessageReader {
```

```
    private AmazonSQS sqsClient;
    private String queueUrl;

    public SQSMessageReader(AmazonSQS sqsClient, String queueUrl) {
        this.sqsClient = sqsClient;
        this.queueUrl = queueUrl;
    }

    public String readMessage() {
        ReceiveMessageRequest receiveMessageRequest = new
ReceiveMessageRequest(queueUrl)
                .withMaxNumberOfMessages(1)
                .withWaitTimeSeconds(10);

        ReceiveMessageResult receiveMessageResult =
sqsClient.receiveMessage(receiveMessageRequest);

        if (receiveMessageResult.getMessages().size() > 0) {
            Message message = receiveMessageResult.getMessages().get(0);
            String messageBody = message.getBody();
            String receiptHandle = message.getReceiptHandle();

            // Process the message
            System.out.println("Received message: " + messageBody);

            // Delete the message from the queue
            sqsClient.deleteMessage(queueUrl, receiptHandle);

            return messageBody;
        } else {
            return null;
        }
    }
```

此代码创建一个 SQSMessage Reader 类，该类将 AmazonSQS 客户端和队列网址作为构造函数参数。readMessage () 方法使用从队列中提取最多 1 条消息，最长等待时间为 10 秒。ReceiveMessageRequest

如果收到消息，则会打印消息正文并使用将消息从队列中删除 DeleteMessageRequest。然后返回消息正文。如果未收到任何消息，则返回 null。

结束关于生成代码的 Amazon Q 聊天

聊天结果不足

在开发代码时，开发人员可以让 Amazon Q 生成常用函数、提供建议或解释代码。有时，如果出现提示（例如问题或代码片段），Amazon Q 可能会提供不符合您期望的一般反馈或结果。在这些场景中，请尝试以下操作：

- 尝试不同的提示以改善您从 Amazon Q 获得的代码生成结果。
- 按 URLs 照 Amazon Q 在回答您的提示或问题时经常提供的内容进行操作。
- 提供反馈以帮助改善 Amazon Q. 的未来表现

例如，开发人员正在创建检索增强生成 (RAG) 解决方案，鉴于句子之间的语义关系，他们需要对文档进行分块。开发者按如下方式与 Amazon Q 进行了交谈。

聊一聊 RAG 解决方案

用户：

对于 RAG 解决方案，生成用于语义分块的代码

亚马逊 Q：

当然，这里有一个示例，说明如何为检索增强生成 (RAG) 解决方案实现语义分块 Python 使用该 LangChain 库：

```
from langchain.text_splitter import CharacterTextSplitter, TokenTextSplitter
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import Chroma

# Initialize the text splitter
text_splitter = TokenTextSplitter(
    chunk_size=500,
    chunk_overlap=50,
    length_function=len
)
# Initialize the embeddings model
embeddings = HuggingFaceEmbeddings()

# Create the vector store
vectorstore = Chroma(
    embedding_function=embeddings.embed_documents,
    persist_directory="./chroma-data"
```

```
)

# Chunk the text using semantic similarity
def semantic_chunk(text):
    chunks = text_splitter.split_text(text)
    embeddings = embeddings.embed_documents(chunks)
    vectorstore.add_texts(chunks, embeddings)
    return chunks
```

在这个例子中，我们使用的是 TokenTextSplitter fro LangChain m 将输入文本拆分为 500 个字符的块，重叠部分为 50 个字符。这些区域有：HuggingFaceEmbeddings model 用于为每个区块生成嵌入内容，而且 Chroma 向量存储用于存储块及其嵌入内容。

semantic_chunk 函数接收输入文本，使用 text_splitter 将其分成块，使用嵌入模型为每个区块生成嵌入内容，然后将这些块及其嵌入添加到 vectorstore.

这种方法允许您将输入文本拆分为语义上有意义的块，这对于RAG解决方案中的各种任务（例如问答或摘要）非常有用。

请注意，这是一个基本示例，您可能需要根据您的特定用例和数据调整参数（例如 chunk_size、chunk_overlap）或文本拆分器和嵌入模型的选择。

关于 RAG 解决方案的 Amazon Q 聊天已结束

但是，这种方法仅提供了一种使用滑动窗口技术对文档进行分块的基本方法。开发人员想要一种依赖于测量嵌入的高级方法。在这些情况下，开发者必须根据自己的目标在提示中添加更多细节。当开发者使用引用已知的 AI21 文本拆分器功能时 LangChain，他们会收到改进的响应，如以下聊天所示。

聊一聊 RAG 解决方案（带有改进的提示音）

用户：

上面的提示仅与拆分每个区块的令牌大小有关。我想使用 AI21 语义文本拆分器。使用语言链的 A121 语义文本拆分生成一个函数。

亚马逊 Q：

好吧，让我们使用 AI21 语义文本拆分器 LangChain 来实现语义分块。这里有一个例子：[1]

```
from langchain.text_splitter import AI21SemanticTextSplitter
from langchain.embeddings import HuggingFaceEmbeddings
```

```
from langchain.vectorstores import Chroma

# Initialize the text splitter
text_splitter = AI21SemanticTextSplitter()

# Initialize the embeddings model
embeddings = HuggingFaceEmbeddings()

# Create the vector store
vectorstore = Chroma(
    embedding_function=embeddings.embed_documents,
    persist_directory="./chroma-data"
)

