Best practices with Amazon Q Developer for in-line and assistant code generation

AWS Prescriptive Guidance

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

AWS Prescriptive Guidance: Best practices with Amazon Q Developer for in-line and assistant code generation

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

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Introduction 1
Objectives 1
Developer workflows 3
Design and planning 4
Coding 4
Code review 5
Integration and deployment 5
Advanced capabilities 6
Amazon Q Developer code transformation 6
Amazon Q Developer customizations 6
Best coding practices 8
Best practices for onboarding 8
Prerequisites for Amazon Q Developer 8
Best practices when using Amazon Q Developer 8
Data privacy and content usage in Amazon Q Developer 9
Best practices for code generation 9
Best practices for code recommendations 10
Code examples 12
Python examples 12
Generate classes and functions 12
Document code 14
Generate algorithms 15
Generate unit tests 17
Java examples 18
Generate classes and functions 19
Document code 21
Generate algorithms 23
Generate unit tests 26
Chat examples 27
Ask about AWS services 27
Generate code 28
Generate unit tests 30
Explain code 32
Troubleshooting 35

Empty code generation 35
Continuous comments 36
Incorrect in-line code generation 37
Inadequate results from chats 42
FAQs 46
What is Amazon Q Developer? 46
How do I access Amazon Q Developer? 46
What programming languages does Amazon Q Developer support? 46
How can I provide context to Amazon Q Developer for better code generation? 46
What should I do if in-line code generation with Amazon Q Developer isn't accurate? 47
How can I use the Amazon Q Developer chat capability for code generation and troubleshooting? 47
What are some best practices for using Amazon Q Developer? 47
Can I customize Amazon Q Developer to generate recommendations based on my own code? 47
Next steps 48
Resources 49
AWS blogs 49
AWS documentation 49
AWS workshops 49
Contributors 50
Document history 51
Glossary 52
.... 52
A 53
B 56
C 58
D 61
E 65
F 67
G 69
H 70
I 71
L 74
M 75
O 79

P 82
Q 84
R 85
S 88
T 92
U 93
V 94
W 94
Z 95

Best practices with Amazon Q Developer for in-line and assistant code generation

Amazon Web Services (Contributors)

August 2024 (Document history)

Traditionally, developers have relied on their own expertise, documentation, and code snippets from various sources to write and maintain code. Although these methods have served the industry well, they can be time-consuming and prone to human errors, leading to inefficiencies and potential bugs.

This is where Amazon Q Developer steps in to improve the developer's journey. Amazon Q Developer is a powerful AWS generative AI-powered assistant designed to accelerate code development tasks by providing intelligent code generation and recommendations.

However, as with any new technology, there can be challenges. Unrealistic expectations, onboarding difficulties, troubleshooting inaccurate code generations, and properly using Amazon Q capabilities are common hurdles that developers might face. This comprehensive guide addresses these challenges, providing real-life scenarios, detailed best practices, troubleshooting, and practical real-life code examples specifically for Python and Java, two of the most widely adopted programming languages.

This guide focuses on using Amazon Q Developer to perform code development tasks such as:

- **Code completion** – Generate in-line suggestions as developers code in real time.
- **Code improvements and advice** – Discuss software development, generate new code with natural language, and improve existing code.

Objectives

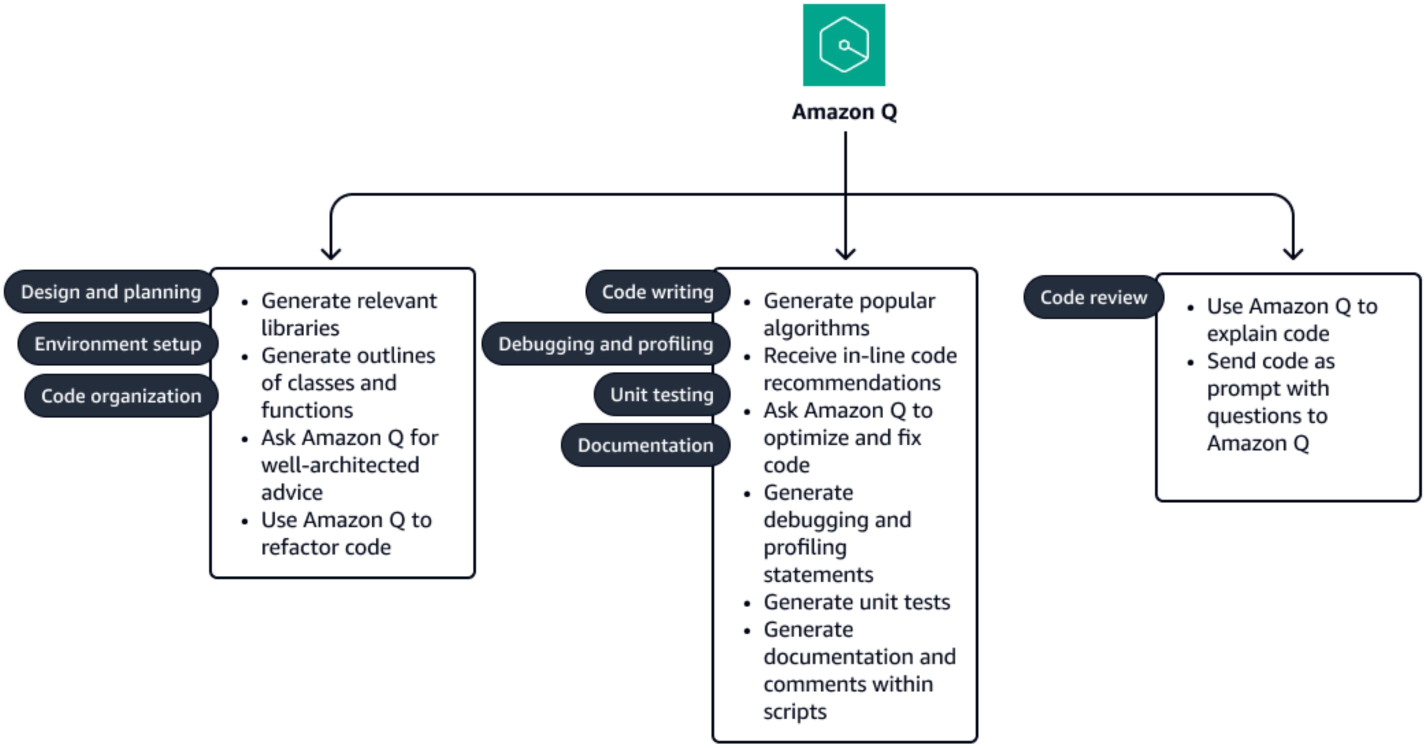
This guide's goal is to support developers who are new or continuous users of Amazon Q Developer, helping them to use the service successfully in their everyday coding tasks. Development team managers can also benefit from reading this guide.

This guide provides you with the following insights about using Amazon Q Developer:

- Understand effective use of Amazon Q Developer for code development
    - Give best practices for integrating Amazon Q Developer into a developer's workflow.
    - Offer step-by-step guidance with examples for successful code generation and recommendations.
- Mitigate common challenges and promote developer clarity for using Amazon Q Developer
    - Offer strategies and insights to meet developer expectations and overcome hurdles related to code generation accuracy and performance.
- Provide troubleshooting and error handling
    - Equip developers with Amazon Q Developer code generation troubleshooting guidance for addressing inaccurate results or unexpected behavior.
    - Provide real-life examples and scenarios specific to Python and Java.
- Optimize workflows and productivity
    - Optimize code development workflows with Amazon Q Developer.
    - Discuss strategies to enhance developer productivity.

Using Amazon Q Developer in developer workflows

Developers follow a standard workflow encompassing the stages of requirement gathering, design and planning, coding, testing, code review, and deployment. This section focuses on how you can use Amazon Q Developer capabilities to optimize key development steps.


Amazon Q
Design and planning
Environment setup
Code organization
Generate relevant libraries
Generate outlines of classes and functions
Ask Amazon Q for well-architected advice
Use Amazon Q to refactor code
Code writing
Debugging and profiling
Unit testing
Documentation
Generate popular algorithms
Receive in-line code recommendations
Ask Amazon Q to optimize and fix code
Generate debugging and profiling statements
Generate unit tests
Generate documentation and comments within scripts
Code review
Use Amazon Q to explain code
Send code as prompt with questions to Amazon Q

The previous diagram shows how Amazon Q Developer can accelerate and streamline the following common tasks in stages of code development:

- Design and planning | Environment setup | Code organization
  - Generate relevant libraries
  - Generate outlines of classes and functions
  - Ask Amazon Q for well-architected advice
  - Use Amazon Q to refactor code
- Code writing | Debugging and profiling | Unit testing | Documentation
  - Generate popular algorithms
  - Receive in-line code recommendations
  - Ask Amazon Q to optimize and fix code

  - Generate debugging and profiling statements
  - Generate unit tests
  - Generate documentation and comments within scripts
- Code review
  - Ask Amazon Q to explain code
  - Send code as prompt with questions to Amazon Q

Design and planning

After gathering business and technical requirements, developers design new, or extend existing, codebases. During this phase, Amazon Q Developer can assist developers to do the following tasks:

- Generate relevant libraries and class and function outlines for well-architected advice.
- Provide guidance for engineering, compatibility, and architectural design queries.

Coding

The coding process uses Amazon Q Developer to accelerate development in the following ways:

- **Environment setup** - Install the AWS Toolkit in your integrated development environment (IDE) (for example, VS Code or IntelliJ). Then, use Amazon Q to generate libraries or receive setup suggestions based on your project goals. For more details, see Best practices for onboarding Amazon Q Developer.
- **Code organization** - Refactor code or obtain organization recommendations from Amazon Q that align with your project objectives.
- **Code writing -** Use in-line suggestions to generate code while developing or ask Amazon Q to generate code by using the Amazon Q chat panel in your IDE. For more details, see Best practices for code generation with Amazon Q Developer.
- **Debugging and profiling -** Generate profiling commands, or use Amazon Q options like **Fix** and **Explain** to debug issues.
- **Unit testing** - Provide code as a prompt to Amazon Q during a chat session and request applicable unit test generation. For more information, see Code examples with Amazon Q Developer.

- **Documentation** - Use in-line suggestions to create comments and docstrings, or use the **Explain** option to generate detailed summaries for code selections. For more information, see Code examples with Amazon Q Developer.

