The AWS CDK layer guide

AWS Prescriptive Guidance

AWS Prescriptive Guidance: The AWS CDK layer guide

Table of Contents

The AWS CDK layer guide

Steven Guggenheimer, Amazon Web Services (AWS)

December 2023 (document history)

One of the main concepts behind the AWS Cloud Development Kit (AWS CDK) is a lot like the concept behind staying warm on a cold day. That concept is called *layering*. On a cold day you put on a shirt, a jacket, and sometimes an even bigger jacket depending on how cold it is. Then if you go inside and the heater is blazing, you can take off one or both jacket layers so you're not too hot. The AWS CDK uses layering to provide different levels of abstraction for using cloud components. Layering ensures that you never have to write too much code or have too little access to resource properties when you deploy your infrastructure as code (IAC) stacks.

If you don't use the AWS CDK, you have to write your AWS CloudFormation templates by hand; that is, you're leveraging only a single layer that forces you to write far more code than is usually necessary. On the other hand, if the AWS CDK were to abstract away everything in CloudFormation that you usually don't need to write out, you wouldn't be able to handle any edge cases.

To address this issue, the AWS CDK splits resource provisioning into three separate and distinct layers:

- **Layer 1** – *The CloudFormation layer*: The most basic layer where the CloudFormation resource and the AWS CDK resource are nearly identical.
- **Layer 2** – *The curated layer*: The layer where CloudFormation resources are abstracted into programmatic classes that streamline much of the boilerplate CloudFormation syntax under the hood. This layer makes up most of the AWS CDK.
- **Layer 3** – *The pattern layer*: The most abstracted layer where you can use the building blocks provided by layers 1 and 2 to customize the code for your specific use case.

Each item from each layer is an instance of a special AWS CDK class called a `Construct`. According to AWS documentation, constructs are "the basic building blocks of AWS CDK apps. A construct represents a 'cloud component' and encapsulates everything AWS CloudFormation needs to create the component." The constructs within these layers are known as *L1*, *L2*, and *L3 constructs* depending on which layer they belong to. In this guide we'll take a tour through each AWS CDK layer to find out what they're used for and why they matter.

This guide is intended for technical managers, leads, and developers who are interested in digging deeper into the core concepts that make the AWS CDK work. The AWS CDK is a popular tool, but it's very common for teams to miss out on a large portion of what it has to offer. When you begin to understand the concepts described in this guide, you can unlock a whole new world of possibilities and optimize your teams' resource provisioning processes.

In this guide:

Layer 1 constructs

L1 constructs are the building blocks of the AWS CDK and are easily distinguished from other constructs by the prefix `Cfn`. For example, the Amazon DynamoDB package in the AWS CDK contains a `Table` construct, which is an L2 construct. The corresponding L1 construct is called `CfnTable`, and it directly represents a CloudFormation DynamoDB `Table`. It's impossible to use the AWS CDK without accessing this first layer, although an AWS CDK application typically never uses an L1 construct directly. However, in the majority of cases, the L2 and L3 constructs that developers are accustomed to using rely heavily on L1 constructs. So you can think of L1 constructs as the bridge between CloudFormation and the AWS CDK.

The sole purpose of the AWS CDK is to generate CloudFormation templates by using standard coding languages. After you run the **cdk synth** CLI command and the resulting CloudFormation templates are generated, the AWS CDK's job is complete. The **cdk deploy** command is there just as a convenience, but what you're doing when you run that command happens entirely within CloudFormation. The piece of the puzzle that translates AWS CDK code into the format that CloudFormation understands is the L1 construct.

The AWS CDK–CloudFormation lifecycle for L1 constructs

The process for creating and using L1 constructs consists of these steps:

1. The AWS CDK build process converts CloudFormation specifications into programmatic code in the form of L1 constructs.
2. Developers write code that either directly or indirectly references L1 constructs as part of an AWS CDK application.
3. Developers run the **cdk synth** command to convert programmatic code back into the format dictated by the CloudFormation specifications (templates).
4. Developers run the **cdk deploy** command to deploy the CloudFormation stacks within these templates into AWS account environments.

Let's do a little exercise. Go to the AWS CDK open source repository on GitHub, pick a random AWS service, and then go to the AWS CDK package for that service (located in the folder `packages`, `aws-cdk-lib`, `aws-<servicename>`, `lib`). For this example let's pick Amazon S3, but this works for any service. If you look at the main index.ts file for that package, you'll see a line that reads:

```
export * from './s3.generated';
```

However, you won't see the `s3.generated` file anywhere in the corresponding directory. This is because L1 constructs are auto-generated from the CloudFormation resource specification during the AWS CDK build process. So you'll see `s3.generated` in the package only after you run the AWS CDK build command for the package.

The AWS CloudFormation resource specification

The AWS CloudFormation resource specification defines infrastructure as code (IAC) for AWS and determines how code within CloudFormation templates is converted into resources in an AWS account. This specification defines AWS resources in JSON format on a per-Region level. Each resource is given a unique resource type name that follows the format `provider::service::resource`. For example, the resource type name for an Amazon S3 bucket would be `AWS::S3::Bucket`, and the resource type name for an Amazon S3 access point would be `AWS::S3::AccessPoint`. These resource types can be rendered in a CloudFormation template by using the syntax defined in the AWS CloudFormation resource specification. When the AWS CDK build process runs, each resource type also becomes an L1 construct.