# Chunk the text using semantic similarity
def semantic_chunk(text):
    chunks = text_splitter.split_text(text)
    embeddings = embeddings.embed_documents(chunks)
    vectorstore.add_texts(chunks, embeddings)
    return chunks
```

在此示例中，我们使用 from 将 AI21SemanticTextSplitter 输入文本拆分 LangChain 为语义上有意义的块。这些区域有：HuggingFaceEmbeddings model 用于为每个区块创建嵌入内容，而且 Chroma 向量存储用于存储块及其嵌入内容。

semantic_chunk 函数接收输入文本，使用 text_splitter 将其分成块，使用嵌入模型为每个区块创建嵌入内容，然后将这些块及其嵌入添加到 vectorstore.

这种方法允许您将输入文本拆分为语义上有意义的块，这对于RAG解决方案中的各种任务（例如问答或摘要）非常有用。

请注意，您可能需要根据您的特定用例和数据调整参数或嵌入模型的选择。

来源 [1] 了解如何在中使用文本拆分器 LangChain

soosweb3.hashnode.dev >-text-splitters the-ultimate-langchain-series

关于 RAG 解决方案的 Amazon Q 聊天已结束（提示已改进）

根据聊天结果，开发者可以使用该功能并导航到信息来源。

通常，当您与 Amazon Q Developer 聊天时，高级主题和问题需要在提示中提供更具体的背景信息。如果您认为聊天结果不准确，请使用竖起大拇指图标提供有关 Amazon Q 回复的反馈。Amazon Q

Developer 不断使用反馈来改进 future 的版本。对于产生积极结果的互动，使用竖起大拇指图标提供反馈很有用。

FAQs关于 Amazon Q 开发者

本节提供了有关使用 Amazon Q Develor 进行代码开发的常见问题解答。

什么是 Amazon Q Developer？

Amazon Q Developer 是一项强大的生成式人工智能驱动服务，旨在通过提供智能代码生成和建议来加快代码开发任务。2024 年 4 月 30 日，亚马逊 CodeWhisperer 成为 Amazon Q Developer 的一员。

我如何访问亚马逊 Q 开发者？

Amazon Q Developer 作为 Visual Studio 代码 AWS 工具包的一部分提供 JetBrains IDEs，比如 IntelliJ 和。 PyCharm要开始使用，请安装最新 AWS Toolkit 版本。

Amazon Q 开发者支持哪些编程语言？

生成 Visual Studio JetBrains IDEs，Amazon Q 开发者支持 Python, Java, JavaScript, TypeScript, C#,Go, Rust, PHP, Ruby、Kotlin、C、C++、Shell 脚本和 Scala 语言的脚本生成。SQL尽管本指南的重点是 Python 以及 Java 例如, 这些概念适用于任何支持的编程语言。

如何向 Amazon Q Developer 提供背景信息，以便更好地生成代码？

从现有代码开始，导入相关库，创建类和函数，或者建立代码框架。使用标准注释块进行自然语言提示。让你的脚本专注于特定的目标，并将不同的功能模块化成具有相关上下文的单独脚本。有关更多信息，请参阅 Amazon Q 开发人员的最佳编码实践。

如果使用 Amazon Q Developer 生成内嵌代码不准确，我该怎么办？

查看脚本的上下文，确保存在库，并确保类和函数与新代码相关。对代码进行模块化，并按目标区分不同的类和函数。写下清晰而具体的提示或评论。如果您仍然不确定代码的准确性并且无法继续操作，请开始与 Amazon Q 聊天，然后向其发送包含说明的代码片段。有关更多信息，请参阅 Amazon Q Developer 中的代码生成场景疑难解答。

如何使用 Amazon Q Developer 聊天功能生成代码和进行故障排除？

与 Amazon Q 聊天以生成常用函数、寻求推荐或解释代码。如果初始响应不令人满意，请尝试不同的提示并按照提供的提示进行操作URLs。此外，还要向 Amazon Q 提供反馈，以帮助改善其未来的聊天性能。使用竖起大拇指和大拇指向下图标提供反馈。有关更多信息，请参阅聊天示例。

使用 Amazon Q 开发者的一些最佳实践？

提供相关的背景信息，对提示进行实验和迭代，在接受代码建议之前对其进行审查，使用自定义功能，并了解数据隐私和内容使用政策。有关更多信息，请参阅 Amazon Q 开发人员生成代码的最佳实践和使用 Amaz on Q 开发人员推荐代码的最佳实践。

我能否自定义 Amazon Q Developer 以根据自己的代码生成推荐？

是的，请使用自定义，这是 Amazon Q Developer 的一项高级功能。通过自定义，企业可以提供自己的代码存储库，让 Amazon Q Developer 能够推荐内联代码建议。有关更多信息，请参阅 Amazon Q 开发人员的高级功能和资源。

使用 Amazon Q 开发者的后续步骤

利用从本综合指南中获得的知识，您可以在编码工作流程中有效地使用 Amazon Q Developer。AWS Toolkit 在你首选的 IDE（Visual Studio Code 或 JetBrains）中安装，然后开始探索 Amazon Q Developer 提供的基于人工智能的生成代码生成和推荐。

充分发挥 Amazon Q Developer 潜能的最有效方法是亲身体验自己的代码。在将 Amazon Q 集成到开发生命周期中时，请参阅本指南，了解最佳实践、疑难解答和真实示例。

此外，请查看参考资料中引用的 AWS 博客和开发者指南，随时了解最新情况。这些资源提供了最新更新、最佳实践和见解，可帮助您优化对 Amazon Q Developer 的使用。

您的反馈对于改进本指南并帮助它继续成为开发者的宝贵资源非常宝贵。分享您的经验、挑战以及对 future 版本的建议。您的意见将通过其他示例、故障排除场景和根据您的需求量身定制的见解来增强指南。

资源

AWS 博客

- Accelerate your Software Development Lifecycle with Amazon Q
- 使用 Amazon Q 开发者代理重新构想软件开发
- Amazon Q 的五个疑难解答示例
- 使用您的私有代码库在您的IDE中自定义 Amazon Q Developer
- 用于代码转换的 Amazon Q 开发者代理加速 Java 升级的三种方式
- 利用 Amazon Q 开发人员进行高效的代码调试和维护
- 使用 Amazon Q 开发者测试您的应用程序

AWS 文档

- Amazon Q 开发者用户指南
- Amazon Q 开发者代码自定义
- Amazon Q 开发者代码转换

AWS 工作坊

- Amazon Q 开发者沉浸日
- Amazon Q 开发者研讨会——构建 Q-Words 应用程序
- Amazon Q 开发者研讨会-创建有效的提示

贡献者

以下人员为本指南做出了贡献：

- 乔 King，高级数据科学家， AWS
- Prateek Gupta，团队负责人 — Sr. CAA， AWS
- Manohar Reddy Arranagu，建筑师， DevOps AWS
- Soumik Roy，云应用程序架构师， AWS
- Sanket Shinde，顾问， AWS

文档历史记录

下表介绍了本指南的一些重要更改。如果您希望收到有关未来更新的通知，可以订阅 RSSFeed 源。

变更	说明	日期
初次发布	—	2024 年 8 月 16 日

AWS 规范性指导词汇表

以下是 AWS 规范性指导提供的策略、指南和模式中的常用术语。若要推荐词条，请使用术语表末尾的提供反馈链接。

数字

7 R

将应用程序迁移到云中的 7 种常见迁移策略。这些策略以 Gartner 于 2011 年确定的 5 R 为基础，包括以下内容：

- 重构/重新架构 - 充分利用云原生功能来提高敏捷性、性能和可扩展性，以迁移应用程序并修改其架构。这通常涉及到移植操作系统和数据库。示例：将您的本地 Oracle 数据库迁移到兼容 Amazon Aurora PostgreSQL 的版本。
- 更换平台 - 将应用程序迁移到云中，并进行一定程度的优化，以利用云功能。示例：在中将您的本地 Oracle 数据库迁移到适用于 Oracle 的亚马逊关系数据库服务 (Amazon RDS) AWS Cloud。
- 重新购买 - 转换到其他产品，通常是从传统许可转向 SaaS 模式。示例：将您的客户关系管理 (CRM) 系统迁移到 Salesforce.com。