Code review

Reviewers need to comprehend development code before promoting it to production. To accelerate this process, use the Amazon Q **Explain** and **Optimize** options, or send code selections with custom prompt instructions to Amazon Q in a chat session. For more information, see Chat examples.

Integration and deployment

Ask Amazon Q for guidance about continuous integration, delivery pipelines, and deployment best practices that are specific to your project's architecture.

Using these recommendations, you can learn to effectively harness Amazon Q Developer features, optimizing your workflows and increasing productivity across the entire development lifecycle.

Advanced capabilities of Amazon Q Developer

Although this guide focuses on using Amazon Q Developer in hands-on programming tasks, it's important to be aware of its following advanced capabilities:

- Amazon Q Developer code transformation
- Amazon Q Developer customizations

Amazon Q Developer code transformation

The Amazon Q Developer Agent for code transformation can upgrade the code language version of your files without the need for you to rewrite the code manually. It works by analyzing your existing code files and automatically rewriting them to use a newer version of the language. For example, Amazon Q transforms a single module if you're working in an IDE like Eclipse. If you're using Visual Studio Code, Amazon Q can transform an entire project or workspace.

Use Amazon Q when you want to perform common code upgrade tasks such as the following:

- Update code to work with the new syntax of the language version.
- Run unit tests to validate successful compilation and execution.
- Check and resolve deployment issues.

Amazon Q can save developers from days to months of tedious and repetitive work to upgrade code bases.

As of June 2024, Amazon Q Developer supports upgrading Java code and can transform Java 8 code to newer versions such as Java 11 or 17.

Amazon Q Developer customizations

With its customizations capability, Amazon Q Developer can provide in-line suggestions based on a company's own codebase. The company provides their code repository either to Amazon Simple Storage Service (Amazon S3) or through AWS CodeConnections, formerly known as AWS CodeStar Connections. Then, Amazon Q uses the custom code repository with enabled security to recommend coding patterns that are relevant to developers in that organization.

When using Amazon Q Developer customizations, be aware of the following:

- As of June 2024, the Amazon Q Developer Customizations feature is in preview mode. As a result, the feature might be limited in availability and support.
- Custom in-line code suggestions will only be accurate given the quality of the code repositories that are provided. We recommend that you review an evaluation score for each customization that you create.
- To optimize performance, we recommend that you include at least 20 data files containing the given language where all source files are greater than 10MB. Make sure that your repository consists of referable source code and not metadata files (for example, config files, property files, and readme files.)

By using Amazon Q Developer customizations, you can save time in the following ways:

- Use recommendations that are based on your own company proprietary code.
- Increase re-usability of existing code bases.
- Create repeatable patterns that are generalized across your company.

Best coding practices with Amazon Q Developer

This section discusses best practices for coding with Amazon Q Developer. The best practices include the following categories:

- Onboarding - Methods and considerations when onboarding
- Code generation - Guidance to use code generation successfully
- Code recommendations - Techniques to improve code

Best practices for onboarding Amazon Q Developer

Amazon Q Developer is a powerful generative AI coding assistant that is available through popular IDEs like Visual Studio Code and JetBrains. This section focuses on best practices for accessing and onboarding Amazon Q Developer to your coding development environment.

Prerequisites for Amazon Q Developer

Amazon Q Developer is available as part of the AWS Toolkit for Visual Studio Code and the AWS Toolkit for JetBrains (for example, IntelliJ and PyCharm). For Visual Studio Code and JetBrains IDEs, Amazon Q Developer supports Python, Java, JavaScript, TypeScript, C#, Go, Rust, PHP, Ruby, Kotlin, C, C++, Shell scripting, SQL, and Scala.

For detailed instructions on installing the AWS Toolkit for both Visual Studio Code and a JetBrains IDE, see Installing the Amazon Q Developer extension or plugin in your IDE in the *Amazon Q Developer User Guide*.

Best practices when using Amazon Q Developer

General best practices when using Amazon Q Developer include the following:

- Provide relevant context to get more accurate responses, like programming languages, frameworks, and tools that are in use. Break down complex problems into smaller components.
- Experiment and iterate on your prompts and questions. Programming often involves trying different approaches.
- Always review code suggestions before you accept them, and edit as needed to make sure that they do exactly what you intended.

- Use the customization capability to make Amazon Q Developer aware of your internal libraries, APIs, best practices, and architectural patterns for more relevant recommendations.

Data privacy and content usage in Amazon Q Developer

When deciding to use Amazon Q Developer, you should understand how your data and content are used. Following are key points:

- For Amazon Q Developer Pro users, your code content is not used for service improvement or model training.
- For Amazon Q Developer Free Tier users, you can opt out of having your content used for service improvement through IDE settings or AWS Organizations policies.
- Transmitted content is encrypted, and any stored content is secured with encryption at rest and access controls. For more information, see Data encryption in Amazon Q Developer in the *Amazon Q Developer User Guide*.

Best practices for code generation with Amazon Q Developer

Amazon Q Developer provides automatic code generation, auto-completion, and natural language code suggestions. Following are best practices for using Amazon Q Developer in-line coding assistance:

- **Provide context to help improve accuracy of responses**

  Start with existing code, import libraries, create classes and functions, or establish code skeletons. This context will help to significantly improve code generation quality.

- **Code naturally**

  Use Amazon Q Developer code generation like a robust auto-completion engine. Code as you normally do, and let Amazon Q provide suggestions as you type or pause. If code generation isn't available or you're stuck on a code issue, initiate Amazon Q by typing **Alt+C** on a PC or **Option +C** on MacOS. For more information about common actions that you can take while using in-line suggestions, see Using shortcut keys in the *Amazon Q Developer User Guide*.

- **Include import libraries that are relevant to your script's objectives**

Include relevant import libraries to help Amazon Q understand the context and generate code accordingly. You can also ask Amazon Q to suggest relevant import statements.

- **Maintain clear and focused context**

  Keep your script focused on specific objectives, and modularize distinct functionalities into separate scripts with relevant context. Avoid noisy or confusing context.

- **Experiment with prompts**

  Explore different prompts to nudge Amazon Q to produce useful results in code generation. For example, try the following approaches:

  - Use standard comment blocks for natural language prompts.
  - Create skeletons with comments to fill in classes and functions.
  - Be specific in your prompts, providing details instead of generalization.

- **Chat with Amazon Q Developer and ask for assistance**

  If Amazon Q Developerisn't providing accurate suggestions, chat with Amazon Q Developer in your IDE. It can provide code snippets or full classes and functions to kickstart your context. For more information, see Chatting with Amazon Q Developer about code in the *Amazon Q Developer User Guide.*

Best practices for code recommendations with Amazon Q Developer

Amazon Q Developer can take developer questions and evaluate code to offer recommendations which range from code generation and bug fixes to guidance using natural language. Following are best practices for using chat in Amazon Q:

- **Generate code from scratch**

  For new projects or when you need a general function (for example, copying files from Amazon S3), ask Amazon Q Developer to generate code examples using natural language prompts. Amazon Q can provide related links to public resources for further validation and investigation.

- **Seek coding knowledge and error explanations**

When faced with coding problems or error messages, provide the code block (with error message, if applicable) and your question as a prompt to Amazon Q Developer. This context will help Amazon Q provide accurate and relevant responses.

- **Improve existing code**

  To fix known errors or optimize code (for example, to reduce complexity), select the relevant code block and send it to Amazon Q Developer with your request. Be specific with your prompts for better results.
- **Explain code functionality**

  When exploring new code repositories, select a code block or an entire script and send it to Amazon Q Developer for explanation. Reduce the selection size for more specific explanations.
- **Generate unit tests**

  After sending a code block as a prompt, ask Amazon Q Developer to generate unit tests. This approach can save time and development costs related to code coverage and DevOps.
- **Find AWS answers**

  Amazon Q Developer is a valuable resource for developers working with AWS services because it contains a vast amount of knowledge related to AWS. Whether you're facing challenges with a specific AWS service, encountering error messages that are specific to AWS, or trying to learn a new AWS service, Amazon Q often provides relevant and useful information.

  Always review the recommendations that Amazon Q Developer provides to you. Then, make necessary edits and execute tests to make sure that the code meets your intended functionality.

Code examples with Amazon Q Developer

This section provides realistic examples of how to improve your experience and code generation with a focus on the Python and Java languages. Along with in-line examples, some scenarios show how you can use the chat capability of Amazon Q Developer to solicit prompts for code recommendations while you're programming.

For more information, see the following topics:

- Python examples
- Java examples
- Chat examples

Python examples

The following Python examples demonstrate how you can get started in generating in-line code suggestions with Amazon Q Developer. These illustrative examples show you how to trigger the coding assistant for your own projects. A majority of the following examples are applicable to other languages:

- Generate classes and functions
- Document code
- Generate algorithms
- Generate unit tests

Generate classes and functions

To generate full or partial class implementation, use code comments. The comment describes the intention of the class. Start simple, and add more details if necessary. After you generate a class. you can continue to generate functions within its skeleton.

When you write comments, it's preferable to use the language's comment structure. In Python, for single line comments, use **#**. For multiline comments, use `'''  '''` or `"""  """`.

Input:

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

Output:

```
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
```

Now that there is an initial class, an additional function can be generated with additional comments.

Input:

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

Output:

```
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}")
```

Document code

Reduce time in code documentation by relying on in-line documentation. Add Python comment breaks `''' '''`.

Input:

```
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}")
```

Output:

```
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}")
```

Generate algorithms

Amazon Q Developer can generate popular algorithms, especially as related to AWS SDKs. In your script, add a prompt that specifies the function you want.

This example shows how to generate a dictionary of user id keys to name values, and then generate a sorting function by using the bubble sort algorithm.

Input:

```
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
    '''
```

Output:

```
    @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)
```

In addition, you can create prompts for standard procedures. In this example, the user creates a prompt to store the dictionary to an Amazon DynamoDB table. At the beginning of the script, the user adds a prompt to add DynamoDB libraries to help provide Amazon Q Developer with additional context.

Input:

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

Output:

```
'''
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')
```

Now, the user can easily ask Amazon Q to help with storing the dictionary in DynamoDB.

Input:

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

Output:

```
@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'])
```

Generate unit tests

With in-line comments, Amazon Q Developer can generate unit tests to perform assertions and other tests. In this example, the user asks Amazon Q to generate applicable libraries for unit testing to set the context of the test script. Then, the user asks Amazon Q to write a test function for the `sort_user_id_name_map` function.