Consequently, each L1 construct is a programmatic mirror image of its corresponding CloudFormation resource. Every property that you would apply in a CloudFormation template is available when you instantiate an L1 construct, and every required CloudFormation property is also required as an argument when you instantiate the corresponding L1 construct. The following table compares an S3 bucket as represented in a CloudFormation template with the same S3 bucket as defined as an AWS CDK L1 construct.

CloudFormation template

```
"myS3Bucket": {
    "Type": "AWS::S3::Bucket",
    "Properties": {
      "BucketName": "my-s3-bucket",
      "BucketEncryption": {
        "ServerSideEncryptionConfig
uration": [
          {
            "ServerSideEncrypt
ionByDefault": {
```

L1 construct

```
new CfnBucket(this, "myS3Bucket", {
  bucketName: "my-s3-bucket",
  bucketEncryption: {
    serverSideEncryptionConfigu
ration: [
      {
        serverSideEncryptionByDefau
lt: {
          sseAlgorithm: "AES256"
        }
```

```
              "SSEAlgorithm": "AES256"
            }
          }
        ]
      },
      "MetricsConfigurations": [
        {
          "Id": "myConfig"
        }
      ],
      "OwnershipControls": {
        "Rules": [
          {
            "ObjectOwnership":
 "BucketOwnerPreferred"
          }
        ]
      },
      "PublicAccessBlockConfigura
tion": {
        "BlockPublicAcls": true,
        "BlockPublicPolicy": true,
        "IgnorePublicAcls": true,
        "RestrictPublicBuckets": true
      },
      "VersioningConfiguration": {
        "Status": "Enabled"
      }
    }
  }
```

```
      }
    ]
  },
  metricsConfigurations: [
    {
      id: "myConfig"
    }
  ],
  ownershipControls: {
    rules: [
      {
        objectOwnership: "BucketOw
nerPreferred"
      }
    ]
  },
  publicAccessBlockConfiguration: {
    blockPublicAcls: true,
    blockPublicPolicy: true,
    ignorePublicAcls: true,
    restrictPublicBuckets: true
  },
  versioningConfiguration: {
    status: "Enabled"
  }
});
```

As you can see, the L1 construct is the exact manifestation in code of the CloudFormation resource. There are no shortcuts or simplifications, so the amount of boilerplate text that must be written is roughly the same. However, one of the great advantages to using the AWS CDK is supposed to be that it helps eliminate a lot of that boilerplate CloudFormation syntax. So how does that happen? That's where the L2 construct comes in.

Layer 2 constructs

The AWS CDK open source repository is written primarily by using the TypeScript programming language and consists of numerous packages and modules. The main package library, called `aws-cdk-lib`, is roughly divided into one package per AWS service, although this is not always the case. As previously discussed, the L1 constructs are automatically generated during the build process, so what is all that code you see when you look inside the repository? Those are L2 constructs, which are abstractions of L1 constructs.

The packages also contain a collection of TypeScript types, enums, and interfaces as well as helper classes that add more functionality, but those items all serve L2 constructs. All L2 constructs call their corresponding L1 constructs in their constructors upon instantiation, and the resulting L1 construct that is created can be accessed from layer 2 like this:

```
const role = new Bucket(this, "my-bucket", {/*...BucketProps*/});
const cfnBucket = role.node.defaultChild;
```

The L2 construct takes the default properties, convenience methods, and other syntactic sugar and applies them to the L1 construct. This takes away much of the repetition and verbosity that is necessary to provision resources directly in CloudFormation.

All L2 constructs build their corresponding L1 constructs under the hood. However, L2 constructs don't actually extend L1 constructs. Both L1 and L2 constructs inherit a special class called Construct. In version 1 of the AWS CDK the `Construct` class was built into the development kit, but in version 2 it's a separate standalone package. This is so other packages such as the Cloud Development Kit for Terraform (CDKTF) can include it as a dependency. Any class that inherits the `Construct` class is an L1, L2, or an L3 construct. L2 constructs extend this class directly whereas L1 constructs extend a class called `CfnResource`, as shown in the following table.

L1 inheritance tree	**L2 inheritance tree**
L1 construct	*L2 construct*
# class CfnResource	*# class Construct*
## abstract class CfnRefElement	
### abstract class CfnElement	

class Construct

If both L1 and L2 constructs inherit the `Construct` class, why don't L2 constructs just extend L1? Well, the classes between the `Construct` class and layer 1 lock the L1 construct in place as a mirror image of the CloudFormation resource. They contain abstract methods (methods that downstream classes **must** include) like `_toCloudFormation`, which forces the construct to directly output CloudFormation syntax. L2 constructs skip over those classes and extend the `Construct` class directly. This gives them the flexibility to abstract much of the code needed for L1 constructs by building them separately within their constructors.