- 更换主机（直接迁移）- 将应用程序迁移到云中，无需进行任何更改即可利用云功能。示例：在中的 EC2 实例上将您的本地 Oracle 数据库迁移到 Oracle AWS Cloud。
- 重新定位（虚拟机监控器级直接迁移）：将基础设施迁移到云中，无需购买新硬件、重写应用程序或修改现有操作。您可以将服务器从本地平台迁移到同一平台的云服务。示例：将Microsoft Hyper-V应用程序迁移到 AWS。
- 保留（重访）- 将应用程序保留在源环境中。其中可能包括需要进行重大重构的应用程序，并且您希望将工作推迟到以后，以及您希望保留的遗留应用程序，因为迁移它们没有商业上的理由。
- 停用 - 停用或删除源环境中不再需要的应用程序。

A

ABAC

请参阅基于属性的访问控制。

抽象服务

参见托管服务。

ACID

参见原子性、一致性、隔离性、耐久性。

主动-主动迁移

一种数据库迁移方法，在这种方法中，源数据库和目标数据库保持同步（通过使用双向复制工具或双写操作），两个数据库都在迁移期间处理来自连接应用程序的事务。这种方法支持小批量、可控的迁移，而不需要一次性割接。与主动-被动迁移相比，它更灵活，但需要更多的工作。

主动-被动迁移

一种数据库迁移方法，在这种方法中，源数据库和目标数据库保持同步，但在将数据复制到目标数据库时，只有源数据库处理来自连接应用程序的事务。目标数据库在迁移期间不接受任何事务。

聚合函数

一个 SQL 函数，它对一组行进行操作并计算该组的单个返回值。聚合函数的示例包括SUM和MAX。

AI

参见人工智能。

AIOps

参见人工智能操作。

匿名化

永久删除数据集中个人信息的过程。匿名化可以帮助保护个人隐私。匿名化数据不再被视为个人数据。

反模式

一种用于解决反复出现的问题的常用解决方案，而在这类问题中，此解决方案适得其反、无效或不如替代方案有效。

应用程序控制

一种安全方法，仅允许使用经批准的应用程序，以帮助保护系统免受恶意软件的侵害。

应用程序组合

有关组织使用的每个应用程序的详细信息的集合，包括构建和维护该应用程序的成本及其业务价值。这些信息是产品组合发现和分析过程的关键，有助于识别需要进行迁移、现代化和优化的应用程序并确定其优先级。

人工智能（AI）

计算机科学领域致力于使用计算技术执行通常与人类相关的认知功能，例如学习、解决问题和识别模式。有关更多信息，请参阅什么是人工智能？

人工智能操作 (AIOps)

使用机器学习技术解决运营问题、减少运营事故和人为干预以及提高服务质量的过程。有关如何在 AIOps AWS 迁移策略中使用的更多信息，请参阅操作集成指南。

非对称加密

一种加密算法，使用一对密钥，一个公钥用于加密，一个私钥用于解密。您可以共享公钥，因为它不用于解密，但对私钥的访问应受到严格限制。

原子性、一致性、隔离性、持久性（ACID）

一组软件属性，即使在出现错误、电源故障或其他问题的情况下，也能保证数据库的数据有效性和操作可靠性。

基于属性的访问权限控制（ABAC）

根据用户属性（如部门、工作角色和团队名称）创建精细访问权限的做法。有关更多信息，请参阅 AWS Identity and Access Management (I AM) 文档 AWS中的 AB AC。

权威数据源

存储主要数据版本的位置，被认为是最可靠的信息源。您可以将数据从权威数据源复制到其他位置，以便处理或修改数据，例如对数据进行匿名化、编辑或假名化。

可用区

中的一个不同位置 AWS 区域 ，不受其他可用区域故障的影响，并向同一区域中的其他可用区提供低成本、低延迟的网络连接。

AWS 云采用框架 (AWS CAF)

该框架包含指导方针和最佳实践 AWS ，可帮助组织制定高效且有效的计划，以成功迁移到云端。AWS CAF将指导分为六个重点领域，称为视角：业务、人员、治理、平台、安全和运营。业务、人员和治理角度侧重于业务技能和流程；平台、安全和运营角度侧重于技术技能和流程。例如，人员角度针对的是负责人力资源（HR）、人员配置职能和人员管理的利益相关者。从这个角度来看，AWS CAF 为人员发展、培训和沟通提供了指导，以帮助组织为成功采用云做好准备。有关更多信息，请参阅 AWS CAF 网站和 AWS CAF 白皮书。

AWS 工作负载资格框架 (AWS WQF)

一种评估数据库迁移工作负载、推荐迁移策略和提供工作估算的工具。 AWS WQF 包含在 AWS Schema Conversion Tool (AWS SCT) 中。它用来分析数据库架构和代码对象、应用程序代码、依赖关系和性能特征，并提供评测报告。

B

坏机器人

旨在破坏个人或组织或对其造成伤害的机器人。

BCP

参见业务连续性计划。

行为图

一段时间内资源行为和交互的统一交互式视图。您可以使用 Amazon Detective 的行为图来检查失败的登录尝试、可疑的 API 调用和类似的操作。有关更多信息，请参阅 Detective 文档中的行为图中的数据。

大端序系统

一个先存储最高有效字节的系统。另请参见字节顺序。

二进制分类

一种预测二进制结果（两个可能的类别之一）的过程。例如，您的 ML 模型可能需要预测诸如“该电子邮件是否为垃圾邮件？”或“这个产品是书还是汽车？”之类的问题

bloom 筛选条件

一种概率性、内存高效的数据结构，用于测试元素是否为集合的成员。

蓝/绿部署

一种部署策略，您可以创建两个独立但完全相同的环境。在一个环境中运行当前的应用程序版本（蓝色），在另一个环境中运行新的应用程序版本（绿色）。此策略可帮助您在影响最小的情况下快速回滚。

自动程序

一种通过互联网运行自动任务并模拟人类活动或互动的软件应用程序。有些机器人是有用或有益的，例如在互联网上索引信息的网络爬虫。其他一些被称为恶意机器人的机器人旨在破坏个人或组织或对其造成伤害。

僵尸网络

被恶意软件感染并受单方（称为机器人牧民或机器人操作员）控制的机器人网络。僵尸网络是最著名的扩展机器人及其影响力的机制。

分支

代码存储库的一个包含区域。在存储库中创建的第一个分支是主分支。您可以从现有分支创建新分支，然后在新分支中开发功能或修复错误。为构建功能而创建的分支通常称为功能分支。当功能可以发布时，将功能分支合并回主分支。有关更多信息，请参阅关于分支（GitHub 文档）。

破碎的玻璃通道

在特殊情况下，通过批准的流程，用户 AWS 账户 可以快速访问他们通常没有访问权限的内容。有关更多信息，请参阅 Well -Architected 指南中的 “实施破碎玻璃程序” 指示 AWS 器。

棕地策略

您环境中的现有基础设施。在为系统架构采用棕地策略时，您需要围绕当前系统和基础设施的限制来设计架构。如果您正在扩展现有基础设施，则可以将棕地策略和全新策略混合。

缓冲区缓存

存储最常访问的数据的内存区域。

业务能力

企业如何创造价值（例如，销售、客户服务或营销）。微服务架构和开发决策可以由业务能力驱动。有关更多信息，请参阅在 AWS上运行容器化微服务白皮书中的围绕业务能力进行组织部分。

业务连续性计划（BCP）

一项计划，旨在应对大规模迁移等破坏性事件对运营的潜在影响，并使企业能够快速恢复运营。

C

CAF

参见AWS 云采用框架。

金丝雀部署

向最终用户缓慢而渐进地发布版本。当你有信心时，你可以部署新版本并全部替换当前版本。

CCoE

参见云卓越中心。

CDC

请参阅变更数据捕获。

更改数据捕获（CDC）

跟踪数据来源（如数据库表）的更改并记录有关更改的元数据的过程。您可以将 CDC 用于各种目的，例如审计或复制目标系统中的更改以保持同步。

混沌工程

故意引入故障或破坏性事件来测试系统的弹性。您可以使用 AWS Fault Injection Service (AWS FIS) 来执行实验，对您的 AWS 工作负载施加压力并评估其响应。

CI/CD

查看持续集成和持续交付。

分类

一种有助于生成预测的分类流程。分类问题的 ML 模型预测离散值。离散值始终彼此不同。例如，一个模型可能需要评估图像中是否有汽车。

客户端加密

在目标 AWS 服务 收到数据之前，对数据进行本地加密。

云卓越中心 (CCoE)

一个多学科团队，负责推动整个组织的云采用工作，包括开发云最佳实践、调动资源、制定迁移时间表、领导组织完成大规模转型。有关更多信息，请参阅 AWS Cloud 企业战略博客上的 CCoE 帖子。

云计算

通常用于远程数据存储和 IoT 设备管理的云技术。云计算通常与边缘计算技术相关。

云运营模型

在 IT 组织中，一种用于构建、完善和优化一个或多个云环境的运营模型。有关更多信息，请参阅构建您的云运营模型。

云采用阶段

组织迁移到以下阶段时通常会经历四个阶段 AWS Cloud：

- 项目 - 出于概念验证和学习目的，开展一些与云相关的项目
- 基础 — 进行基础投资以扩大云采用率（例如，创建 landing zone、定义 CCo E、建立运营模型）
- 迁移 - 迁移单个应用程序
- 重塑 - 优化产品和服务，在云中创新