Input:

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

Output:

```
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()
```

In addition, in a chat between the user and Amazon Q Developer, it suggests and generates unit tests based on the user's code input prompts. For more information, see Chat examples.

Java examples

The following Java examples demonstrate how users can get started in generating in-line code suggestions. These illustrative examples show you how to trigger the coding assistant for your own projects. A majority of the following examples are applicable to other languages:

- Generate classes and functions
- Document code
- Generate algorithms
- Generate unit tests

Generate classes and functions

To generate full or partial class implementation, use code comments. The comment describes the intention of the class. Start simple, and add more details if necessary. After you generate a class, you can continue to generate functions within its skeleton.

When writing comments, it's preferable to use the language's comment structure. In Java, this is `/** */`.

Input:

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

Output:

```
/**
 * 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);
    }
}
```

Now that there is an initial class, an additional function can be generated with additional comments.

Input:

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

Output:

```
/**
 * 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());
    }
}
```

Document code

Reduce time in code documentation by relying on in-line documentation. Add a Java comment break `/* */`.

Input:

```
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;
        };
    }
}
```

If in-line generation doesn't occur, execute an Amazon Q Developer in-line suggestion after placing your cursor where you want documentation.

Output:

```
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;
        };
    }
}
```

For simpler explanations of code (classes, functions, or script elements), use forward characters (`//`)that are strategically placed before code sections where you want to add documentation.

Input:

```
//
```

```
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;
        };
    }
}
```

Output:

```
// 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;
        };
    }
}
```

Generate algorithms

Amazon Q Developer can generate popular algorithms, especially related to AWS SDKs. In your script, add a prompt that specifies the function you want.

This example generates a hash map of user id keys to name values and then generates a sorting function using the bubble sort algorithm.

Input:

```
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
    */


}
```

Output:

```
/*
* 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;
}
```

In addition, users can create prompts for standard procedures. In this example, the user creates a prompt to store the hash map to an Amazon DynamoDB table. At the beginning of the script, the user adds a prompt to add DynamoDB Java libraries to help provide Amazon Q Developer with additional context.

Input:

```
// 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
     */
```

Output:

```
// 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);
            }});
        }
    }
```

Generate unit tests

With in-line comments, Amazon Q Developer can generate unit tests to perform assertions and other JUnit tests. In this example, the user first asks Amazon Q to generate applicable libraries for JUnit to set the context of the test script. Next, the user creates a public class followed by a declarator and comment strings.

Input:

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

public class UnitTests {

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

Output:

```
@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);
}
```

In addition, in a chat between the user and Amazon Q Developer, it suggests and generates unit tests based on the user's code input prompts. For more information, see Chat examples.

Chat examples

Amazon Q Developer chat assists developers with a variety of tasks from coding, testing, and upgrading applications, to diagnosing errors, performing security scanning and fixes, and optimizing AWS resources. This section addresses how chatting with Amazon Q can support a developer in typical programing tasks such as the following:

- Ask about AWS services
- Generate code
- Generate unit tests
- Explain code

Ask about AWS services

Users can ask Amazon Q Developer questions about AWS services, such as finding the right service or understanding best practices.

Chat about AWS services

User:

What's the best container service to use to run my workload if I need to keep my costs low?

Amazon Q:

Based on the information provided, it seems like you are looking for a cost-effective container service to run your workloads. In that case, I would recommend considering Amazon Elastic Container Service (Amazon ECS) with Fargate. [1]

Amazon ECS with Fargate is a serverless compute engine for containers that allows you to run your containers without having to manage any underlying infrastructure. With Fargate, you don't need to provision or manage servers—Fargate takes care of that for you. This can help keep your costs low as you only pay for the compute resources your containers actually use, without having to pay for idle server capacity. [2]

Some key benefits of using Amazon ECS with Fargate:

• No need to manage servers or clusters – Fargate handles the infrastructure provisioning and management for you.

• Pay-as-you-go pricing – You only pay for the compute resources your containers use, without having to pay for idle capacity.

• Automatic scaling – Fargate can automatically scale your containers up and down based on demand.

• Tight integration with other AWS services like load balancing, logging, and monitoring.

The resources provided also mention that the ECS service offers flexibility in terms of container scheduling and placement, which can help optimize your costs further.

I'd recommend reviewing the pricing details for Amazon ECS with Fargate on the AWS website to get a better sense of the costs based on your specific workload requirements. And of course, feel free to let me know if you have any other questions!

Sources

[1] Which AWS Container Service is Best for You?

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

[2] Java container options for non-HTTP worloads

stackoverflow.com > questions > 43417016

End Amazon Q chat about AWS services

Generate code

Users can ask Amazon Q Developer for assistance to write code snippets.

Chat about writing code

User:

Can you give me an example of how to list all S3 buckets using the AWS SDK for Python (Boto)?

Amazon Q:

Certainly, here's an example of how to list all S3 buckets using the AWS SDK for Python (Boto)[1]:

```
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'])
```

Here's how the code works:

1. We first import the boto3 library, which is the SDK for Python.
2. We create an S3 client using `boto3.client('s3')`.
3. We call the `list_buckets()` method on the S3 client, which returns a response containing a list of all the buckets in the account.
4. We then iterate over the `Buckets` list in the response and print the name of each bucket.

This is a basic example, but the AWS SDK for Python (Boto) provides many more advanced features for working with Amazon S3, such as creating, deleting, and managing objects in buckets, setting bucket policies, and more. You can find more examples and documentation in the AWS SDK for Python (Boto)) documentation. [2]

Remember, when working with AWS services, it's important to follow best practices for handling credentials and sensitive information, such as storing AWS access keys in environment variables instead of hardcoding them in the code.

Sources: [1] Amazon S3 examples using SDK for Python (Boto3) - AWS SDK Code Examples docs.aws.amazon.com > code-library > latest-1x > ... > python-s3.html

[2] Get started with Amazon S3 buckets and objects using an AWS SDK - AWS SDK Code Examples docs.aws.amazon.com > code-library > latest-1x > ... > s3_example_s3_buckets_GettingStarted_sect...

End Amazon Q chat about writing code

Generate unit tests

Users can send selected code as a prompt to Amazon Q Developer, where additional instructions are applied such as generating a unit test. For step-by-step instructions, see Send code to Amazon Q in the *Amazon Q Developer User Guide*.

Chat about unit tests

User:

Write a unit test with mock data for the following function:

```
// 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;
       }
```

Amazon Q:

To write a unit test with mock data for the pollMessages function, we can use a mocking framework like Mockito. Here's an example:

```
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);
    }
}
```

In this example, we're using Mockito to mock the AmazonSQS client and set up the expected behavior for the receiveMessage method so that it returns two mock messages with bodies "message1" and "message2". Then, we call the pollMessages method and assert that the returned list of messages matches the expected values.

The key points are:

1. We use the @ExtendWith(MockitoExtension.class) annotation to enable Mockito integration with JUnit.
2. We use the @Mock annotation to create a mock instance of the AmazonSQS client.

3. We use the @InjectMocks annotation to inject the mock AmazonSQS client into the SQSUtils class.
4. In the test method, we set up the expected behavior of the receiveMessage method using Mockito.when().thenReturn().
5. We call the pollMessages method and assert that the returned list of messages matches the expected values.

This approach allows us to test the pollMessages method without actually interacting with the Amazon SQS service, which can make the test faster and more reliable.

End Amazon Q chat about unit tests

Explain code

For code review processes, developers can send a code selection to Amazon Q Developer to better understand the code. In addition, the explanation that Amazon Q provides can also serve documentation purposes. In the following example, Amazon Q explains the handler function with an AWS Lambda function, and then proposes a docstring.

Chat about explaining code

User:

Explain the following part of my code:

```
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
        }
    )
```

Amazon Q:

The provided code block appears to be a AWS Lambda function written in Python. Let me explain the key aspects of this code:

1. The function is named lambda_handler and it is the entry point for the Lambda function.
2. It uses the boto3 library to interact with AWS DynamoDB and S3 services.
3. The function retrieves an S3 object key and bucket name from the incoming event data.
4. It then calls a detect_text function (which is not shown in the provided code) to extract the full text content of the S3 object.
5. The function also retrieves metadata about the S3 object using the head_object method.
6. It then extracts the current time and calls a detect_restaurant_closure function (also not shown) to determine the reason and outcome of a restaurant closure based on the extracted text.
7. Finally, the function stores the following information in a DynamoDB table:
    - The S3 object key as the id
    - The creation time
    - The reason and outcome of the restaurant closure
    - The extracted S3 object text

Overall, this Lambda function appears to be part of a larger system that processes S3 objects, extracts text content, analyzes it for restaurant closure information, and stores the results in a

DynamoDB table. The specific details of the detect_text and detect_restaurant_closure functions would be needed to fully understand the functionality of this code.

End of Amazon Q response

User: summarize this explanation into a docstring for this lambda_handler function

Amazon Q: Here's a suggested docstring for the lambda_handler function[1]:

```
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...
```

This docstring provides a high-level summary of the function's purpose, the input parameters it expects, and the expected return value (or lack thereof, in this case). It gives the user a clear understanding of what the lambda_handler function does without delving into the implementation details.

Sources: [1] Define Lambda function handler in C# - AWS Lambda [https://docs.aws.amazon.com/lambda/latest/dg/csharp-handler.html - :~:text=The Lambda function handler is,, exits, or times out.](https://docs.aws.amazon.com/lambda/latest/dg/csharp-handler.html)

End of Amazon Q chat about explaining code

Troubleshooting code generation scenarios in Amazon Q Developer

You might encounter the following common scenarios with inaccurate code generation and resolutions when using Amazon Q Developer:

- Empty code generation
- Continuous comments
- Incorrect in-line code generation
- Inadequate results from chats

Empty code generation

While developing code, you might notice the following issues:

- Amazon Q does not provide a suggestion.
- The message "No Suggestion from Amazon Q" appears in your IDE.

Your first thought might be that Amazon Q isn't working properly. However, the root cause of these issues is typically associated with the context in the script or open project within the IDE.

If Amazon Q Developer doesn't provide a suggestion automatically, you can use the following shortcuts to manually execute Amazon Q in-line suggestions:

- PC - **Alt+C**
- MacOS - **Option+C**

For more information, see Using shortcut keys in the *Amazon Q Developer User Guide*.

In most scenarios, Amazon Q generates a suggestion. When Amazon Q returns a "No Suggestion from Amazon Q" message, review the following common resolutions for this issue:

- **Inadequate context** – Make sure that applicable libraries are present. Make sure that classes and functions defined in the script relate to new code.