The previous section featured a side-by-side comparison of an S3 bucket from a CloudFormation template and that same S3 bucket rendered as an L1 construct. That comparison showed that the properties and syntax are nearly identical, and the L1 construct saves only three or four lines compared with the CloudFormation construct. Now let's compare the L1 construct with the L2 construct for the same S3 bucket:

L1 construct for S3 bucket

```
new CfnBucket(this, "myS3Bucket", {
  bucketName: "my-s3-bucket",
  bucketEncryption: {
    serverSideEncryptionConfigu
ration: [
      {
        serverSideEncryptionByDefau
lt: {
          sseAlgorithm: "AES256"
        }
      }
    ]
  },
  metricsConfigurations: [
    {
      id: "myConfig"
    }
  ],
  ownershipControls: {
    rules: [
      {
```

L2 construct for S3 bucket

```
new Bucket(this, "myS3Bucket", {
  bucketName: "my-s3-bucket",
  encryption: BucketEncryption.S
3_MANAGED,
  metrics: [
    {
      id: "myConfig"
    },
  ],
  objectOwnership: ObjectOwnership.BU
CKET_OWNER_PREFERRED,
  blockPublicAccess: BlockPubl
icAccess.BLOCK_ALL,
  versioned: true
});
```

```
        objectOwnership: "BucketOw
nerPreferred"
      }
    ]
  },
  publicAccessBlockConfiguration: {
    blockPublicAcls: true,
    blockPublicPolicy: true,
    ignorePublicAcls: true,
    restrictPublicBuckets: true
  },
  versioningConfiguration: {
    status: "Enabled"
  }
});
```

As you can see, the L2 construct is less than half the size of the L1 construct. L2 constructs use numerous techniques to accomplish this consolidation. Some of these techniques apply to a single L2 construct, but others can be reused across multiple constructs so they are separated into their own class for reusability. L2 constructs consolidate CloudFormation syntax in several ways, as discussed in the following sections.

Default properties

The simplest way to consolidate the code for provisioning a resource is to turn the most common property settings into defaults. The AWS CDK has access to powerful programming languages and CloudFormation doesn't, so these defaults are often conditional in nature. Sometimes several lines of CloudFormation configuration can be eliminated from the AWS CDK code because those settings can be inferred from the values of other properties that are passed to the construct.

Structs, types, and interfaces

Although the AWS CDK is available in several programming languages, it is written natively in TypeScript, so that language's type system is used to define the types that make up L2 constructs. Diving deep into that type system is beyond the scope of this guide; see the TypeScript documentation for details. To summarize, a TypeScript `type` describes what kind of data a particular variable holds. This could be basic data such as a `string`, or more complex data such as

an `object`. A TypeScript `interface` is another way of expressing the TypeScript object type, and a `struct` is another name for an interface.

TypeScript doesn't use the term *struct*, but if you look in the AWS CDK API Reference, you'll see that a struct is actually just another TypeScript interface within the code. The API Reference also refers to certain interfaces as interfaces, too. If structs and interfaces are the same thing, why does the AWS CDK documentation make a distinction between them?

What the AWS CDK refers to as *structs* are interfaces that represent any object used by an L2 construct. This includes the object types for the property arguments that are passed to the L2 construct during instantiation, such as `BucketProps` for the S3 Bucket construct and `TableProps` for the DynamoDB Table construct, as well as other TypeScript interfaces that are used within the AWS CDK. In short, if it's a TypeScript interface within the AWS CDK and its name isn't prefixed by the letter `I`, the AWS CDK calls it a *struct*.

Conversely, the AWS CDK uses the term *interface* to represent the base elements a plain object would need to be considered a proper representation of a particular construct or helper class. That is, an interface describes what an L2 construct's public properties must be. All AWS CDK interface names are the names of existing constructs or helper classes prefixed by the letter `I`. All L2 constructs extend the `Construct` class, but they also implement their corresponding interface. So the L2 construct `Bucket` implements the `IBucket` interface.

Static methods

Every instance of an L2 construct is also an instance of its corresponding interface, but the reverse isn't true. This is important when looking through a struct to see which data types are required. If a struct has a property called `bucket`, which requires the data type `IBucket`, you could pass either an object that contains the properties listed in the `IBucket` interface or an instance of an L2 `Bucket`. Either one would work. However, if that `bucket` property called for an L2 `Bucket`, you could pass only a `Bucket` instance in that field.

This distinction becomes very important when you import pre-existing resources into your stack. You can create an L2 construct for any resource that's native to your stack, but if you need to reference a resource that was created outside the stack, you have to use that L2 construct's interface. That's because creating an L2 construct creates a new resource if one doesn't already exist within that stack. References to existing resources must be plain objects that conform to that L2 construct's interface.

To make this easier in practice, most L2 constructs have a set of static methods associated with them that return that L2 construct's interface. These static methods usually start with the word `from`. The first two arguments passed to these methods are the same `scope` and `id` arguments required for a standard L2 construct. However, the third argument isn't `props` but a small subset of properties (or sometimes just one property) that defines an interface. For this reason, when you pass an L2 construct, in most cases only the elements of the interface are required. This is so you can use imported resources as well, where possible.

```
// Example of referencing an external S3 bucket
const preExistingBucket = Bucket.fromBucketName(this, "external-bucket", "name-of-
bucket-that-already-exists");
```

However, you shouldn't rely heavily on interfaces. You should import resources and use interfaces directly only when absolutely necessary, because interfaces don't provide many of the properties—such as helper methods—that make an L2 construct so powerful.