Stephen Orban在 AWS Cloud 企业战略博客的博客文章《云优先之旅和采用阶段》中定义了这些阶段。有关它们与 AWS 迁移策略的关系的信息，请参阅迁移准备指南。

CMDB

参见配置管理数据库。

代码存储库

通过版本控制过程存储和更新源代码和其他资产（如文档、示例和脚本）的位置。常见的云存储库包括GitHub或Bitbucket Cloud。每个版本的代码都称为一个分支。在微服务结构中，每个存储库都专门用于一个功能。单个 CI/CD 管道可以使用多个存储库。

冷缓存

一种空的、填充不足或包含过时或不相关数据的缓冲区缓存。这会影响性能，因为数据库实例必须从主内存或磁盘读取，这比从缓冲区缓存读取要慢。

冷数据

很少访问的数据，且通常是历史数据。查询此类数据时，通常可以接受慢速查询。将这些数据转移到性能较低且成本更低的存储层或类别可以降低成本。

计算机视觉 (CV)

一种 AI 领域，它使用机器学习来分析和提取数字图像和视频等视觉格式中的信息。例如，Amazon SageMaker AI 为 CV 提供了图像处理算法。

配置偏差

对于工作负载，配置会从预期状态发生变化。这可能会导致工作负载变得不合规，而且通常是渐进的，不是故意的。

配置管理数据库（CMDB）

一种存储库，用于存储和管理有关数据库及其 IT 环境的信息，包括硬件和软件组件及其配置。您通常在迁移的产品组合发现和分析阶段使用来自 CMDB 的数据。

合规性包

一系列 AWS Config 规则和补救措施，您可以汇编这些规则和补救措施，以自定义您的合规性和安全性检查。您可以使用 YAML 模板将一致性包作为单个实体部署在 AWS 账户 和区域或整个组织中。有关更多信息，请参阅 AWS Config 文档中的一致性包。

持续集成和持续交付（CI/CD）

自动执行软件发布过程的源代码、构建、测试、暂存和生产阶段的过程。 CI/CD 通常被描述为管道。 CI/CD 可以帮助您实现流程自动化、提高生产力、提高代码质量和更快地交付。有关更多信息，请参阅持续交付的优势。CD 也可以表示持续部署。有关更多信息，请参阅持续交付与持续部署。

CV

参见计算机视觉。

D

静态数据

网络中静止的数据，例如存储中的数据。

数据分类

根据网络中数据的关键性和敏感性对其进行识别和分类的过程。它是任何网络安全风险管理策略的关键组成部分，因为它可以帮助您确定对数据的适当保护和保留控制。数据分类是 Well-Architecte AWS d Framework 中安全支柱的一个组成部分。有关详细信息，请参阅数据分类。

数据漂移

生产数据与用来训练机器学习模型的数据之间的有意义差异，或者输入数据随时间推移的有意义变化。数据漂移可能降低机器学习模型预测的整体质量、准确性和公平性。

传输中数据

在网络中主动移动的数据，例如在网络资源之间移动的数据。

数据网格

一种架构框架，可提供分布式、去中心化的数据所有权以及集中式管理和治理。

数据最少化

仅收集并处理绝对必要数据的原则。在中进行数据最小化 AWS Cloud 可以降低隐私风险、成本和分析碳足迹。

数据边界

AWS 环境中的一组预防性防护措施，可帮助确保只有可信身份才能访问来自预期网络的可信资源。有关更多信息，请参阅在上构建数据边界。 AWS

数据预处理

将原始数据转换为 ML 模型易于解析的格式。预处理数据可能意味着删除某些列或行，并处理缺失、不一致或重复的值。

数据溯源

在数据的整个生命周期跟踪其来源和历史的过程，例如数据如何生成、传输和存储。

数据主体

正在收集和处理其数据的个人。

数据仓库

一种支持商业智能（例如分析）的数据管理系统。数据仓库通常包含大量历史数据，通常用于查询和分析。

数据库定义语言（DDL）

在数据库中创建或修改表和对象结构的语句或命令。

数据库操作语言（DML）

在数据库中修改（插入、更新和删除）信息的语句或命令。

DDL

参见数据库定义语言。

深度融合

组合多个深度学习模型进行预测。您可以使用深度融合来获得更准确的预测或估算预测中的不确定性。

深度学习

一个 ML 子字段使用多层人工神经网络来识别输入数据和感兴趣的目标变量之间的映射。

defense-in-depth

一种信息安全方法，经过深思熟虑，在整个计算机网络中分层实施一系列安全机制和控制措施，以保护网络及其中数据的机密性、完整性和可用性。当你采用这种策略时 AWS，你会在 AWS

Organizations 结构的不同层面添加多个控件来帮助保护资源。例如，一种 defense-in-depth方法可以结合多因素身份验证、网络分段和加密。

委托管理员

在中 AWS Organizations，兼容的服务可以注册 AWS 成员帐户来管理组织的帐户并管理该服务的权限。此账户被称为该服务的委托管理员。有关更多信息和兼容服务列表，请参阅 AWS Organizations 文档中使用 AWS Organizations的服务。

后

使应用程序、新功能或代码修复在目标环境中可用的过程。部署涉及在代码库中实现更改，然后在应用程序的环境中构建和运行该代码库。

开发环境

参见环境。

侦测性控制

一种安全控制，在事件发生后进行检测、记录日志和发出警报。这些控制是第二道防线，提醒您注意绕过现有预防性控制的安全事件。有关更多信息，请参阅在 AWS上实施安全控制中的侦测性控制。

开发价值流映射 (DVSM)

用于识别对软件开发生命周期中的速度和质量产生不利影响的限制因素并确定其优先级的流程。DVSM 扩展了最初为精益生产实践设计的价值流映射流程。其重点关注在软件开发过程中创造和转移价值所需的步骤和团队。

数字孪生

真实世界系统的虚拟再现，如建筑物、工厂、工业设备或生产线。数字孪生支持预测性维护、远程监控和生产优化。

维度表

在星型架构中，一种较小的表，其中包含事实表中定量数据的数据属性。维度表属性通常是文本字段或行为类似于文本的离散数字。这些属性通常用于查询约束、筛选和结果集标注。

灾难

阻止工作负载或系统在其主要部署位置实现其业务目标的事件。这些事件可能是自然灾害、技术故障或人为操作的结果，例如无意的配置错误或恶意软件攻击。

灾难恢复 (DR)

您用来最大限度地减少灾难造成的停机时间和数据丢失的策略和流程。有关更多信息，请参阅 Well-Architected Fr ame AWS work 中的 “工作负载灾难恢复：云端 AWS 恢复”。

DML

参见数据库操作语言。

领域驱动设计

一种开发复杂软件系统的方法，通过将其组件连接到每个组件所服务的不断发展的领域或核心业务目标。Eric Evans 在其著作领域驱动设计：软件核心复杂性应对之道（Boston: Addison-Wesley Professional, 2003）中介绍了这一概念。有关如何将领域驱动设计与 strangler fig 模式结合使用的信息，请参阅使用容器和 Amazon API Gateway 逐步将原有的 Microsoft ASP.NET（ASMX）Web 服务现代化。

DR

参见灾难恢复。

漂移检测

跟踪与基准配置的偏差。例如，您可以使用 AWS CloudFormation 来检测系统资源中的偏差，也可以使用 AWS Control Tower 来检测着陆区中可能影响监管要求合规性的变化。

DVSM

参见开发价值流映射。

E

EDA

参见探索性数据分析。

EDI

参见电子数据交换。

边缘计算

该技术可提高位于 IoT 网络边缘的智能设备的计算能力。与云计算相比，边缘计算可以减少通信延迟并缩短响应时间。

电子数据交换 (EDI)

组织之间自动交换业务文档。有关更多信息，请参阅什么是电子数据交换。

加密

一种将人类可读的纯文本数据转换为密文的计算过程。

加密密钥

由加密算法生成的随机位的加密字符串。密钥的长度可能有所不同，而且每个密钥都设计为不可预测且唯一。

字节顺序

字节在计算机内存中的存储顺序。大端序系统先存储最高有效字节。小端序系统先存储最低有效字节。

端点

参见服务端点。

端点服务

一种可以在虚拟私有云（VPC）中托管，与其他用户共享的服务。您可以使用其他 AWS 账户 或 AWS Identity and Access Management (IAM) 委托人创建终端节点服务， AWS PrivateLink 并向其授予权限。这些账户或主体可通过创建接口 VPC 端点来私密地连接到您的端点服务。有关更多信息，请参阅 Amazon Virtual Private Cloud（Amazon VPC）文档中的创建端点服务。

企业资源规划 (ERP)

一种自动化和管理企业关键业务流程（例如会计、MES 和项目管理）的系统。

信封加密

用另一个加密密钥对加密密钥进行加密的过程。有关更多信息，请参阅 AWS Key Management Service (AWS KMS) 文档中的信封加密。

环境

正在运行的应用程序的实例。以下是云计算中常见的环境类型：

- 开发环境 — 正在运行的应用程序的实例，只有负责维护应用程序的核心团队才能使用。开发环境用于测试更改，然后再将其提升到上层环境。这类环境有时称为测试环境。