- **Ambiguous prompt –** If the prompt is confusing or unclear, Amazon Q will underperform when it makes in-line code suggestions. In these scenarios, be more specific in your code and comments to elicit proper responses.

If you're stuck, continue coding to the best of your ability. Typically, Amazon Q will recognize your intention, which triggers it to offer suggestions. Another option is to send a code snippet with prompt instructions in an Amazon Q chat. For more information, see Send code to Amazon Q in the *Amazon Q Developer User Guide*.

Continuous comments

You might write a prompt using comments and only receive generic or repeated comment lines from Amazon Q.

Input:

```
// Kafka test
```

Result:

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

Typically, this issue is due to lack of context in the script or poor prompt instructions. We recommend that you use the standard format for comments given the language. In this example, the user changed the prompt to use `/*  */` for a comment block, and that change resulted in better performance from Amazon Q.

Input:

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

Result:

In this result, the user accepted several suggestions sequentially.

```
/*
 * 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());
    }
}
```

Incorrect in-line code generation

Occasionally, Amazon Q Developer in-line suggestions aren't useful or don't meet your prompt's objective. At times, in-line code suggestions might provide a generic function that's similar to a simple Hello World method.

When in-line code suggestions aren't helpful, try the following best practices:

- Review the context of the script. Usually, the more context that you provide, the better results you get from Amazon Q.

- If the context varies and/or is noisy, modularize your code. Group similar functions within the same script. Separate different classes and functions by their objective.
- Write your class and function names tactically to help Amazon Q provide better recommendations.

In this example, the developer is writing a script from scratch which uploads files to Hadoop.

Input:

```
class  MyFunction{

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

Result:

```
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));

}
```

Although Amazon Q provided a recommendation, it didn't meet the developer's expectation. To improve performance, the developer imports libraries and changes the class name to represent Hadoop utilities.

Fix:

```
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
}
```

Result:

In this result, the developer accepted several suggestions sequentially.

```
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!");
}
```

In the same HadoopUtilities script, the developer adds a comment prompt to read an Amazon Simple Queue Service (Amazon SQS) topic.

Input:

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

Result:

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

Rather that use Java methods to read an SQS message from Amazon SQS, Amazon Q suggests printing a string method. However, this suggestion doesn't satisfy the developer's original objective.

In this case, it's preferable to re-modularize the code. The script's focus is on Hadoop methods. Therefore, it might be a better approach to create a utility script that's specifically for SQS operations.

Creating a separate class for SQS utilities improves the in-line suggestion, which can be used in the HadoopUtilities script.

Input:

```
// 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
}
```

Result:

```
// 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;
```

If the in-line suggestion is still not providing expected results, ask Amazon Q Developer in a chat session to generate code.

Chat about generating code

User:

Generate a java function to read a message from an Amazon SQS queue

Amazon Q:

Here is an example Java function to read a message from an Amazon SQS queue:

```
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;
        }
    }
```

This code creates a SQSMessageReader class that takes an AmazonSQS client and a queue URL as constructor parameters. The readMessage() method uses the ReceiveMessageRequest to fetch up to 1 message from the queue, with a maximum wait time of 10 seconds.

If a message is received, the message body is printed and the message is deleted from the queue using the DeleteMessageRequest. The message body is then returned. If no message is received, null is returned.

End Amazon Q chat about generating code

Inadequate results from chats

While developing code, developers can ask Amazon Q to generate common functions, provide recommendations, or explain code. Sometimes, given a prompt (such as a question or code snippet), Amazon Q might provide either general feedback or results that don't meet your expectations. In these scenarios, try the following:

- Experiment with different prompts to improve the code generation results that you get from Amazon Q.
- Follow the URLs that Amazon Q frequently provides in its responses to your prompts or questions.
- Provide feedback to help improve future performance of Amazon Q.

For example, a developer is creating a Retrieval Augmented Generation (RAG) solution and they need to chunk their document given semantic relationships between sentences. The developer chats with Amazon Q as follows.

Chat about a RAG solution

User:

For a RAG solution, generate code for semantic chunking

Amazon Q:

Certainly, here's an example of how you could implement semantic chunking for a Retrieval Augmented Generation (RAG) solution in Python using the LangChain library:

```
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
```

In this example, we're using the TokenTextSplitter from LangChain to split the input text into chunks of 500 characters with an overlap of 50 characters. The HuggingFaceEmbeddings model is used to generate embeddings for each chunk, and the Chroma vector store is used to store the chunks and their embeddings.

The semantic_chunk function takes in the input text, splits it into chunks using the text_splitter, generates embeddings for each chunk using the embeddings model, and then adds the chunks and their embeddings to the vectorstore.

This approach allows you to split the input text into semantically meaningful chunks, which can be useful for a variety of tasks, such as question answering or summarization, in a RAG solution.

Please note that this is a basic example, and you may need to adjust the parameters (e.g., chunk_size, chunk_overlap) or the choice of text splitter and embeddings model based on your specific use case and data.

End of Amazon Q chat about a RAG solution

However, this approach only provides a basic way to chunk documents by using the sliding window technique. The developer wants an advanced way that relies on measuring embeddings. In these scenarios, the developer must add more specifics to the prompt given their objective. When the developer refers to the known AI21 text splitter function with LangChain, they receive an improved response as shown in the following chat.

Chat about a RAG solution (with improved prompt)

User:

The prompt above only relates to splitting regarding token size per chunk. I want to use AI21 Semantic text splitter. Generate a function using A121 semantic text splitting with langchain.

Amazon Q:

Okay, let's use the AI21 Semantic Text Splitter with LangChain to implement semantic chunking. Here's an example: [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
```

In this example, we're using the AI21SemanticTextSplitter from LangChain to split the input text into semantically meaningful chunks. The HuggingFaceEmbeddings model is used to create embeddings for each chunk, and the Chroma vector store is used to store the chunks and their embeddings.

The semantic_chunk function takes in the input text, splits it into chunks using the text_splitter, creates embeddings for each chunk using the embeddings model, and then adds the chunks and their embeddings to the vectorstore.

This approach allows you to split the input text into semantically meaningful chunks, which can be useful for a variety of tasks, such as question answering or summarization, in a RAG solution.

Please note that you may need to adjust the parameters or the choice of embeddings model based on your specific use case and data.

Sources [1] Learn how to use text splitters in LangChain

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

End of Amazon Q chat about a RAG solution (with improved prompt)

Based on the chat results, the developer can use the function and navigate to the source of the information.

In general, advanced topics and questions require more specific context within the prompt when you chat with Amazon Q Developer. If you believe that the results from your chat aren't accurate, use the thumbs-down icon to provide feedback about the Amazon Q response. Amazon Q Developer continuously uses feedback to improve future releases. For interactions that produced positive results, it's useful to provide your feedback by using the thumbs-up icon.

FAQs about Amazon Q Developer

This section provides answers to frequently asked questions about using Amazon Q Developer for code development.

What is Amazon Q Developer?

Amazon Q Developer is a powerful generative AI-powered service designed to accelerate code development tasks by providing intelligent code generation and recommendations. On April 30, 2024, Amazon CodeWhisperer became a part of Amazon Q Developer.

How do I access Amazon Q Developer?

Amazon Q Developer is available as part of the AWS Toolkits for Visual Studio Code and JetBrains IDEs, such as IntelliJ and PyCharm. To get started, install the latest AWS Toolkit version.

What programming languages does Amazon Q Developer support?

For Visual Studio Code and JetBrains IDEs, Amazon Q Developer supports Python, Java, JavaScript, TypeScript, C#, Go, Rust, PHP, Ruby, Kotlin, C, C++, Shell scripting, SQL, and Scala. Although this guide focuses on Python and Java for example purposes, the concepts are applicable to any supported programming language.

How can I provide context to Amazon Q Developer for better code generation?

Start with existing code, import relevant libraries, create classes and functions, or establish code skeletons. Use standard comment blocks for natural language prompts. Keep your script focused on specific objectives, and modularize distinct functionalities into separate scripts with relevant context. For more information, see Best coding practices with Amazon Q Developer.

What should I do if in-line code generation with Amazon Q Developer isn't accurate?

Review the context of the script, ensure libraries are present, and make sure that classes and functions relate to the new code. Modularize your code, and separate different classes and functions by their objective. Write clear and specific prompts or comments. If you're still uncertain about the code's accuracy and you can't proceed with it, start a chat with Amazon Q and send it the code snippet with instructions. For more information, see Troubleshooting code generation scenarios in Amazon Q Developer.

How can I use the Amazon Q Developer chat capability for code generation and troubleshooting?

Chat with Amazon Q to generate common functions, ask for recommendations, or explain code. If the initial response isn't satisfactory, experiment with different prompts and follow the provided URLs. Also, provide feedback to Amazon Q to help improve its future chat performance. Use the thumbs-up and thumbs-down icons to provide your feedback. For more information, see Chat examples.

What are some best practices for using Amazon Q Developer?

Provide relevant context, experiment, and iterate on prompts, review code suggestions before accepting them, use customization capabilities, and understand data privacy and content usage policies. For more information, see Best practices for code generation with Amazon Q Developer and Best practices for code recommendations with Amazon Q Developer.

Can I customize Amazon Q Developer to generate recommendations based on my own code?

Yes, use customizations, which is an advanced capability of Amazon Q Developer. With customizations, businesses can provide their own code repositories to enable Amazon Q Developer to recommend in-line code suggestions. For more information, see Advanced capabilities of Amazon Q Developer and Resources.

Next steps in using Amazon Q Developer

With the knowledge gained from this comprehensive guide, you can use Amazon Q Developer in your coding workflow effectively. Install the AWS Toolkit in your preferred IDE (Visual Studio Code or JetBrains), and begin exploring the generative AI-powered code generation and recommendations from Amazon Q Developer.

The most effective way to unlock the full potential of Amazon Q Developer is by hands-on experience with your own code. As you integrate Amazon Q into your development lifecycle, refer to this guide for best practices, troubleshooting, and real-world examples.

Additionally, stay informed by reviewing the AWS blogs and developer guides that are referenced in Resources. These resources provide the latest updates, best practices, and insights to help you optimize your use of Amazon Q Developer.

Your feedback is invaluable for improving this guide and helping it remain a valuable resource for developers. Share your experiences, challenges, and suggestions for future versions. Your input will help enhance the guide with additional examples, troubleshooting scenarios, and insights tailored to your needs.