Helper methods

An L2 construct is a programmatic class rather than a simple object, so it can expose class methods that allow you to manipulate your resource configuration after instantiation has taken place. A good example of this is the AWS Identity and Access Management (IAM) L2 Role construct. The following snippets show two ways to create the same IAM role by using the L2 `Role` construct.

Without a helper method:

```
const role = new Role(this, "my-iam-role", {
   assumedBy: new FederatedPrincipal('my-identity-provider.com'),
   managedPolicies: [
      ManagedPolicy.fromAwsManagedPolicyName("ReadOnlyAccess")
   ],
   inlinePolicies: {
      lambdaPolicy: new PolicyDocument({
         statements: [
            new PolicyStatement({
               effect: Effect.ALLOW,
               actions: [ 'lambda:UpdateFunctionCode' ],
               resources: [ 'arn:aws:lambda:us-east-1:123456789012:function:my-
function' ]
            })
```

```
         ]
      })
   }
});
```

With a helper method:

```
const role = new Role(this, "my-iam-role", {
   assumedBy: new FederatedPrincipal('my-identity-provider.com')
});

role.addManagedPolicy(ManagedPolicy.fromAwsManagedPolicyName("ReadOnlyAccess"));
role.attachInlinePolicy(new Policy(this, "lambda-policy", {
   policyName: "lambdaPolicy",
   statements: [
      new PolicyStatement({
         effect: Effect.ALLOW,
         actions: [ 'lambda:UpdateFunctionCode' ],
         resources: [ 'arn:aws:lambda:us-east-1:123456789012:function:my-function' ]
      })
   ]
 }));
```

The ability to use instance methods to manipulate resource configuration after instantiation gives L2 constructs a lot of additional flexibility over the previous layer. L1 constructs also inherit some resource methods (such as `addPropertyOverride`), but it's not until layer two that you get methods that are specifically designed for that resource and its properties.

Enums

CloudFormation syntax often requires you to specify many details in order to provision a resource properly. However, the majority of use cases are often covered by only a handful of configurations. Representing those configurations by using a series of enumerated values can vastly reduce the amount of code needed.

For example, in the S3 bucket L2 code example from earlier in this section, you have to use the CloudFormation template's `bucketEncryption` property to provide all the details, including the name of the encryption algorithm to be used. Instead, the AWS CDK provides the `BucketEncryption` enum, which takes the five most common forms of bucket encryption and lets you express each by using single variable names.

What about the edge cases that aren't covered by the enums? One of the goals of an L2 construct is to simplify the task of provisioning a layer 1 resource, so certain edge cases that are less commonly used might not be supported in layer 2. To support these edge cases, the AWS CDK lets you manipulate the underlying CloudFormation resource properties directly by using the addPropertyOverride method. For more about property overrides, see the Best practices section of this guide and the section Abstractions and escape hatches in the AWS CDK documentation.

Helper classes

Sometimes an enum can't accomplish the programmatic logic needed to configure a resource for a given use case. In these situations, the AWS CDK often offers a helper class instead. An enum is a simple object that offers a series of key-value pairs, whereas a helper class offers the full capabilities of a TypeScript class. A helper class can still act like an enum by exposing static properties, but those properties could then have their values internally set with conditional logic in the helper class constructor or in a helper method.

So although the `BucketEncryption` enum can reduce the amount of code needed to set an encryption algorithm on an S3 bucket, that same strategy would not work for setting time durations because there are simply too many possible values to choose from. Creating an enum for each value would be far more trouble than it's worth. For this reason, a helper class is used for an S3 bucket's default S3 Object Lock configuration settings, as represented by the ObjectLockRetention class. `ObjectLockRetention` contains two static methods: one for compliance retention and the other for governance retention. Both methods take an instance of the Duration helper class as an argument to express the amount of time that the lock should be configured for.

Another example is the AWS Lambda helper class Runtime. At first glance, it might appear that the static properties associated with this class could be handled by an enum. However, under the hood, each property value represents an instance of the `Runtime` class itself, so the logic performed in the class's constructor could not be achieved within an enum.

Layer 3 constructs

If L1 constructs perform a literal translation of CloudFormation resources into programmatic code, and L2 constructs replace much of the verbose CloudFormation syntax with helper methods and custom logic, what do the L3 constructs do? The answer to that is limited only by your imagination. You can create layer 3 to fit any specific use case. If your project needs a resource that has a specific subset of properties, you can create a reusable L3 construct to meet that need.

L3 constructs are called *patterns* within the AWS CDK. A pattern is any object that extends the `Construct` class in the AWS CDK (or extends a class that extends the `Construct` class) to perform any abstracted logic beyond layer 2. When you use the AWS CDK CLI to run **cdk init** to start a new AWS CDK project, you must choose from three AWS CDK application types: `app`, `lib`, and `sample-app`.

```
Available templates:
* app: Template for a CDK Application
   └ cdk init app --language=[csharp|fsharp|go|java|javascript|python|typescript]
* lib: Template for a CDK Construct Library
   └ cdk init lib --language=typescript
* sample-app: Example CDK Application with some constructs
   └ cdk init sample-app --language=[csharp|fsharp|go|java|javascript|python|typescript]
```

`app` and `sample-app` both represent classic AWS CDK applications where you build and deploy CloudFormation stacks to AWS environments. When you choose `lib`, you're choosing to build a brand new L3 construct. `app` and `sample-app` allow you to pick any language that the AWS CDK supports, but you can only pick TypeScript with `lib`. This is because the AWS CDK is natively written in TypeScript and uses an open source system called JSii to translate the original code into the other supported languages. When you choose `lib` to initiate your project, you're choosing to build an extension to the AWS CDK.