- 下层环境 — 应用程序的所有开发环境，比如用于初始构建和测试的环境。

- 生产环境 — 最终用户可以访问的正在运行的应用程序的实例。在 CI/CD 管道中，生产环境是最后一个部署环境。
- 上层环境 — 除核心开发团队以外的用户可以访问的所有环境。这可能包括生产环境、预生产环境和用户验收测试环境。

epic

在敏捷方法学中，有助于组织工作和确定优先级的功能类别。epics 提供了对需求和实施任务的总体描述。例如， AWS CAF 安全史诗包括身份和访问管理、侦探控制、基础设施安全、数据保护和事件响应。有关 AWS 迁移策略中 epics 的更多信息，请参阅计划实施指南。

ERP

参见企业资源规划。

探索性数据分析（EDA）

分析数据集以了解其主要特征的过程。您收集或汇总数据，并进行初步调查，以发现模式、检测异常并检查假定情况。EDA 通过计算汇总统计数据和创建数据可视化得以执行。

F

事实表

星形架构中的中心表。它存储有关业务运营的定量数据。通常，事实表包含两种类型的列：包含度量的列和包含维度表外键的列。

失败得很快

一种使用频繁和增量测试来缩短开发生命周期的理念。这是敏捷方法的关键部分。

故障隔离边界

在中 AWS Cloud，诸如可用区 AWS 区域、控制平面或数据平面之类的边界，它限制了故障的影响并有助于提高工作负载的弹性。有关更多信息，请参阅AWS 故障隔离边界。

功能分支

参见分支。

特征

您用来进行预测的输入数据。例如，在制造环境中，特征可能是定期从生产线捕获的图像。

特征重要性

特征对于模型预测的重要性。这通常表示为数值分数，可以通过各种技术进行计算，例如 Shapley 加法解释（SHAP）和积分梯度。有关更多信息，请参阅使用机器学习模型的可解释性 AWS。

功能转换

为 ML 流程优化数据，包括使用其他来源丰富数据、扩展值或从单个数据字段中提取多组信息。这使得 ML 模型能从数据中获益。例如，如果您将“2021-05-27 00:15:37”日期分解为“2021”、“五月”、“星期四”和“15”，则可以帮助学习与不同数据成分相关的算法学习精细模式。

少量提示

在要求法学硕士执行类似任务之前，向其提供少量示例，以演示该任务和所需的输出。这种技术是情境学习的应用，模型可以从提示中嵌入的示例（镜头）中学习。对于需要特定格式、推理或领域知识的任务，Few-shot 提示可能非常有效。另请参见零镜头提示。

FGAC

请参阅精细的访问控制。

精细访问控制 (FGAC)

使用多个条件允许或拒绝访问请求。

快闪迁移

一种数据库迁移方法，它使用连续的数据复制，通过更改数据捕获在尽可能短的时间内迁移数据，而不是使用分阶段的方法。目标是将停机时间降至最低。

FM

参见基础模型。

基础模型 (FM)

一个大型深度学习神经网络，一直在广义和未标记数据的大量数据集上进行训练。 FMs 能够执行各种各样的一般任务，例如理解语言、生成文本和图像以及用自然语言进行对话。有关更多信息，请参阅什么是基础模型。

G

生成式人工智能

人工智能模型的子集，这些模型已经过大量数据训练，可以使用简单的文本提示来创建新的内容和工件，例如图像、视频、文本和音频。有关更多信息，请参阅什么是生成式 AI。

地理封锁

请参阅地理限制。

地理限制（地理阻止）

在 Amazon 中 CloudFront，一种阻止特定国家/地区的用户访问内容分发的选项。您可以使用允许列表或阻止列表来指定已批准和已禁止的国家/地区。有关更多信息，请参阅 CloudFront 文档中的限制内容的地理分布。

GitFlow 工作流程

一种方法，在这种方法中，下层和上层环境在源代码存储库中使用不同的分支。Gitflow 工作流程被认为是传统的，而基于主干的工作流程是现代的首选方法。

金色影像

系统或软件的快照，用作部署该系统或软件的新实例的模板。例如，在制造业中，黄金映像可用于在多个设备上配置软件，并有助于提高设备制造运营的速度、可扩展性和生产力。

全新策略

在新环境中缺少现有基础设施。在对系统架构采用全新策略时，您可以选择所有新技术，而不受对现有基础设施（也称为棕地）兼容性的限制。如果您正在扩展现有基础设施，则可以将棕地策略和全新策略混合。

防护机制

一项高级规则，可帮助管理各组织单位的资源、策略和合规性 (OUs)。预防性防护机制会执行策略以确保符合合规性标准。它们是使用服务控制策略和 IAM 权限边界实现的。侦测性防护机制会检测策略违规和合规性问题，并生成警报以进行修复。它们通过使用 AWS Config、、Amazon、 AWS Security Hub GuardDuty AWS Trusted Advisor、Amazon Inspector 和自定义 AWS Lambda 支票来实现。

H

HA

参见高可用性。

异构数据库迁移

将源数据库迁移到使用不同数据库引擎的目标数据库（例如，从 Oracle 迁移到 Amazon Aurora）。异构迁移通常是重新架构工作的一部分，而转换架构可能是一项复杂的任务。AWS 提供了 AWS SCT 来帮助实现架构转换。

高可用性 (HA)

在遇到挑战或灾难时，工作负载无需干预即可连续运行的能力。HA 系统旨在自动进行故障转移、持续提供良好性能，并以最小的性能影响处理不同负载和故障。

历史数据库现代化

一种用于实现运营技术 (OT) 系统现代化和升级以更好满足制造业需求的方法。历史数据库是一种用于收集和存储工厂中各种来源数据的数据库。

抵制数据

从用于训练机器学习模型的数据集中扣留的一部分带有标签的历史数据。通过将模型预测与抵制数据进行比较，您可以使用抵制数据来评估模型性能。

同构数据库迁移

将源数据库迁移到共享同一数据库引擎的目标数据库（例如，从 Microsoft SQL Server 迁移到 Amazon RDS for SQL Server）。同构迁移通常是更换主机或更换平台工作的一部分。您可以使用本机数据库实用程序来迁移架构。

热数据

经常访问的数据，例如实时数据或近期的转化数据。这些数据通常需要高性能存储层或存储类别才能提供快速的查询响应。

修补程序

针对生产环境中关键问题的紧急修复。由于其紧迫性，修补程序通常是在典型的 DevOps 发布工作流程之外进行的。

hypercare 周期

割接之后，迁移团队立即管理和监控云中迁移的应用程序以解决任何问题的时间段。通常，这个周期持续 1-4 天。在 hypercare 周期结束时，迁移团队通常会将应用程序的责任移交给云运营团队。

我

IaC

参见基础设施即代码。

基于身份的策略

附加到一个或多个 IAM 委托人的策略，用于定义他们在 AWS Cloud 环境中的权限。

空闲应用程序

90 天内平均 CPU 和内存使用率在 5% 到 20% 之间的应用程序。在迁移项目中，通常会停用这些应用程序或将其保留在本地。

IIoT

参见工业物联网。

不可变的基础架构

一种为生产工作负载部署新基础架构，而不是更新、修补或修改现有基础架构的模型。不可变基础架构本质上比可变基础架构更一致、更可靠、更可预测。有关更多信息，请参阅 Well-Architected Framework 中的使用不可变基础架构 AWS 部署最佳实践。

入站（入口）VPC

在 AWS 多账户架构中，一种接受、检查和路由来自应用程序外部的网络连接的 VPC。AWS 安全参考架构建议设置您的网络帐户，包括入站、出站和检查， VPCs 以保护您的应用程序与更广泛的互联网之间的双向接口。

增量迁移

一种割接策略，在这种策略中，您可以将应用程序分成小部分进行迁移，而不是一次性完整割接。例如，您最初可能只将几个微服务或用户迁移到新系统。在确认一切正常后，您可以逐步迁移其他微服务或用户，直到停用遗留系统。这种策略降低了大规模迁移带来的风险。

工业 4.0

该术语由克劳斯·施瓦布（Klaus Schwab）于2016年推出，指的是通过连接、实时数据、自动化、分析和人工智能/机器学习的进步实现制造流程的现代化。

基础设施

应用程序环境中包含的所有资源和资产。

基础设施即代码（IaC）

通过一组配置文件预置和管理应用程序基础设施的过程。IaC 旨在帮助您集中管理基础设施、实现资源标准化和快速扩展，使新环境具有可重复性、可靠性和一致性。

工业物联网 (IIoT)

在工业领域使用联网的传感器和设备，例如制造业、能源、汽车、医疗保健、生命科学和农业。有关更多信息，请参阅制定工业物联网 (IIoT) 数字化转型战略。

检查 VPC

在 AWS 多账户架构中，一种集中式 VPC，用于管理对 VPCs（相同或不同 AWS 区域）、互联网和本地网络之间的网络流量的检查。AWS 安全参考架构建议设置您的网络帐户，包括入站、出站和检查， VPCs 以保护您的应用程序与更广泛的互联网之间的双向接口。

物联网（IoT）

由带有嵌入式传感器或处理器的连接物理对象组成的网络，这些传感器或处理器通过互联网或本地通信网络与其他设备和系统进行通信。有关更多信息，请参阅什么是 IoT？

可解释性

它是机器学习模型的一种特征，描述了人类可以理解模型的预测如何取决于其输入的程度。有关更多信息，请参阅使用机器学习模型的可解释性 AWS。