Resources

AWS blogs

- Accelerate your Software Development Lifecycle with Amazon Q
- Reimagining software development with the Amazon Q Developer Agent
- Five troubleshooting examples with Amazon Q
- Customize Amazon Q Developer in your IDE with your private code base
- Three ways Amazon Q Developer agent for code transformation accelerates Java upgrades
- Leveraging Amazon Q Developer for Efficient Code Debugging and Maintenance
- Testing your applications with Amazon Q Developer

AWS documentation

- Amazon Q Developer User Guide
- Amazon Q Developer code customization
- Amazon Q Developer code transformation

AWS workshops

- Amazon Q Developer Immersion Day
- Amazon Q Developer Workshop - Building the Q-Words App
- Amazon Q Developer Workshop - Creating Effective Prompts

Contributors

The following individuals contributed to this guide:

- Joe King, Senior Data Scientist, AWS
- Prateek Gupta, Team Lead – Sr. CAA, AWS
- Manohar Reddy Arranagu, DevOps Architect, AWS
- Soumik Roy, Cloud Application Architect, AWS
- Sanket Shinde, Consultant, AWS

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Initial publication	—	August 16, 2024

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Numbers

7 Rs

Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following:

- Refactor/re-architect – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- Replatform (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- Rehost (lift and shift) – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- Relocate (hypervisor-level lift and shift) – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. You migrate servers from an on-premises platform to a cloud service for the same platform. Example: Migrate a Microsoft Hyper-V application to AWS.
- Retain (revisit) – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later time, and legacy applications that you want to retain, because there’s no business justification for migrating them.

- Retire – Decommission or remove applications that are no longer needed in your source environment.

A

ABAC

See attribute-based access control.

abstracted services

See managed services.

ACID

See atomicity, consistency, isolation, durability.

active-active migration

A database migration method in which the source and target databases are kept in sync (by using a bidirectional replication tool or dual write operations), and both databases handle transactions from connecting applications during migration. This method supports migration in small, controlled batches instead of requiring a one-time cutover. It’s more flexible but requires more work than active-passive migration.

active-passive migration

A database migration method in which in which the source and target databases are kept in sync, but only the source database handles transactions from connecting applications while data is replicated to the target database. The target database doesn’t accept any transactions during migration.

aggregate function

A SQL function that operates on a group of rows and calculates a single return value for the group. Examples of aggregate functions include `SUM` and `MAX`.

AI

See artificial intelligence.

AIOps

See artificial intelligence operations.

anonymization

The process of permanently deleting personal information in a dataset. Anonymization can help protect personal privacy. Anonymized data is no longer considered to be personal data.

anti-pattern

A frequently used solution for a recurring issue where the solution is counter-productive, ineffective, or less effective than an alternative.

application control

A security approach that allows the use of only approved applications in order to help protect a system from malware.

application portfolio

A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to the portfolio discovery and analysis process and helps identify and prioritize the applications to be migrated, modernized, and optimized.

artificial intelligence (AI)

The field of computer science that is dedicated to using computing technologies to perform cognitive functions that are typically associated with humans, such as learning, solving problems, and recognizing patterns. For more information, see What is Artificial Intelligence?

artificial intelligence operations (AIOps)

The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the operations integration guide.

asymmetric encryption

An encryption algorithm that uses a pair of keys, a public key for encryption and a private key for decryption. You can share the public key because it isn't used for decryption, but access to the private key should be highly restricted.

atomicity, consistency, isolation, durability (ACID)

A set of software properties that guarantee the data validity and operational reliability of a database, even in the case of errors, power failures, or other problems.

attribute-based access control (ABAC)

The practice of creating fine-grained permissions based on user attributes, such as department, job role, and team name. For more information, see ABAC for AWS in the AWS Identity and Access Management (IAM) documentation.

authoritative data source

A location where you store the primary version of data, which is considered to be the most reliable source of information. You can copy data from the authoritative data source to other locations for the purposes of processing or modifying the data, such as anonymizing, redacting, or pseudonymizing it.

Availability Zone

A distinct location within an AWS Region that is insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

AWS Cloud Adoption Framework (AWS CAF)

A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the AWS CAF website and the AWS CAF whitepaper.

AWS Workload Qualification Framework (AWS WQF)

A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.

B

bad bot

A bot that is intended to disrupt or cause harm to individuals or organizations.

BCP

See business continuity planning.

behavior graph

A unified, interactive view of resource behavior and interactions over time. You can use a behavior graph with Amazon Detective to examine failed logon attempts, suspicious API calls, and similar actions. For more information, see Data in a behavior graph in the Detective documentation.

big-endian system

A system that stores the most significant byte first. See also endianness.

binary classification

A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as “Is this email spam or not spam?" or "Is this product a book or a car?"

bloom filter

A probabilistic, memory-efficient data structure that is used to test whether an element is a member of a set.

blue/green deployment

A deployment strategy where you create two separate but identical environments. You run the current application version in one environment (blue) and the new application version in the other environment (green). This strategy helps you quickly roll back with minimal impact.

bot

A software application that runs automated tasks over the internet and simulates human activity or interaction. Some bots are useful or beneficial, such as web crawlers that index information on the internet. Some other bots, known as *bad bots*, are intended to disrupt or cause harm to individuals or organizations.

botnet

Networks of bots that are infected by malware and are under the control of a single party, known as a *bot herder* or *bot operator*. Botnets are the best-known mechanism to scale bots and their impact.

branch

A contained area of a code repository. The first branch created in a repository is the *main branch*. You can create a new branch from an existing branch, and you can then develop features or fix bugs in the new branch. A branch you create to build a feature is commonly referred to as a *feature branch*. When the feature is ready for release, you merge the feature branch back into the main branch. For more information, see About branches (GitHub documentation).

break-glass access

In exceptional circumstances and through an approved process, a quick means for a user to gain access to an AWS account that they don't typically have permissions to access. For more information, see the Implement break-glass procedures indicator in the AWS Well-Architected guidance.

brownfield strategy

The existing infrastructure in your environment. When adopting a brownfield strategy for a system architecture, you design the architecture around the constraints of the current systems and infrastructure. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

buffer cache

The memory area where the most frequently accessed data is stored.

business capability

What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the Organized around business capabilities section of the Running containerized microservices on AWS whitepaper.

business continuity planning (BCP)

A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.

C

CAF

See AWS Cloud Adoption Framework.

canary deployment

The slow and incremental release of a version to end users. When you are confident, you deploy the new version and replace the current version in its entirety.

CCoE

See Cloud Center of Excellence.

CDC

See change data capture.

change data capture (CDC)

The process of tracking changes to a data source, such as a database table, and recording metadata about the change. You can use CDC for various purposes, such as auditing or replicating changes in a target system to maintain synchronization.

chaos engineering

Intentionally introducing failures or disruptive events to test a system's resilience. You can use AWS Fault Injection Service (AWS FIS) to perform experiments that stress your AWS workloads and evaluate their response.

CI/CD

See continuous integration and continuous delivery.

classification

A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.

client-side encryption

Encryption of data locally, before the target AWS service receives it.

Cloud Center of Excellence (CCoE)

A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the CCoE posts on the AWS Cloud Enterprise Strategy Blog.

cloud computing

The cloud technology that is typically used for remote data storage and IoT device management. Cloud computing is commonly connected to edge computing technology.

cloud operating model

In an IT organization, the operating model that is used to build, mature, and optimize one or more cloud environments. For more information, see Building your Cloud Operating Model.

cloud stages of adoption

The four phases that organizations typically go through when they migrate to the AWS Cloud:

- Project – Running a few cloud-related projects for proof of concept and learning purposes
- Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model)
- Migration – Migrating individual applications
- Re-invention – Optimizing products and services, and innovating in the cloud

These stages were defined by Stephen Orban in the blog post The Journey Toward Cloud-First & the Stages of Adoption on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the migration readiness guide.

CMDB

See configuration management database.

code repository

A location where source code and other assets, such as documentation, samples, and scripts, are stored and updated through version control processes. Common cloud repositories include GitHub or Bitbucket Cloud. Each version of the code is called a *branch*. In a microservice structure, each repository is devoted to a single piece of functionality. A single CI/CD pipeline can use multiple repositories.

cold cache

A buffer cache that is empty, not well populated, or contains stale or irrelevant data. This affects performance because the database instance must read from the main memory or disk, which is slower than reading from the buffer cache.

cold data

Data that is rarely accessed and is typically historical. When querying this kind of data, slow queries are typically acceptable. Moving this data to lower-performing and less expensive storage tiers or classes can reduce costs.

computer vision (CV)

A field of AI that uses machine learning to analyze and extract information from visual formats such as digital images and videos. For example, Amazon SageMaker AI provides image processing algorithms for CV.

configuration drift

For a workload, a configuration change from the expected state. It might cause the workload to become noncompliant, and it's typically gradual and unintentional.

configuration management database (CMDB)

A repository that stores and manages information about a database and its IT environment, including both hardware and software components and their configurations. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.

conformance pack

A collection of AWS Config rules and remediation actions that you can assemble to customize your compliance and security checks. You can deploy a conformance pack as a single entity in an AWS account and Region, or across an organization, by using a YAML template. For more information, see Conformance packs in the AWS Config documentation.

continuous integration and continuous delivery (CI/CD)

The process of automating the source, build, test, staging, and production stages of the software release process. CI/CD is commonly described as a pipeline. CI/CD can help you automate processes, improve productivity, improve code quality, and deliver faster. For more information, see Benefits of continuous delivery. CD can also stand for *continuous deployment*. For more information, see Continuous Delivery vs. Continuous Deployment.

CV

See computer vision.