Any class that extends the `Construct` class can be an L3 construct, but the most common use cases for layer 3 are resource interactions, resource extensions, and custom resources. Most L3 constructs use one or more of these three cases in order to extend AWS CDK functionality.

Resource interactions

A solution typically employs several AWS services that work together. For example, an Amazon CloudFront distribution often uses an S3 bucket as its origin and AWS WAF for protection against

common exploits. AWS AppSync and Amazon API Gateway often use Amazon DynamoDB tables as data sources for their APIs. A pipeline in AWS CodePipeline often uses Amazon S3 as its source and AWS CodeBuild for its build stages. In these cases it's often useful to create a single L3 construct that handles the provisioning of two or more interconnected L2 constructs.

Here's an example of an L3 construct that provisions a CloudFront distribution along with its S3 origin, an AWS WAF to put in front of it, an Amazon Route 53 record, and an AWS Certificate Manager (ACM) certificate to add a custom endpoint with encryption in transit—all in one reusable construct:

```
// Define the properties passed to the L3 construct
export interface CloudFrontWebsiteProps {
     distributionProps: DistributionProps
     bucketProps: BucketProps
     wafProps: CfnWebAclProps
     zone: IHostedZone
}

// Define the L3 construct
export class CloudFrontWebsite extends Construct {
     public distribution: Distribution

     constructor(
        scope: Construct,
        id: string,
        props: CloudFrontWebsiteProps
     ) {
         super(scope, id);

         const certificate = new Certificate(this, "Certificate", {
            domainName: props.zone.zoneName,
            validation: CertificateValidation.fromDns(props.zone)
         });
         const defaultBehavior = {
            origin: new S3Origin(new Bucket(this, "bucket", props.bucketProps))
         }
         const waf = new CfnWebACL(this, "waf", props.wafProps);
         this.distribution = new Distribution(this, id, {
            ...props.distributionProps,
            defaultBehavior,
            certificate,
            domainNames: [this.domainName],
            webAclId: waf.attrArn,
```

```
            });
        }
}
```

Notice that CloudFront, Amazon S3, Route 53, and ACM all use L2 constructs, but the web ACL (which defines rules for handling web requests) uses an L1 construct. This is because the AWS CDK is an evolving open source package that isn't fully complete, and there is no L2 construct for `WebAcl` yet. However, anyone can contribute to the AWS CDK by creating new L2 constructs. So until the AWS CDK offers an L2 construct for `WebAcl`, you have to use an L1 construct. To create a new website by using the L3 construct `CloudFrontWebsite`, you use the following code:

```
const siteADotCom = new CloudFrontWebsite(stack, "siteA", siteAProps);
const siteBDotCom = new CloudFrontWebsite(stack, "siteB", siteBProps);
const siteCDotCom = new CloudFrontWebsite(stack, "siteC", siteCProps);
```

In this example, the CloudFront `Distribution` L2 construct is exposed as a public property of the L3 construct. There will still be cases where you need to expose L3 properties such as this, as necessary. In fact we're going to see `Distribution` again later, in the Custom resources section.

The AWS CDK includes a few examples of resource interaction patterns such as this one. In addition to the `aws-ecs` package that contains the L2 constructs for Amazon Elastic Container Service (Amazon ECS), the AWS CDK has a package called aws-ecs-patterns. This package contains several L3 constructs that combine Amazon ECS with Application Load Balancers, Network Load Balancers, and target groups while offering different versions that are preset for Amazon Elastic Compute Cloud (Amazon EC2) and AWS Fargate. Because many serverless applications use Amazon ECS only with Fargate, these L3 constructs provide a convenience that can save developers time and customers money.

Resource extensions

Some use cases require resources to have specific default settings that are not native to the L2 construct. At the stack level, this can be handled by using aspects, but another convenient way to give an L2 construct new defaults is by extending layer 2. Because a construct is any class that inherits the `Construct` class, and L2 constructs extend that class, you can also create an L3 construct by directly extending an L2 construct.

This can be especially useful for custom business logic that supports a customer's singular needs. Let's suppose a company has a repository that stores all of its AWS Lambda function code in a

single directory called `src/lambda` and that most Lambda functions reuse the same runtime and handler name each time. Instead of configuring the code path every time you configure a new Lambda function, you could create a new L3 construct:

```
export class MyCompanyLambdaFunction extends Function {
      constructor(
        scope: Construct,
        id: string,
        props: Partial<FunctionProps> = {}
     ) {
         super(scope, id, {
            handler: 'index.handler',
            runtime: Runtime.NODEJS_LATEST,
            code: Code.fromAsset(`src/lambda/${props.functionName || id}`),
            ...props
     });
}
```

You could then replace the L2 `Function` construct everywhere in the repository as follows:

```
new MyCompanyLambdaFunction(this, "MyFunction");
new MyCompanyLambdaFunction(this, "MyOtherFunction");
new MyCompanyLambdaFunction(this, "MyThirdFunction", {
   runtime: Runtime.PYTHON_3_11
});
```

The defaults allow you to create new Lambda functions on a single line, and the L3 construct is set up so you can still override the default properties if needed.