IoT

参见物联网。

IT 信息库（ITIL）

提供 IT 服务并使这些服务符合业务要求的一套最佳实践。ITIL 是 ITSM 的基础。

IT 服务管理（ITSM）

为组织设计、实施、管理和支持 IT 服务的相关活动。有关将云运营与 ITSM 工具集成的信息，请参阅运营集成指南。

ITIL

请参阅 IT 信息库。

ITSM

请参阅 IT 服务管理。

L

基于标签的访问控制 (LBAC)

强制访问控制 (MAC) 的一种实施方式，其中明确为用户和数据本身分配了安全标签值。用户安全标签和数据安全标签之间的交集决定了用户可以看到哪些行和列。

登录区

landing zone 是一个架构精良的多账户 AWS 环境，具有可扩展性和安全性。这是一个起点，您的组织可以从这里放心地在安全和基础设施环境中快速启动和部署工作负载和应用程序。有关登录区的更多信息，请参阅设置安全且可扩展的多账户 AWS 环境。

大型语言模型 (LLM)

一种基于大量数据进行预训练的深度学习 AI 模型。法学硕士可以执行多项任务，例如回答问题、总结文档、将文本翻译成其他语言以及完成句子。有关更多信息，请参阅什么是 LLMs。

大规模迁移

迁移 300 台或更多服务器。

LBAC

请参阅基于标签的访问控制。

最低权限

授予执行任务所需的最低权限的最佳安全实践。有关更多信息，请参阅 IAM 文档中的应用最低权限许可。

直接迁移

见 7 R。

小端序系统

一个先存储最低有效字节的系统。另请参见字节顺序。

LLM

参见大型语言模型。

下层环境

参见环境。

M

机器学习（ML）

一种使用算法和技术进行模式识别和学习的人工智能。ML 对记录的数据（例如物联网（IoT）数据）进行分析和学习，以生成基于模式的统计模型。有关更多信息，请参阅机器学习。

主分支

参见分支。

恶意软件

旨在危害计算机安全或隐私的软件。恶意软件可能会破坏计算机系统、泄露敏感信息或获得未经授权的访问。恶意软件的示例包括病毒、蠕虫、勒索软件、特洛伊木马、间谍软件和键盘记录器。

托管服务

AWS 服务 它 AWS 运行基础设施层、操作系统和平台，您可以访问端点来存储和检索数据。亚马逊简单存储服务 (Amazon S3) Service 和 Amazon DynamoDB 就是托管服务的示例。这些服务也称为抽象服务。

制造执行系统 (MES)

一种软件系统，用于跟踪、监控、记录和控制在车间将原材料转化为成品的生产过程。

MAP

参见迁移加速计划。

机制

一个完整的过程，在此过程中，您可以创建工具，推动工具的采用，然后检查结果以进行调整。机制是一种在运行过程中自我增强和改进的循环。有关更多信息，请参阅在 Well-Architect AWS ed 框架中构建机制。

成员账户

AWS 账户 除属于组织中的管理账户之外的所有账户 AWS Organizations。一个账户一次只能是一个组织的成员。

MES

参见制造执行系统。

消息队列遥测传输 (MQTT)

一种基于发布/订阅模式的轻量级 machine-to-machine (M2M) 通信协议，适用于资源受限的物联网设备。

微服务

一种小型的独立服务，通过明确的定义进行通信 APIs ，通常由小型的独立团队拥有。例如，保险系统可能包括映射到业务能力（如销售或营销）或子域（如购买、理赔或分析）的微服务。微服务

的好处包括敏捷、灵活扩展、易于部署、可重复使用的代码和恢复能力。有关更多信息，请参阅使用 AWS 无服务器服务集成微服务。

微服务架构

一种使用独立组件构建应用程序的方法，这些组件将每个应用程序进程作为微服务运行。这些微服务使用轻量级通过定义明确的接口进行通信。 APIs该架构中的每个微服务都可以更新、部署和扩展，以满足对应用程序特定功能的需求。有关更多信息，请参阅在上实现微服务。 AWS

迁移加速计划（MAP）

AWS 该计划提供咨询支持、培训和服务，以帮助组织为迁移到云奠定坚实的运营基础，并帮助抵消迁移的初始成本。MAP 提供了一种以系统的方式执行遗留迁移的迁移方法，以及一套用于自动执行和加速常见迁移场景的工具。

大规模迁移

将大部分应用程序组合分波迁移到云中的过程，在每一波中以更快的速度迁移更多应用程序。本阶段使用从早期阶段获得的最佳实践和经验教训，实施由团队、工具和流程组成的迁移工厂，通过自动化和敏捷交付简化工作负载的迁移。这是 AWS 迁移策略的第三阶段。

迁移工厂

跨职能团队，通过自动化、敏捷的方法简化工作负载迁移。迁移工厂团队通常包括运营、业务分析师和所有者、迁移工程师、开发人员和冲刺 DevOps 领域的专业人员。20% 到 50% 的企业应用程序组合由可通过工厂方法优化的重复模式组成。有关更多信息，请参阅本内容集中有关迁移工厂的讨论和云迁移工厂指南。

迁移元数据

有关完成迁移所需的应用程序和服务器的信息。每种迁移模式都需要一套不同的迁移元数据。迁移元数据的示例包括目标子网、安全组和 AWS 账户。

迁移模式

一种可重复的迁移任务，详细列出了迁移策略、迁移目标以及所使用的迁移应用程序或服务。示例： EC2 使用 AWS 应用程序迁移服务重新托管向 Amazon 的迁移。

迁移组合评测（MPA）

一种在线工具，可提供信息，用于验证迁移到的业务案例。 AWS Cloud MPA 提供了详细的组合评测（服务器规模调整、定价、TCO 比较、迁移成本分析）以及迁移计划（应用程序数据分析和数据收集、应用程序分组、迁移优先级排序和波次规划）。所有 AWS 顾问和 APN 合作伙伴顾问均可免费使用 MPA 工具（需要登录）。

迁移准备情况评测（MRA）

使用 AWS CAF 深入了解组织的云就绪状态、确定优势和劣势以及制定行动计划以缩小已发现差距的过程。有关更多信息，请参阅迁移准备指南。MRA 是 AWS 迁移策略的第一阶段。

迁移策略

用于将工作负载迁移到的方法 AWS Cloud。有关更多信息，请参阅此词汇表中的 “7 R” 条目，并参阅调动组织以加快大规模迁移。

ML

参见机器学习。

现代化

将过时的（原有的或单体）应用程序及其基础设施转变为云中敏捷、弹性和高度可用的系统，以降低成本、提高效率和利用创新。有关更多信息，请参阅中的应用程序现代化策略。 AWS Cloud

现代化准备情况评估

一种评估方式，有助于确定组织应用程序的现代化准备情况；确定收益、风险和依赖关系；确定组织能够在多大程度上支持这些应用程序的未来状态。评估结果是目标架构的蓝图、详细说明现代化进程发展阶段和里程碑的路线图以及解决已发现差距的行动计划。有关更多信息，请参阅中的评估应用程序的现代化准备情况 AWS Cloud。

单体应用程序（单体式）

作为具有紧密耦合进程的单个服务运行的应用程序。单体应用程序有几个缺点。如果某个应用程序功能的需求激增，则必须扩展整个架构。随着代码库的增长，添加或改进单体应用程序的功能也会变得更加复杂。若要解决这些问题，可以使用微服务架构。有关更多信息，请参阅将单体分解为微服务。

MPA

参见迁移组合评估。

MQTT

请参阅消息队列遥测传输。

多分类器

一种帮助为多个类别生成预测（预测两个以上结果之一）的过程。例如，ML 模型可能会询问“这个产品是书、汽车还是手机？”或“此客户最感兴趣什么类别的产品？”

可变基础架构

一种用于更新和修改现有生产工作负载基础架构的模型。为了提高一致性、可靠性和可预测性，Well-Architect AWS ed Framework 建议使用不可变基础设施作为最佳实践。

O

OAC

请参阅源站访问控制。

OAI

参见源访问身份。

OCM

参见组织变更管理。

离线迁移

一种迁移方法，在这种方法中，源工作负载会在迁移过程中停止运行。这种方法会延长停机时间，通常用于小型非关键工作负载。

OI

参见运营集成。

OLA

参见运营层协议。

在线迁移

一种迁移方法，在这种方法中，源工作负载无需离线即可复制到目标系统。在迁移过程中，连接工作负载的应用程序可以继续运行。这种方法的停机时间为零或最短，通常用于关键生产工作负载。

OPC-UA

参见开放流程通信-统一架构。

开放流程通信-统一架构 (OPC-UA)

一种用于工业自动化的 machine-to-machine（M2M）通信协议。OPC-UA 提供了数据加密、身份验证和授权方案的互操作性标准。

运营级别协议（OLA）

一项协议，阐明了 IT 职能部门承诺相互交付的内容，以支持服务水平协议（SLA）。

运营准备情况审查 (ORR)

一份问题清单和相关的最佳实践，可帮助您理解、评估、预防或缩小事件和可能的故障的范围。有关更多信息，请参阅 Well-Architecte AWS d Frame work 中的运营准备情况评估 (ORR)。

操作技术 (OT)

与物理环境配合使用以控制工业运营、设备和基础设施的硬件和软件系统。在制造业中，OT 和信息技术 (IT) 系统的集成是工业 4.0 转型的重点。

运营整合（OI）

在云中实现运营现代化的过程，包括就绪计划、自动化和集成。有关更多信息，请参阅运营整合指南。

组织跟踪