D

data at rest

Data that is stationary in your network, such as data that is in storage.

data classification

A process for identifying and categorizing the data in your network based on its criticality and sensitivity. It is a critical component of any cybersecurity risk management strategy because it helps you determine the appropriate protection and retention controls for the data. Data classification is a component of the security pillar in the AWS Well-Architected Framework. For more information, see Data classification.

data drift

A meaningful variation between the production data and the data that was used to train an ML model, or a meaningful change in the input data over time. Data drift can reduce the overall quality, accuracy, and fairness in ML model predictions.

data in transit

Data that is actively moving through your network, such as between network resources.

data mesh

An architectural framework that provides distributed, decentralized data ownership with centralized management and governance.

data minimization

The principle of collecting and processing only the data that is strictly necessary. Practicing data minimization in the AWS Cloud can reduce privacy risks, costs, and your analytics carbon footprint.

data perimeter

A set of preventive guardrails in your AWS environment that help make sure that only trusted identities are accessing trusted resources from expected networks. For more information, see Building a data perimeter on AWS.

data preprocessing

To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.

data provenance

The process of tracking the origin and history of data throughout its lifecycle, such as how the data was generated, transmitted, and stored.

data subject

An individual whose data is being collected and processed.

data warehouse

A data management system that supports business intelligence, such as analytics. Data warehouses commonly contain large amounts of historical data, and they are typically used for queries and analysis.

database definition language (DDL)

Statements or commands for creating or modifying the structure of tables and objects in a database.

database manipulation language (DML)

Statements or commands for modifying (inserting, updating, and deleting) information in a database.

DDL

See database definition language.

deep ensemble

To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.

deep learning

An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.

defense-in-depth

An information security approach in which a series of security mechanisms and controls are thoughtfully layered throughout a computer network to protect the confidentiality, integrity, and availability of the network and the data within. When you adopt this strategy on AWS, you add multiple controls at different layers of the AWS Organizations structure to help secure resources. For example, a defense-in-depth approach might combine multi-factor authentication, network segmentation, and encryption.

delegated administrator

In AWS Organizations, a compatible service can register an AWS member account to administer the organization's accounts and manage permissions for that service. This account is called the *delegated administrator* for that service. For more information and a list of compatible services, see Services that work with AWS Organizations in the AWS Organizations documentation.

deployment

The process of making an application, new features, or code fixes available in the target environment. Deployment involves implementing changes in a code base and then building and running that code base in the application's environments.

development environment

See environment.

detective control

A security control that is designed to detect, log, and alert after an event has occurred. These controls are a second line of defense, alerting you to security events that bypassed the preventative controls in place. For more information, see Detective controls in *Implementing security controls on AWS*.

development value stream mapping (DVSM)

A process used to identify and prioritize constraints that adversely affect speed and quality in a software development lifecycle. DVSM extends the value stream mapping process originally designed for lean manufacturing practices. It focuses on the steps and teams required to create and move value through the software development process.

digital twin

A virtual representation of a real-world system, such as a building, factory, industrial equipment, or production line. Digital twins support predictive maintenance, remote monitoring, and production optimization.

dimension table

In a star schema, a smaller table that contains data attributes about quantitative data in a fact table. Dimension table attributes are typically text fields or discrete numbers that behave like text. These attributes are commonly used for query constraining, filtering, and result set labeling.

disaster

An event that prevents a workload or system from fulfilling its business objectives in its primary deployed location. These events can be natural disasters, technical failures, or the result of human actions, such as unintentional misconfiguration or a malware attack.

disaster recovery (DR)

The strategy and process you use to minimize downtime and data loss caused by a disaster. For more information, see Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML

See database manipulation language.

domain-driven design

An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

DR

See disaster recovery.

drift detection

Tracking deviations from a baselined configuration. For example, you can use AWS CloudFormation to detect drift in system resources, or you can use AWS Control Tower to detect changes in your landing zone that might affect compliance with governance requirements.

DVSM

See development value stream mapping.

E

EDA

See exploratory data analysis.

EDI

See electronic data interchange.

edge computing

The technology that increases the computing power for smart devices at the edges of an IoT network. When compared with cloud computing, edge computing can reduce communication latency and improve response time.

electronic data interchange (EDI)

The automated exchange of business documents between organizations. For more information, see What is Electronic Data Interchange.

encryption

A computing process that transforms plaintext data, which is human-readable, into ciphertext.

encryption key

A cryptographic string of randomized bits that is generated by an encryption algorithm. Keys can vary in length, and each key is designed to be unpredictable and unique.

endianness

The order in which bytes are stored in computer memory. Big-endian systems store the most significant byte first. Little-endian systems store the least significant byte first.

endpoint

See service endpoint.

endpoint service

A service that you can host in a virtual private cloud (VPC) to share with other users. You can create an endpoint service with AWS PrivateLink and grant permissions to other AWS accounts or to AWS Identity and Access Management (IAM) principals. These accounts or principals can connect to your endpoint service privately by creating interface VPC endpoints. For more information, see Create an endpoint service in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

enterprise resource planning (ERP)

A system that automates and manages key business processes (such as accounting, MES, and project management) for an enterprise.

envelope encryption

The process of encrypting an encryption key with another encryption key. For more information, see Envelope encryption in the AWS Key Management Service (AWS KMS) documentation.

environment

An instance of a running application. The following are common types of environments in cloud computing:

- development environment – An instance of a running application that is available only to the core team responsible for maintaining the application. Development environments are used to test changes before promoting them to upper environments. This type of environment is sometimes referred to as a *test environment*.
- lower environments – All development environments for an application, such as those used for initial builds and tests.
- production environment – An instance of a running application that end users can access. In a CI/CD pipeline, the production environment is the last deployment environment.
- upper environments – All environments that can be accessed by users other than the core development team. This can include a production environment, preproduction environments, and environments for user acceptance testing.

epic

In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the program implementation guide.

ERP

See enterprise resource planning.

exploratory data analysis (EDA)

The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies, and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.

F

fact table

The central table in a star schema. It stores quantitative data about business operations. Typically, a fact table contains two types of columns: those that contain measures and those that contain a foreign key to a dimension table.

fail fast

A philosophy that uses frequent and incremental testing to reduce the development lifecycle. It is a critical part of an agile approach.

fault isolation boundary

In the AWS Cloud, a boundary such as an Availability Zone, AWS Region, control plane, or data plane that limits the effect of a failure and helps improve the resilience of workloads. For more information, see AWS Fault Isolation Boundaries.

feature branch

See branch.

features

The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.

feature importance

How significant a feature is for a model's predictions. This is usually expressed as a numerical score that can be calculated through various techniques, such as Shapley Additive Explanations (SHAP) and integrated gradients. For more information, see Machine learning model interpretability with AWS.

feature transformation

To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML model to benefit from the data. For example, if you break down the "2021-05-27 00:15:37" date into "2021", "May", "Thu", and "15", you can help the learning algorithm learn nuanced patterns associated with different data components.

few-shot prompting

Providing an LLM with a small number of examples that demonstrate the task and desired output before asking it to perform a similar task. This technique is an application of in-context learning, where models learn from examples (*shots*) that are embedded in prompts. Few-shot prompting can be effective for tasks that require specific formatting, reasoning, or domain knowledge. See also zero-shot prompting.

FGAC

See fine-grained access control.

fine-grained access control (FGAC)

The use of multiple conditions to allow or deny an access request.

flash-cut migration

A database migration method that uses continuous data replication through change data capture to migrate data in the shortest time possible, instead of using a phased approach. The objective is to keep downtime to a minimum.

FM

See foundation model.

foundation model (FM)

A large deep-learning neural network that has been training on massive datasets of generalized and unlabeled data. FMs are capable of performing a wide variety of general tasks, such as understanding language, generating text and images, and conversing in natural language. For more information, see What are Foundation Models.

G

generative AI

A subset of AI models that have been trained on large amounts of data and that can use a simple text prompt to create new content and artifacts, such as images, videos, text, and audio. For more information, see What is Generative AI.

geo blocking

See geographic restrictions.

geographic restrictions (geo blocking)

In Amazon CloudFront, an option to prevent users in specific countries from accessing content distributions. You can use an allow list or block list to specify approved and banned countries. For more information, see Restricting the geographic distribution of your content in the CloudFront documentation.

Gitflow workflow

An approach in which lower and upper environments use different branches in a source code repository. The Gitflow workflow is considered legacy, and the trunk-based workflow is the modern, preferred approach.

golden image

A snapshot of a system or software that is used as a template to deploy new instances of that system or software. For example, in manufacturing, a golden image can be used to provision software on multiple devices and helps improve speed, scalability, and productivity in device manufacturing operations.

greenfield strategy

The absence of existing infrastructure in a new environment. When adopting a greenfield strategy for a system architecture, you can select all new technologies without the restriction of compatibility with existing infrastructure, also known as brownfield. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

guardrail

A high-level rule that helps govern resources, policies, and compliance across organizational units (OUs). *Preventive guardrails* enforce policies to ensure alignment to compliance standards. They are implemented by using service control policies and IAM permissions boundaries. *Detective guardrails* detect policy violations and compliance issues, and generate alerts for remediation. They are implemented by using AWS Config, AWS Security Hub, Amazon GuardDuty, AWS Trusted Advisor, Amazon Inspector, and custom AWS Lambda checks.

H

HA

See high availability.

heterogeneous database migration

Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. AWS provides AWS SCT that helps with schema conversions.

high availability (HA)

The ability of a workload to operate continuously, without intervention, in the event of challenges or disasters. HA systems are designed to automatically fail over, consistently deliver high-quality performance, and handle different loads and failures with minimal performance impact.

historian modernization

An approach used to modernize and upgrade operational technology (OT) systems to better serve the needs of the manufacturing industry. A *historian* is a type of database that is used to collect and store data from various sources in a factory.

holdout data

A portion of historical, labeled data that is withheld from a dataset that is used to train a machine learning model. You can use holdout data to evaluate the model performance by comparing the model predictions against the holdout data.

homogeneous database migration

Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.

hot data

Data that is frequently accessed, such as real-time data or recent translational data. This data typically requires a high-performance storage tier or class to provide fast query responses.

hotfix

An urgent fix for a critical issue in a production environment. Due to its urgency, a hotfix is usually made outside of the typical DevOps release workflow.

hypercare period

Immediately following cutover, the period of time when a migration team manages and monitors the migrated applications in the cloud in order to address any issues. Typically, this period is 1–4 days in length. At the end of the hypercare period, the migration team typically transfers responsibility for the applications to the cloud operations team.

I

IaC

See infrastructure as code.

identity-based policy

A policy attached to one or more IAM principals that defines their permissions within the AWS Cloud environment.

idle application

An application that has an average CPU and memory usage between 5 and 20 percent over a period of 90 days. In a migration project, it is common to retire these applications or retain them on premises.