Extending L2 constructs directly works best when you just want to add new defaults to existing L2 constructs. If you need other custom logic as well, it's better to extend the `Construct` class. The reason for this stems from the `super` method, which is called within the constructor. In classes that extend other classes, the `super` method is used to call the parent class's constructor, and this **must** be the first thing that happens within your constructor. This means that any manipulation of passed arguments or other custom logic can happen only **after** the original L2 construct has been created. If you need to perform any of this custom logic before you instantiate your L2 construct, it's better to follow the pattern outlined previously in the Resource interactions section.

Custom resources

Custom resources are a powerful feature in CloudFormation that let you run custom logic from a Lambda function that's activated during stack deployment. Whenever you need any processes during deployment that aren't directly supported by CloudFormation, you can use a custom resource to make it happen. The AWS CDK offers classes that allow you to create custom resources programmatically as well. By using custom resources within an L3 constructor, you can make a construct out of almost anything.

One of the advantages of using Amazon CloudFront is its strong global caching capabilities. If you want to manually reset that cache so that your website immediately reflects new changes made to your origin, you can use a CloudFront invalidation. However, invalidations are processes that run on a CloudFront distribution instead of being properties of a CloudFront distribution. They can be created and applied to an existing distribution at any time, so they aren't natively part of the provisioning and deployment process.

In this scenario, you might want to create and run an invalidation after every update to a distribution's origin. Because of custom resources, you can create an L3 construct that looks something like this:

```
export interface CloudFrontInvalidationProps {
     distribution: Distribution
     region?: string
     paths?: string[]
}

export class CloudFrontInvalidation extends Construct {
      constructor(
        scope: Construct,
        id: string,
        props: CloudFrontInvalidationProps
     ) {
          super(scope, id);
          const policy = AwsCustomResourcePolicy.fromSdkCalls({
               resources:AwsCustomResourcePolicy.ANY_RESOURCE
          });
          new AwsCustomResource(scope, `${id}Invalidation`, {
             policy,
             onUpdate: {
                  service: 'CloudFront',
                  action: 'createInvalidation',
```

```
                region: props.region || 'us-east-1',
                physicalResourceId:
 PhysicalResourceId.fromResponse('Invalidation.Id'),
                parameters: {
                   DistributionId: props.distribution.distributionId,
                   InvalidationBatch: {
                       Paths: {
                          Quantity: props.paths?.length || 1,
                          Items: props.paths || ['/*']
                       },
                       CallerReference: crypto.randomBytes(5).toString('hex')
                   }
                }
            }
         }
     }
}
```

Using the distribution we created earlier in the `CloudFrontWebsite` L3 construct, you could do this very easily:

```
new CloudFrontInvalidation(this, 'MyInvalidation', {
       distribution: siteADotCom.distribution
});
```

This L3 construct uses an AWS CDK L3 construct called AwsCustomResource to create a custom resource that performs custom logic. `AwsCustomResource` is very convenient when you need to make exactly one AWS SDK call, because it allows you to do that without having to write any Lambda code. If you have more complex requirements and want to implement your own logic, you can use the basic CustomResource class directly.

Another good example of the AWS CDK using a custom resource L3 construct is S3 bucket deployment. The Lambda function created by the custom resource within the constructor of this L3 construct adds functionality that CloudFormation wouldn't be able to handle otherwise: it adds and updates objects in an S3 bucket. Without S3 bucket deployment, you wouldn't be able to put content into the S3 bucket you just created as part of your stack, which would be very inconvenient.

The best example of the AWS CDK eliminating the need to write out reams of CloudFormation syntax is this basic `S3BucketDeployment`:

```
new BucketDeployment(this, 'BucketObjects', {
  sources: [Source.asset('./path-to-my-bucket-content')],
  destinationBucket: myS3Bucket
});
```