由此创建的跟踪 AWS CloudTrail ，用于记录组织 AWS 账户 中所有人的所有事件 AWS Organizations。该跟踪是在每个 AWS 账户 中创建的，属于组织的一部分，并跟踪每个账户的活动。有关更多信息，请参阅 CloudTrail文档中的为组织创建跟踪。

组织变革管理（OCM）

一个从人员、文化和领导力角度管理重大、颠覆性业务转型的框架。OCM 通过加快变革采用、解决过渡问题以及推动文化和组织变革，帮助组织为新系统和战略做好准备和过渡。在 AWS 迁移策略中，该框架被称为人员加速，因为云采用项目需要变更的速度。有关更多信息，请参阅 OCM 指南。

来源访问控制（OAC）

在中 CloudFront，一个增强的选项，用于限制访问以保护您的亚马逊简单存储服务 (Amazon S3) 内容。OAC 全部支持所有 S3 存储桶 AWS 区域、使用 AWS KMS (SSE-KMS) 进行服务器端加密，以及对 S3 存储桶的动态PUT和DELETE请求。

来源访问身份（OAI）

在中 CloudFront，一个用于限制访问权限以保护您的 Amazon S3 内容的选项。当您使用 OAI 时，CloudFront 会创建一个 Amazon S3 可以对其进行身份验证的委托人。经过身份验证的委托人只能通过特定 CloudFront 分配访问 S3 存储桶中的内容。另请参阅 OAC，其中提供了更精细和增强的访问控制。

ORR

参见运营准备情况审查。

OT

参见运营技术。

出站（出口）VPC

在 AWS 多账户架构中，一种处理从应用程序内部启动的网络连接的 VPC。AWS 安全参考架构建议设置您的网络帐户，包括入站、出站和检查， VPCs 以保护您的应用程序与更广泛的互联网之间的双向接口。

P

权限边界

附加到 IAM 主体的 IAM 管理策略，用于设置用户或角色可以拥有的最大权限。有关更多信息，请参阅 IAM 文档中的权限边界。

个人身份信息 (PII)

直接查看其他相关数据或与之配对时可用于合理推断个人身份的信息。PII 的示例包括姓名、地址和联系信息。

PII

查看个人身份信息。

playbook

一套预定义的步骤，用于捕获与迁移相关的工作，例如在云中交付核心运营功能。playbook 可以采用脚本、自动化运行手册的形式，也可以是操作现代化环境所需的流程或步骤的摘要。

PLC

参见可编程逻辑控制器。

PLM

参见产品生命周期管理。

policy

一个对象，可以在中定义权限（参见基于身份的策略）、指定访问条件（参见基于资源的策略）或定义组织中所有账户的最大权限 AWS Organizations （参见服务控制策略）。

多语言持久性

根据数据访问模式和其他要求，独立选择微服务的数据存储技术。如果您的微服务采用相同的数据存储技术，它们可能会遇到实现难题或性能不佳。如果微服务使用最适合其需求的数据存储，则可以更轻松地实现微服务，并获得更好的性能和可扩展性。有关更多信息，请参阅在微服务中实现数据持久性。

组合评测

一个发现、分析和确定应用程序组合优先级以规划迁移的过程。有关更多信息，请参阅评估迁移准备情况。

谓词

返回`true`或的查询条件`false`，通常位于子`WHERE`句中。

谓词下推

一种数据库查询优化技术，可在传输前筛选查询中的数据。这减少了必须从关系数据库检索和处理的数据量，并提高了查询性能。

预防性控制

一种安全控制，旨在防止事件发生。这些控制是第一道防线，帮助防止未经授权的访问或对网络的意外更改。有关更多信息，请参阅在 AWS上实施安全控制中的预防性控制。

主体

中 AWS 可以执行操作和访问资源的实体。此实体通常是 IAM 角色的根用户或用户。 AWS 账户有关更多信息，请参阅 IAM 文档中角色术语和概念中的主体。

通过设计保护隐私

一种在整个开发过程中考虑隐私的系统工程方法。

私有托管区

一个容器，其中包含有关您希望 Amazon Route 53 如何响应针对一个或多个 VPCs域名及其子域名的 DNS 查询的信息。有关更多信息，请参阅 Route 53 文档中的私有托管区的使用。

主动控制

一种安全控制措施，旨在防止部署不合规的资源。这些控件会在资源配置之前对其进行扫描。如果资源与控件不兼容，则不会对其进行配置。有关更多信息，请参阅 AWS Control Tower 文档中的控制参考指南，并参见在上实施安全控制中的主动控制 AWS。

产品生命周期管理 (PLM)

在产品的整个生命周期中，从设计、开发和上市，到成长和成熟，再到衰落和移除，对产品进行数据和流程的管理。

生产环境

参见环境。

可编程逻辑控制器 (PLC)

在制造业中，一种高度可靠、适应性强的计算机，用于监控机器并实现制造过程自动化。

提示链接

使用一个 LLM 提示的输出作为下一个提示的输入，以生成更好的响应。该技术用于将复杂的任务分解为子任务，或者迭代地完善或扩展初步响应。它有助于提高模型响应的准确性和相关性，并允许获得更精细的个性化结果。

假名化

用占位符值替换数据集中个人标识符的过程。假名化可以帮助保护个人隐私。假名化数据仍被视为个人数据。

publish/subscribe (pub/sub)

一种支持微服务间异步通信的模式，以提高可扩展性和响应能力。例如，在基于微服务的 MES 中，微服务可以将事件消息发布到其他微服务可以订阅的频道。系统可以在不更改发布服务的情况下添加新的微服务。

Q

查询计划

一系列步骤，例如指令，用于访问 SQL 关系数据库系统中的数据。

查询计划回归

当数据库服务优化程序选择的最佳计划不如数据库环境发生特定变化之前时。这可能是由统计数据、约束、环境设置、查询参数绑定更改和数据库引擎更新造成的。

R

RACI 矩阵

参见负责任、负责、咨询、知情（RACI）。

RAG

请参见检索增强生成。

勒索软件

一种恶意软件，旨在阻止对计算机系统或数据的访问，直到付款为止。

RASCI 矩阵

参见负责任、负责、咨询、知情（RACI）。

RCAC

请参阅行和列访问控制。

只读副本

用于只读目的的数据库副本。您可以将查询路由到只读副本，以减轻主数据库的负载。

重新架构师

见 7 R。

恢复点目标 (RPO)

自上一个数据恢复点以来可接受的最长时间。这决定了从上一个恢复点到服务中断之间可接受的数据丢失情况。

恢复时间目标 (RTO)

服务中断和服务恢复之间可接受的最大延迟。

重构

见 7 R。

区域

地理区域内的 AWS 资源集合。每一个 AWS 区域 都相互隔离，彼此独立，以提供容错、稳定性和弹性。有关更多信息，请参阅指定 AWS 区域 您的账户可以使用的账户。

回归

一种预测数值的 ML 技术。例如，要解决“这套房子的售价是多少？”的问题 ML 模型可以使用线性回归模型，根据房屋的已知事实（如建筑面积）来预测房屋的销售价格。

重新托管

见 7 R。

版本

在部署过程中，推动生产环境变更的行为。

搬迁

见 7 R。

更换平台

见 7 R。

回购

见 7 R。

故障恢复能力

应用程序抵御中断或从中断中恢复的能力。在中规划弹性时，高可用性和灾难恢复是常见的考虑因素。 AWS Cloud有关更多信息，请参阅AWS Cloud 弹性。

基于资源的策略

一种附加到资源的策略，例如 AmazonS3 存储桶、端点或加密密钥。此类策略指定了允许哪些主体访问、支持的操作以及必须满足的任何其他条件。

责任、问责、咨询和知情（RACI）矩阵

定义参与迁移活动和云运营的所有各方的角色和责任的矩阵。矩阵名称源自矩阵中定义的责任类型：负责 (R)、问责 (A)、咨询 (C) 和知情 (I)。支持 (S) 类型是可选的。如果包括支持，则该矩阵称为 RASCI 矩阵，如果将其排除在外，则称为 RACI 矩阵。

响应性控制

一种安全控制，旨在推动对不良事件或偏离安全基线的情况进行修复。有关更多信息，请参阅在AWS上实施安全控制中的响应性控制。

保留

见 7 R。

退休

见 7 R。

检索增强生成（RAG）

一种生成式人工智能技术，其中法学硕士在生成响应之前引用其训练数据源之外的权威数据源。例如，RAG 模型可以对组织的知识库或自定义数据执行语义搜索。有关更多信息，请参阅什么是 RAG。

轮换

定期更新密钥以使攻击者更难访问凭据的过程。

行列访问控制 (RCAC)

使用已定义访问规则的基本、灵活的 SQL 表达式。RCAC 由行权限和列掩码组成。

RPO

参见恢复点目标。

RTO

参见恢复时间目标。

运行手册

执行特定任务所需的一套手动或自动程序。它们通常是为了简化重复性操作或高错误率的程序而设计的。

S

SAML 2.0

许多身份提供商 (IdPs) 使用的开放标准。此功能支持联合单点登录 (SSO)，因此用户无需在 IAM 中为组织中的所有人创建用户即可登录 AWS Management Console 或调用 AWS API 操作。有关基于 SAML 2.0 的联合身份验证的更多信息，请参阅 IAM 文档中的关于基于 SAML 2.0 的联合身份验证。

SCADA