IIoT

See industrial Internet of Things.

immutable infrastructure

A model that deploys new infrastructure for production workloads instead of updating, patching, or modifying the existing infrastructure. Immutable infrastructures are inherently more consistent, reliable, and predictable than mutable infrastructure. For more information, see the Deploy using immutable infrastructure best practice in the AWS Well-Architected Framework.

inbound (ingress) VPC

In an AWS multi-account architecture, a VPC that accepts, inspects, and routes network connections from outside an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

incremental migration

A cutover strategy in which you migrate your application in small parts instead of performing a single, full cutover. For example, you might move only a few microservices or users to the new system initially. After you verify that everything is working properly, you can incrementally move additional microservices or users until you can decommission your legacy system. This strategy reduces the risks associated with large migrations.

Industry 4.0

A term that was introduced by Klaus Schwab in 2016 to refer to the modernization of manufacturing processes through advances in connectivity, real-time data, automation, analytics, and AI/ML.

infrastructure

All of the resources and assets contained within an application's environment.

infrastructure as code (IaC)

The process of provisioning and managing an application's infrastructure through a set of configuration files. IaC is designed to help you centralize infrastructure management, standardize resources, and scale quickly so that new environments are repeatable, reliable, and consistent.

industrial Internet of Things (IIoT)

The use of internet-connected sensors and devices in the industrial sectors, such as manufacturing, energy, automotive, healthcare, life sciences, and agriculture. For more information, see Building an industrial Internet of Things (IIoT) digital transformation strategy.

inspection VPC

In an AWS multi-account architecture, a centralized VPC that manages inspections of network traffic between VPCs (in the same or different AWS Regions), the internet, and on-premises networks. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

Internet of Things (IoT)

The network of connected physical objects with embedded sensors or processors that communicate with other devices and systems through the internet or over a local communication network. For more information, see What is IoT?

interpretability

A characteristic of a machine learning model that describes the degree to which a human can understand how the model's predictions depend on its inputs. For more information, see Machine learning model interpretability with AWS.

IoT

See Internet of Things.

IT information library (ITIL)

A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.

IT service management (ITSM)

Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the operations integration guide.

ITIL

See IT information library.

ITSM

See IT service management.

L

label-based access control (LBAC)

An implementation of mandatory access control (MAC) where the users and the data itself are each explicitly assigned a security label value. The intersection between the user security label and data security label determines which rows and columns can be seen by the user.

landing zone

A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see Setting up a secure and scalable multi-account AWS environment.

large language model (LLM)

A deep learning AI model that is pretrained on a vast amount of data. An LLM can perform multiple tasks, such as answering questions, summarizing documents, translating text into other languages, and completing sentences. For more information, see What are LLMs.

large migration

A migration of 300 or more servers.

LBAC

See label-based access control.

least privilege

The security best practice of granting the minimum permissions required to perform a task. For more information, see Apply least-privilege permissions in the IAM documentation.

lift and shift

See 7 Rs.

little-endian system

A system that stores the least significant byte first. See also endianness.

LLM

See large language model.

lower environments

See environment.

M

machine learning (ML)

A type of artificial intelligence that uses algorithms and techniques for pattern recognition and learning. ML analyzes and learns from recorded data, such as Internet of Things (IoT) data, to generate a statistical model based on patterns. For more information, see Machine Learning.

main branch

See branch.

malware

Software that is designed to compromise computer security or privacy. Malware might disrupt computer systems, leak sensitive information, or gain unauthorized access. Examples of malware include viruses, worms, ransomware, Trojan horses, spyware, and keyloggers.

managed services

AWS services for which AWS operates the infrastructure layer, the operating system, and platforms, and you access the endpoints to store and retrieve data. Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB are examples of managed services. These are also known as *abstracted services*.

manufacturing execution system (MES)

A software system for tracking, monitoring, documenting, and controlling production processes that convert raw materials to finished products on the shop floor.

MAP

See Migration Acceleration Program.

mechanism

A complete process in which you create a tool, drive adoption of the tool, and then inspect the results in order to make adjustments. A mechanism is a cycle that reinforces and improves itself as it operates. For more information, see Building mechanisms in the AWS Well-Architected Framework.

member account

All AWS accounts other than the management account that are part of an organization in AWS Organizations. An account can be a member of only one organization at a time.

MES

See manufacturing execution system.

Message Queuing Telemetry Transport (MQTT)

A lightweight, machine-to-machine (M2M) communication protocol, based on the publish/subscribe pattern, for resource-constrained IoT devices.

microservice

A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see Integrating microservices by using AWS serverless services.

microservices architecture

An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see Implementing microservices on AWS.

Migration Acceleration Program (MAP)

An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.

migration at scale

The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the AWS migration strategy.

migration factory

Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners, migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the discussion of migration factories and the Cloud Migration Factory guide in this content set.

migration metadata

The information about the application and server that is needed to complete the migration. Each migration pattern requires a different set of migration metadata. Examples of migration metadata include the target subnet, security group, and AWS account.

migration pattern

A repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. Example: Rehost migration to Amazon EC2 with AWS Application Migration Service.

Migration Portfolio Assessment (MPA)

An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The MPA tool (requires login) is available free of charge to all AWS consultants and APN Partner consultants.

Migration Readiness Assessment (MRA)

The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the migration readiness guide. MRA is the first phase of the AWS migration strategy.

migration strategy

The approach used to migrate a workload to the AWS Cloud. For more information, see the 7 Rs entry in this glossary and see Mobilize your organization to accelerate large-scale migrations.

ML

See machine learning.

modernization

Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see Strategy for modernizing applications in the AWS Cloud.

modernization readiness assessment

An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see Evaluating modernization readiness for applications in the AWS Cloud.

monolithic applications (monoliths)

Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see Decomposing monoliths into microservices.

MPA

See Migration Portfolio Assessment.

MQTT

See Message Queuing Telemetry Transport.

multiclass classification

A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"

mutable infrastructure

A model that updates and modifies the existing infrastructure for production workloads. For improved consistency, reliability, and predictability, the AWS Well-Architected Framework recommends the use of immutable infrastructure as a best practice.

O

OAC

See origin access control.

OAI

See origin access identity.

OCM

See organizational change management.

offline migration

A migration method in which the source workload is taken down during the migration process. This method involves extended downtime and is typically used for small, non-critical workloads.

OI

See operations integration.

OLA

See operational-level agreement.

online migration

A migration method in which the source workload is copied to the target system without being taken offline. Applications that are connected to the workload can continue to function during the migration. This method involves zero to minimal downtime and is typically used for critical production workloads.

OPC-UA

See Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

A machine-to-machine (M2M) communication protocol for industrial automation. OPC-UA provides an interoperability standard with data encryption, authentication, and authorization schemes.

operational-level agreement (OLA)

An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).

operational readiness review (ORR)

A checklist of questions and associated best practices that help you understand, evaluate, prevent, or reduce the scope of incidents and possible failures. For more information, see Operational Readiness Reviews (ORR) in the AWS Well-Architected Framework.

operational technology (OT)

Hardware and software systems that work with the physical environment to control industrial operations, equipment, and infrastructure. In manufacturing, the integration of OT and information technology (IT) systems is a key focus for Industry 4.0 transformations.

operations integration (OI)

The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the operations integration guide.

organization trail

A trail that’s created by AWS CloudTrail that logs all events for all AWS accounts in an organization in AWS Organizations. This trail is created in each AWS account that’s part of the organization and tracks the activity in each account. For more information, see Creating a trail for an organization in the CloudTrail documentation.

organizational change management (OCM)

A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the OCM guide.

origin access control (OAC)

In CloudFront, an enhanced option for restricting access to secure your Amazon Simple Storage Service (Amazon S3) content. OAC supports all S3 buckets in all AWS Regions, server-side encryption with AWS KMS (SSE-KMS), and dynamic `PUT` and `DELETE` requests to the S3 bucket.

origin access identity (OAI)

In CloudFront, an option for restricting access to secure your Amazon S3 content. When you use OAI, CloudFront creates a principal that Amazon S3 can authenticate with. Authenticated principals can access content in an S3 bucket only through a specific CloudFront distribution. See also OAC, which provides more granular and enhanced access control.

ORR

See operational readiness review.

OT

See operational technology.

outbound (egress) VPC

In an AWS multi-account architecture, a VPC that handles network connections that are initiated from within an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

P

permissions boundary

An IAM management policy that is attached to IAM principals to set the maximum permissions that the user or role can have. For more information, see Permissions boundaries in the IAM documentation.

personally identifiable information (PII)

Information that, when viewed directly or paired with other related data, can be used to reasonably infer the identity of an individual. Examples of PII include names, addresses, and contact information.

PII

See personally identifiable information.

playbook

A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.

PLC

See programmable logic controller.

PLM

See product lifecycle management.

policy

An object that can define permissions (see identity-based policy), specify access conditions (see resource-based policy), or define the maximum permissions for all accounts in an organization in AWS Organizations (see service control policy).

polyglot persistence

Independently choosing a microservice’s data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see Enabling data persistence in microservices.

portfolio assessment

A process of discovering, analyzing, and prioritizing the application portfolio in order to plan the migration. For more information, see Evaluating migration readiness.

predicate

A query condition that returns `true` or `false`, commonly located in a `WHERE` clause.

predicate pushdown

A database query optimization technique that filters the data in the query before transfer. This reduces the amount of data that must be retrieved and processed from the relational database, and it improves query performance.

preventative control

A security control that is designed to prevent an event from occurring. These controls are a first line of defense to help prevent unauthorized access or unwanted changes to your network. For more information, see Preventative controls in *Implementing security controls on AWS*.

principal

An entity in AWS that can perform actions and access resources. This entity is typically a root user for an AWS account, an IAM role, or a user. For more information, see *Principal* in Roles terms and concepts in the IAM documentation.

privacy by design

A system engineering approach that takes privacy into account through the whole development process.

private hosted zones

A container that holds information about how you want Amazon Route 53 to respond to DNS queries for a domain and its subdomains within one or more VPCs. For more information, see Working with private hosted zones in the Route 53 documentation.

proactive control

A security control designed to prevent the deployment of noncompliant resources. These controls scan resources before they are provisioned. If the resource is not compliant with the control, then it isn't provisioned. For more information, see the Controls reference guide in the

AWS Control Tower documentation and see Proactive controls in *Implementing security controls on AWS*.

product lifecycle management (PLM)

The management of data and processes for a product throughout its entire lifecycle, from design, development, and launch, through growth and maturity, to decline and removal.

production environment

See environment.

programmable logic controller (PLC)

In manufacturing, a highly reliable, adaptable computer that monitors machines and automates manufacturing processes.

prompt chaining

Using the output of one LLM prompt as the input for the next prompt to generate better responses. This technique is used to break down a complex task into subtasks, or to iteratively refine or expand a preliminary response. It helps improve the accuracy and relevance of a model's responses and allows for more granular, personalized results.

pseudonymization

The process of replacing personal identifiers in a dataset with placeholder values. Pseudonymization can help protect personal privacy. Pseudonymized data is still considered to be personal data.

publish/subscribe (pub/sub)

A pattern that enables asynchronous communications among microservices to improve scalability and responsiveness. For example, in a microservices-based MES, a microservice can publish event messages to a channel that other microservices can subscribe to. The system can add new microservices without changing the publishing service.