Compare that with the CloudFormation code that you'd have to write to accomplish the same thing:

```
"lambdapolicyA5E98E09": {
   "Type": "AWS::IAM::Policy",
   "Properties": {
    "PolicyDocument": {
     "Statement": [
      {
       "Action": "lambda:UpdateFunctionCode",
       "Effect": "Allow",
       "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function"
      }
     ],
     "Version": "2012-10-17"
    },
    "PolicyName": "lambdaPolicy",
    "Roles": [
     {
      "Ref": "myiamroleF09C7974"
     }
    ]
   },
   "Metadata": {
    "aws:cdk:path": "CdkScratchStack/lambda-policy/Resource"
   }
  },
  "BucketObjectsAwsCliLayer8C081206": {
   "Type": "AWS::Lambda::LayerVersion",
   "Properties": {
    "Content": {
     "S3Bucket": {
      "Fn::Sub": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}"
     },
     "S3Key": "e2277687077a2abf9ae1af1cc9565e6715e2ebb62f79ec53aa75a1af9298f642.zip"
    },
    "Description": "/opt/awscli/aws"
   },
```

```
   "Metadata": {
    "aws:cdk:path": "CdkScratchStack/BucketObjects/AwsCliLayer/Resource",
    "aws:asset:path":
"asset.e2277687077a2abf9ae1af1cc9565e6715e2ebb62f79ec53aa75a1af9298f642.zip",
    "aws:asset:is-bundled": false,
    "aws:asset:property": "Content"
   }
  },
  "BucketObjectsCustomResourceB12E6837": {
   "Type": "Custom::CDKBucketDeployment",
   "Properties": {
    "ServiceToken": {
     "Fn::GetAtt": [
      "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756C81C01536",
      "Arn"
     ]
    },
    "SourceBucketNames": [
     {
      "Fn::Sub": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}"
     }
    ],
    "SourceObjectKeys": [
     "f888a9d977f0b5bdbc04a1f8f07520ede6e00d4051b9a6a250860a1700924f26.zip"
    ],
    "DestinationBucketName": {
     "Ref": "myS3Bucket77F80CC0"
    },
    "Prune": true
   },
   "UpdateReplacePolicy": "Delete",
   "DeletionPolicy": "Delete",
   "Metadata": {
    "aws:cdk:path": "CdkScratchStack/BucketObjects/CustomResource/Default"
   }
  },
  "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRole89A01265": {
   "Type": "AWS::IAM::Role",
   "Properties": {
    "AssumeRolePolicyDocument": {
     "Statement": [
      {
       "Action": "sts:AssumeRole",
       "Effect": "Allow",
```

```
       "Principal": {
        "Service": "lambda.amazonaws.com"
       }
      }
     ],
     "Version": "2012-10-17"
    },
    "ManagedPolicyArns": [
     {
      "Fn::Join": [
       "",
       [
        "arn:",
        {
         "Ref": "AWS::Partition"
        },
        ":iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
       ]
      ]
     }
    ]
   },
   "Metadata": {
    "aws:cdk:path": "CdkScratchStack/
Custom::CDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756C/ServiceRole/Resource"
   }
  },

 "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRoleDefaultPolicy88902FDF":
 {
   "Type": "AWS::IAM::Policy",
   "Properties": {
    "PolicyDocument": {
     "Statement": [
      {
       "Action": [
        "s3:GetBucket*",
        "s3:GetObject*",
        "s3:List*"
       ],
       "Effect": "Allow",
       "Resource": [
        {
         "Fn::Join": [
```

```
          "",
          [
           "arn:",
           {
            "Ref": "AWS::Partition"
           },
           ":s3:::",
           {
            "Fn::Sub": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}"
           },
           "/*"
          ]
         ]
        },
        {
         "Fn::Join": [
          "",
          [
           "arn:",
           {
            "Ref": "AWS::Partition"
           },
           ":s3:::",
           {
            "Fn::Sub": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}"
           }
          ]
         ]
        }
       ]
      },
      {
       "Action": [
        "s3:Abort*",
        "s3:DeleteObject*",
        "s3:GetBucket*",
        "s3:GetObject*",
        "s3:List*",
        "s3:PutObject",
        "s3:PutObjectLegalHold",
        "s3:PutObjectRetention",
        "s3:PutObjectTagging",
        "s3:PutObjectVersionTagging"
       ],
```

```
       "Effect": "Allow",
       "Resource": [
        {
         "Fn::GetAtt": [
          "myS3Bucket77F80CC0",
          "Arn"
         ]
        },
        {
         "Fn::Join": [
          "",
          [
           {
            "Fn::GetAtt": [
             "myS3Bucket77F80CC0",
             "Arn"
            ]
           },
           "/*"
          ]
         ]
        }
       ]
      }
     ],
     "Version": "2012-10-17"
    },
    "PolicyName":
 "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRoleDefaultPolicy88902FDF",
    "Roles": [
     {
      "Ref":
 "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRole89A01265"
     }
    ]
   },
   "Metadata": {
    "aws:cdk:path": "CdkScratchStack/
Custom::CDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756C/ServiceRole/DefaultPolicy/
Resource"
   }
  },
  "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756C81C01536": {
   "Type": "AWS::Lambda::Function",
```

```
   "Properties": {
    "Code": {
     "S3Bucket": {
      "Fn::Sub": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}"
     },
     "S3Key": "9eb41a5505d37607ac419321497a4f8c21cf0ee1f9b4a6b29aa04301aea5c7fd.zip"
    },
    "Role": {
     "Fn::GetAtt": [
      "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRole89A01265",
      "Arn"
     ]
    },
    "Environment": {
     "Variables": {
      "AWS_CA_BUNDLE": "/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem"
     }
    },
    "Handler": "index.handler",
    "Layers": [
     {
      "Ref": "BucketObjectsAwsCliLayer8C081206"
     }
    ],
    "Runtime": "python3.9",
    "Timeout": 900
   },
   "DependsOn": [

 "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRoleDefaultPolicy88902FDF",
    "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRole89A01265"
   ],
   "Metadata": {
    "aws:cdk:path": "CdkScratchStack/
Custom::CDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756C/Resource",
    "aws:asset:path":
 "asset.9eb41a5505d37607ac419321497a4f8c21cf0ee1f9b4a6b29aa04301aea5c7fd",
    "aws:asset:is-bundled": false,
    "aws:asset:property": "Code"
   }
  }
```

4 lines versus 241 lines is a **huge** difference! And this is just one example of what's possible when you leverage layer 3 to customize your stacks.