参见监督控制和数据采集。

SCP

参见服务控制政策。

secret

在中 AWS Secrets Manager，您以加密形式存储的机密或受限信息，例如密码或用户凭证。它由密钥值及其元数据组成。密钥值可以是二进制、单个字符串或多个字符串。有关更多信息，请参阅 Secret s Manager 密钥中有什么？ 在 Secrets Manager 文档中。

安全性源于设计

一种在整个开发过程中考虑安全性的系统工程方法。

安全控制

一种技术或管理防护机制，可防止、检测或降低威胁行为体利用安全漏洞的能力。安全控制主要有四种类型：预防性、侦测、响应式和主动式。

安全加固

缩小攻击面，使其更能抵御攻击的过程。这可能包括删除不再需要的资源、实施授予最低权限的最佳安全实践或停用配置文件中不必要的功能等操作。

安全信息和事件管理（SIEM）系统

结合了安全信息管理（SIM）和安全事件管理（SEM）系统的工具和服务。SIEM 系统会收集、监控和分析来自服务器、网络、设备和其他来源的数据，以检测威胁和安全漏洞，并生成警报。

安全响应自动化

一种预定义和编程的操作，旨在自动响应或修复安全事件。这些自动化可作为侦探或响应式安全控制措施，帮助您实施 AWS 安全最佳实践。自动响应操作的示例包括修改 VPC 安全组、修补 Amazon EC2 实例或轮换证书。

服务器端加密

由接收数据的人在目的地对数据 AWS 服务 进行加密。

服务控制策略（SCP）

一种策略，用于集中控制组织中所有账户的权限 AWS Organizations。 SCPs 定义防护措施或限制管理员可以委托给用户或角色的操作。您可以使用 SCPs 允许列表或拒绝列表来指定允许或禁止哪些服务或操作。有关更多信息，请参阅 AWS Organizations 文档中的服务控制策略。

服务端点

的入口点的 URL AWS 服务。您可以使用端点，通过编程方式连接到目标服务。有关更多信息，请参阅 AWS 一般参考 中的 AWS 服务 端点。

服务水平协议（SLA）

一份协议，阐明了 IT 团队承诺向客户交付的内容，比如服务正常运行时间和性能。

服务级别指示器 (SLI)

对服务性能方面的衡量，例如其错误率、可用性或吞吐量。

服务级别目标 (SLO)

代表服务运行状况的目标指标，由服务级别指标衡量。

责任共担模式

描述您在云安全与合规方面共同承担 AWS 的责任的模型。 AWS 负责云的安全，而您则负责云中的安全。有关更多信息，请参阅责任共担模式。

SIEM

参见安全信息和事件管理系统。

单点故障 (SPOF)

应用程序的单个关键组件出现故障，可能会中断系统。

SLA

参见服务级别协议。

SLI

参见服务级别指标。

SLO

参见服务级别目标。

split-and-seed 模型

一种扩展和加速现代化项目的模式。随着新功能和产品发布的定义，核心团队会拆分以创建新的产品团队。这有助于扩展组织的能力和服务，提高开发人员的工作效率，支持快速创新。有关更多信息，请参阅中的分阶段实现应用程序现代化的方法。 AWS Cloud

恶作剧

参见单点故障。

星型架构

一种数据库组织结构，它使用一个大型事实表来存储交易数据或测量数据，并使用一个或多个较小的维度表来存储数据属性。此结构专为在数据仓库中使用或用于商业智能目的而设计。

strangler fig 模式

一种通过逐步重写和替换系统功能直至可以停用原有的系统来实现单体系统现代化的方法。这种模式用无花果藤作为类比，这种藤蔓成长为一棵树，最终战胜并取代了宿主。该模式是由 Martin Fowler 提出的，作为重写单体系统时管理风险的一种方法。有关如何应用此模式的示例，请参阅使用容器和 Amazon API Gateway 逐步将原有的 Microsoft ASP.NET（ASMX）Web 服务现代化。

子网

您的 VPC 内的一个 IP 地址范围。子网必须位于单个可用区中。

监控和数据采集 (SCADA)

在制造业中，一种使用硬件和软件来监控有形资产和生产操作的系统。

对称加密

一种加密算法，它使用相同的密钥来加密和解密数据。

综合测试

以模拟用户交互的方式测试系统，以检测潜在问题或监控性能。你可以使用 Amazon S CloudWatch ynthetics 来创建这些测试。

系统提示符

一种向法学硕士提供上下文、说明或指导方针以指导其行为的技术。系统提示有助于设置上下文并制定与用户交互的规则。

T

tags

键值对，用作组织资源的元数据。 AWS 标签可帮助您管理、识别、组织、搜索和筛选资源。有关更多信息，请参阅标记您的 AWS 资源。

目标变量

您在监督式 ML 中尝试预测的值。这也被称为结果变量。例如，在制造环境中，目标变量可能是产品缺陷。

任务列表

一种通过运行手册用于跟踪进度的工具。任务列表包含运行手册的概述和要完成的常规任务列表。对于每项常规任务，它包括预计所需时间、所有者和进度。

测试环境

参见环境。

训练

为您的 ML 模型提供学习数据。训练数据必须包含正确答案。学习算法在训练数据中查找将输入数据属性映射到目标（您希望预测的答案）的模式。然后输出捕获这些模式的 ML 模型。然后，您可以使用 ML 模型对不知道目标的新数据进行预测。

中转网关

一个网络传输中心，可用于将您的网络 VPCs 和本地网络互连。有关更多信息，请参阅 AWS Transit Gateway 文档中的什么是公交网关。

基于中继的工作流程

一种方法，开发人员在功能分支中本地构建和测试功能，然后将这些更改合并到主分支中。然后，按顺序将主分支构建到开发、预生产和生产环境。

可信访问权限

向您指定的服务授予权限，该服务可以代表您在其账户中执行任务。 AWS Organizations 当需要服务相关的角色时，受信任的服务会在每个账户中创建一个角色，为您执行管理任务。有关更多信息，请参阅 AWS Organizations 文档中的AWS Organizations 与其他 AWS 服务一起使用。

优化

更改训练过程的各个方面，以提高 ML 模型的准确性。例如，您可以通过生成标签集、添加标签，并在不同的设置下多次重复这些步骤来优化模型，从而训练 ML 模型。

双披萨团队

一个小 DevOps 团队，你可以用两个披萨来喂食。双披萨团队的规模可确保在软件开发过程中充分协作。

U

不确定性

这一概念指的是不精确、不完整或未知的信息，这些信息可能会破坏预测式 ML 模型的可靠性。不确定性有两种类型：认知不确定性是由有限的、不完整的数据造成的，而偶然不确定性是由数据中固有的噪声和随机性导致的。有关更多信息，请参阅量化深度学习系统中的不确定性指南。

无差别任务

也称为繁重工作，即创建和运行应用程序所必需的工作，但不能为最终用户提供直接价值或竞争优势。无差别任务的示例包括采购、维护和容量规划。

上层环境

参见环境。

V

vacuum 操作

一种数据库维护操作，包括在增量更新后进行清理，以回收存储空间并提高性能。

版本控制

跟踪更改的过程和工具，例如存储库中源代码的更改。

VPC 对等连接

两者之间的连接 VPCs ，允许您使用私有 IP 地址路由流量。有关更多信息，请参阅 Amazon VPC 文档中的什么是 VPC 对等连接。

漏洞

损害系统安全的软件缺陷或硬件缺陷。

W

热缓存

一种包含经常访问的当前相关数据的缓冲区缓存。数据库实例可以从缓冲区缓存读取，这比从主内存或磁盘读取要快。

暖数据

不常访问的数据。查询此类数据时，通常可以接受中速查询。

窗口函数

一个 SQL 函数，用于对一组以某种方式与当前记录相关的行进行计算。窗口函数对于处理任务很有用，例如计算移动平均线或根据当前行的相对位置访问行的值。

工作负载

一系列资源和代码，它们可以提供商业价值，如面向客户的应用程序或后端过程。

工作流

迁移项目中负责一组特定任务的职能小组。每个工作流都是独立的，但支持项目中的其他工作流。例如，组合工作流负责确定应用程序的优先级、波次规划和收集迁移元数据。组合工作流将这些资产交付给迁移工作流，然后迁移服务器和应用程序。

蠕虫

参见一次写入，多读。

WQF

参见AWS 工作负载资格框架。

一次写入，多次读取 (WORM)

一种存储模型，它可以一次写入数据并防止数据被删除或修改。授权用户可以根据需要多次读取数据，但他们无法对其进行更改。这种数据存储基础架构被认为是不可变的。

Z

零日漏洞利用

一种利用未修补漏洞的攻击，通常是恶意软件。

零日漏洞

生产系统中不可避免的缺陷或漏洞。威胁主体可能利用这种类型的漏洞攻击系统。开发人员经常因攻击而意识到该漏洞。

零镜头提示

向法学硕士提供执行任务的说明，但没有示例（镜头）可以帮助指导任务。法学硕士必须使用其预先训练的知识来处理任务。零镜头提示的有效性取决于任务的复杂性和提示的质量。另请参阅 few-shot 提示。

僵尸应用程序

平均 CPU 和内存使用率低于 5% 的应用程序。在迁移项目中，通常会停用这些应用程序。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。