Q

query plan

A series of steps, like instructions, that are used to access the data in a SQL relational database system.

query plan regression

When a database service optimizer chooses a less optimal plan than it did before a given change to the database environment. This can be caused by changes to statistics, constraints, environment settings, query parameter bindings, and updates to the database engine.

R

RACI matrix

See responsible, accountable, consulted, informed (RACI).

RAG

See Retrieval Augmented Generation.

ransomware

A malicious software that is designed to block access to a computer system or data until a payment is made.

RASCI matrix

See responsible, accountable, consulted, informed (RACI).

RCAC

See row and column access control.

read replica

A copy of a database that’s used for read-only purposes. You can route queries to the read replica to reduce the load on your primary database.

re-architect

See 7 Rs.

recovery point objective (RPO)

The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

recovery time objective (RTO)

The maximum acceptable delay between the interruption of service and restoration of service.

refactor

See 7 Rs.

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see Specify which AWS Regions your account can use.

regression

An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).

rehost

See 7 Rs.

release

In a deployment process, the act of promoting changes to a production environment.

relocate

See 7 Rs.

replatform

See 7 Rs.

repurchase

See 7 Rs.

resiliency

An application's ability to resist or recover from disruptions. High availability and disaster recovery are common considerations when planning for resiliency in the AWS Cloud. For more information, see AWS Cloud Resilience.

resource-based policy

A policy attached to a resource, such as an Amazon S3 bucket, an endpoint, or an encryption key. This type of policy specifies which principals are allowed access, supported actions, and any other conditions that must be met.

responsible, accountable, consulted, informed (RACI) matrix

A matrix that defines the roles and responsibilities for all parties involved in migration activities and cloud operations. The matrix name is derived from the responsibility types defined in the matrix: responsible (R), accountable (A), consulted (C), and informed (I). The support (S) type is optional. If you include support, the matrix is called a *RASCI matrix*, and if you exclude it, it's called a *RACI matrix*.

responsive control

A security control that is designed to drive remediation of adverse events or deviations from your security baseline. For more information, see Responsive controls in *Implementing security controls on AWS*.

retain

See 7 Rs.

retire

See 7 Rs.

Retrieval Augmented Generation (RAG)

A generative AI technology in which an LLM references an authoritative data source that is outside of its training data sources before generating a response. For example, a RAG model might perform a semantic search of an organization's knowledge base or custom data. For more information, see What is RAG.

rotation

The process of periodically updating a secret to make it more difficult for an attacker to access the credentials.

row and column access control (RCAC)

The use of basic, flexible SQL expressions that have defined access rules. RCAC consists of row permissions and column masks.

RPO

See recovery point objective.

RTO

See recovery time objective.

runbook

A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.

S

SAML 2.0

An open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API operations without you having to create user in IAM for everyone in your organization. For more information about SAML 2.0-based federation, see About SAML 2.0-based federation in the IAM documentation.

SCADA

See supervisory control and data acquisition.

SCP

See service control policy.

secret

In AWS Secrets Manager, confidential or restricted information, such as a password or user credentials, that you store in encrypted form. It consists of the secret value and its metadata. The secret value can be binary, a single string, or multiple strings. For more information, see What's in a Secrets Manager secret? in the Secrets Manager documentation.

security by design

A system engineering approach that takes security into account through the whole development process.

security control

A technical or administrative guardrail that prevents, detects, or reduces the ability of a threat actor to exploit a security vulnerability. There are four primary types of security controls: preventative, detective, responsive, and proactive.

security hardening

The process of reducing the attack surface to make it more resistant to attacks. This can include actions such as removing resources that are no longer needed, implementing the security best practice of granting least privilege, or deactivating unnecessary features in configuration files.

security information and event management (SIEM) system

Tools and services that combine security information management (SIM) and security event management (SEM) systems. A SIEM system collects, monitors, and analyzes data from servers, networks, devices, and other sources to detect threats and security breaches, and to generate alerts.

security response automation

A predefined and programmed action that is designed to automatically respond to or remediate a security event. These automations serve as detective or responsive security controls that help you implement AWS security best practices. Examples of automated response actions include modifying a VPC security group, patching an Amazon EC2 instance, or rotating credentials.

server-side encryption

Encryption of data at its destination, by the AWS service that receives it.

service control policy (SCP)

A policy that provides centralized control over permissions for all accounts in an organization in AWS Organizations. SCPs define guardrails or set limits on actions that an administrator can delegate to users or roles. You can use SCPs as allow lists or deny lists, to specify which services or actions are permitted or prohibited. For more information, see Service control policies in the AWS Organizations documentation.

service endpoint

The URL of the entry point for an AWS service. You can use the endpoint to connect programmatically to the target service. For more information, see AWS service endpoints in *AWS General Reference*.

service-level agreement (SLA)

An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

service-level indicator (SLI)

A measurement of a performance aspect of a service, such as its error rate, availability, or throughput.

service-level objective (SLO)

A target metric that represents the health of a service, as measured by a service-level indicator.

shared responsibility model

A model describing the responsibility you share with AWS for cloud security and compliance. AWS is responsible for security *of* the cloud, whereas you are responsible for security *in* the cloud. For more information, see Shared responsibility model.

SIEM

See security information and event management system.

single point of failure (SPOF)

A failure in a single, critical component of an application that can disrupt the system.

SLA

See service-level agreement.

SLI

See service-level indicator.

SLO

See service-level objective.

split-and-seed model

A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization’s capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.

SPOF

See single point of failure.

star schema

A database organizational structure that uses one large fact table to store transactional or measured data and uses one or more smaller dimensional tables to store data attributes. This structure is designed for use in a data warehouse or for business intelligence purposes.

strangler fig pattern

An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was introduced by Martin Fowler as a way to manage risk when rewriting monolithic systems. For an example of how to apply this pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

subnet

A range of IP addresses in your VPC. A subnet must reside in a single Availability Zone.

supervisory control and data acquisition (SCADA)

In manufacturing, a system that uses hardware and software to monitor physical assets and production operations.

symmetric encryption

An encryption algorithm that uses the same key to encrypt and decrypt the data.

synthetic testing

Testing a system in a way that simulates user interactions to detect potential issues or to monitor performance. You can use Amazon CloudWatch Synthetics to create these tests.

system prompt

A technique for providing context, instructions, or guidelines to an LLM to direct its behavior. System prompts help set context and establish rules for interactions with users.

T

tags

Key-value pairs that act as metadata for organizing your AWS resources. Tags can help you manage, identify, organize, search for, and filter resources. For more information, see Tagging your AWS resources.

target variable

The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.

task list

A tool that is used to track progress through a runbook. A task list contains an overview of the runbook and a list of general tasks to be completed. For each general task, it includes the estimated amount of time required, the owner, and the progress.

test environment

See environment.

training

To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don't know the target.

transit gateway

A network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see What is a transit gateway in the AWS Transit Gateway documentation.

trunk-based workflow

An approach in which developers build and test features locally in a feature branch and then merge those changes into the main branch. The main branch is then built to the development, preproduction, and production environments, sequentially.

trusted access

Granting permissions to a service that you specify to perform tasks in your organization in AWS Organizations and in its accounts on your behalf. The trusted service creates a service-linked role in each account, when that role is needed, to perform management tasks for you. For more information, see Using AWS Organizations with other AWS services in the AWS Organizations documentation.

tuning

To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.

two-pizza team

A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development.

U

uncertainty

A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data. For more information, see the Quantifying uncertainty in deep learning systems guide.

undifferentiated tasks

Also known as *heavy lifting*, work that is necessary to create and operate an application but that doesn’t provide direct value to the end user or provide competitive advantage. Examples of undifferentiated tasks include procurement, maintenance, and capacity planning.

upper environments

See environment.

V

vacuuming

A database maintenance operation that involves cleaning up after incremental updates to reclaim storage and improve performance.

version control

Processes and tools that track changes, such as changes to source code in a repository.

VPC peering

A connection between two VPCs that allows you to route traffic by using private IP addresses. For more information, see What is VPC peering in the Amazon VPC documentation.

vulnerability

A software or hardware flaw that compromises the security of the system.

W

warm cache

A buffer cache that contains current, relevant data that is frequently accessed. The database instance can read from the buffer cache, which is faster than reading from the main memory or disk.

warm data

Data that is infrequently accessed. When querying this kind of data, moderately slow queries are typically acceptable.

window function

A SQL function that performs a calculation on a group of rows that relate in some way to the current record. Window functions are useful for processing tasks, such as calculating a moving average or accessing the value of rows based on the relative position of the current row.

workload

A collection of resources and code that delivers business value, such as a customer-facing application or backend process.

workstream

Functional groups in a migration project that are responsible for a specific set of tasks. Each workstream is independent but supports the other workstreams in the project. For example, the portfolio workstream is responsible for prioritizing applications, wave planning, and collecting migration metadata. The portfolio workstream delivers these assets to the migration workstream, which then migrates the servers and applications.

WORM

See write once, read many.

WQF

See AWS Workload Qualification Framework.

write once, read many (WORM)

A storage model that writes data a single time and prevents the data from being deleted or modified. Authorized users can read the data as many times as needed, but they cannot change it. This data storage infrastructure is considered immutable.

Z

zero-day exploit

An attack, typically malware, that takes advantage of a zero-day vulnerability.

zero-day vulnerability

An unmitigated flaw or vulnerability in a production system. Threat actors can use this type of vulnerability to attack the system. Developers frequently become aware of the vulnerability as a result of the attack.

zero-shot prompting

Providing an LLM with instructions for performing a task but no examples (*shots*) that can help guide it. The LLM must use its pre-trained knowledge to handle the task. The effectiveness of zero-shot prompting depends on the complexity of the task and the quality of the prompt. See also few-shot prompting.

zombie application

An application that has an average CPU and memory usage below 5 percent. In a migration project, it is common to retire these applications.