Best practices

L1 constructs

- You can't always avoid using L1 constructs directly, but you should avoid it whenever possible. If a specific L2 construct doesn't support your edge case, you can explore these two options instead of using the L1 construct directly:
  - **Access** `defaultChild`: If the CloudFormation property you need isn't available in an L2 construct, you can access the underlying L1 construct by using `L2Construct.node.defaultChild`. You can update any public properties of the L1 construct by accessing them through this property instead of going through the trouble of creating the L1 construct yourself.
  - **Use property overrides**: What if the property that you want to update isn't public? The ultimate escape hatch that allows the AWS CDK to do anything that a CloudFormation template can do is to use a method that's available in every L1 construct: addPropertyOverride. You can manipulate your stack at the CloudFormation template level by passing the CloudFormation property name and value directly to this method.

L2 constructs

- Remember to take advantage of the helper methods that L2 constructs often offer. With layer 2, you don't have to pass every property upon instantiation. L2 helper methods can make resource provisioning exponentially more convenient, especially when conditional logic is needed. One of the most convenient helper methods is derived from the Grant class. This class isn't used directly, but many L2 constructs use it to provide helper methods that make permissions much easier to implement. For example, if you want to give permission to an L2 Lambda function to access an L2 S3 bucket, you could call `s3Bucket.grantReadWrite(lambdaFunction)` instead of creating a new role and policy.

L3 constructs

- Although L3 constructs can be very convenient when you want to make your stacks more reusable and customizable, we recommend that you use them carefully. Consider which type of L3 construct you need or whether you need an L3 construct at all:
  - If you aren't interacting with AWS resources directly, it's often more appropriate to make a helper class instead of extending the `Construct` class. This is because the `Construct` class

performs many actions by default that are needed only if you're directly interacting with AWS resources. So if you don't need those actions performed, it's more efficient to avoid them.

- If you determine that creating a new L3 construct is appropriate, in most cases you will want to extend the `Construct`class directly. Extend other L2 constructs only when you want to update the default properties of the construct. If other L2 constructs or custom logic are involved, extend `Construct` directly and instantiate all resources within the constructor.

FAQ

Can't I use the AWS CDK without understanding layers?

You absolutely can. But as with most powerful tools, the AWS CDK becomes more powerful the more you know about it. Learning how the AWS CDK's layers interact unlocks a new level of understanding that helps simplify your stack deployments far beyond what you can do with just basic AWS CDK knowledge.

Can I make L2 constructs from L1 in the same way that I make L3 constructs from L2?

If a resource already has an L2 construct, we recommend that you use that construct and make your customizations in layer 3. This is because much research has already gone into figuring out the best ways to configure existing L2 constructs for a particular resource. However, there are several L1 constructs whose L2 constructs don't exist yet. In those cases, we encourage you to create your own L2 constructs and share them with others by becoming a contributor to the AWS CDK open source library. You can find everything you need to get started in the contribution guidelines for the AWS CDK.

Which AWS resources don't yet have official L2 constructs?

The number of AWS resources that don't have L2 constructs is getting lower by the day, but if you're interested in helping create an L2 construct for one of these resources, visit the AWS CDK API Reference. Look at the list of resources in the left pane. The resources that have the superscript 1 next to their names don't have official L2 constructs.

Can I make an L2 or L3 construct in any language that the AWS CDK supports?

The AWS CDK supports several programming languages, including TypeScript, JavaScript, Python, Java, C#, and Go. You can create your personal L3 constructs by using the AWS CDK code compiled into the relevant language. However, if you want to contribute to the AWS CDK or create native AWS CDK constructs, you must use TypeScript. This is because TypeScript is the only language that

is native to the AWS CDK. The AWS CDK versions for other languages are built from the native TypeScript code by using an AWS library called JSii.

Where can I find existing L3 constructs outside of the AWS CDK?

There are too many locations to share here, but you can find many of the most popular constructs on the AWS Solutions Constructs website and in the AWS CDK section of the Construct Hub.

Resources

- AWS CDK API Reference
- AWS CloudFormation resource specification
- AWS CDK constructs documentation
- AWS CDK abstractions and escape hatches
- Leverage L2 constructs to reduce the complexity of your AWS CDK application (AWS blog post)
- AWS CloudFormation custom resources
- AWS Solutions Constructs
- Construct Hub
- AWS CDK Examples (GitHub repository)

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	—	December 4, 2023

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 AWS CodeCommit. 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, AWS Panorama offers devices that add CV to on-premises camera networks, and Amazon SageMaker 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.

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.

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.

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.

G

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.

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.

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 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.

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

An approach in system engineering that takes privacy into account throughout the whole engineering 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.

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).

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.

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 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.

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.

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.