La guida ai AWS CDK livelli

AWS Linee guida prescrittive

AWS Linee guida prescrittive: La guida ai AWS CDK livelli

Table of Contents

La guida ai AWS CDK livelli

Steven Guggenheimer, Amazon Web Services ()AWS

Dicembre 2023 (cronologia dei documenti)

Uno dei concetti principali alla base di AWS Cloud Development Kit (AWS CDK) è molto simile al concetto alla base del stare al caldo in una giornata fredda. Questo concetto si chiama stratificazione. In una giornata fredda indossi una camicia, una giacca e talvolta una giacca ancora più grande a seconda di quanto fa freddo. Quindi, se entri e la stufa è accesa, puoi toglierti uno o entrambi gli strati della giacca in modo da non avere troppo caldo. AWS CDK Utilizza la stratificazione per fornire diversi livelli di astrazione per l'utilizzo dei componenti cloud. La stratificazione assicura che non sia mai necessario scrivere troppo codice o avere troppo poco accesso alle proprietà delle risorse quando si implementano gli stack Infrastructure as Code (IAC).

Se non utilizzate il AWS CDK, dovete scrivere i AWS CloudFormationmodelli a mano; in altre parole, state sfruttando un solo livello che vi obbliga a scrivere molto più codice di quanto sia normalmente necessario. D'altra parte, se AWS CDK dovessero astrarre tutto ciò CloudFormation che di solito non è necessario scrivere, non saresti in grado di gestire casi limite.

Per risolvere questo problema, AWS CDK suddivide il provisioning delle risorse in tre livelli separati e distinti:

- Livello 1: il CloudFormation livello: il livello più elementare in cui la CloudFormation risorsa e la AWS CDK risorsa sono quasi identiche.
- Livello 2: il livello curato: il livello in cui CloudFormation le risorse vengono astratte in classi programmatiche che semplificano gran parte della sintassi standard CloudFormation . Questo livello costituisce la maggior parte di. AWS CDK
- Livello 3: Il livello del pattern: Il livello più astratto in cui è possibile utilizzare i blocchi costitutivi forniti dai livelli 1 e 2 per personalizzare il codice per il caso d'uso specifico.

Ogni elemento di ogni livello è un'istanza di una AWS CDK classe speciale chiamata a`Construct`. Secondo AWS la documentazione, i costrutti sono «gli elementi costitutivi di base delle AWS CDK app. Un costrutto rappresenta un «componente cloud» e racchiude tutto ciò che AWS CloudFormation serve per creare il componente». I costrutti all'interno di questi livelli sono noti come costrutti L1, L2 e L3 a seconda del livello a cui appartengono. In questa guida esamineremo ogni AWS CDK livello per scoprire a cosa servono e perché sono importanti.

Questa guida è destinata ai responsabili tecnici, ai lead e agli sviluppatori interessati ad approfondire i concetti fondamentali alla base del AWS CDK lavoro. AWS CDK È uno strumento popolare, ma è molto comune che i team si perdano gran parte di ciò che ha da offrire. Quando inizi a comprendere i concetti descritti in questa guida, puoi sbloccare un mondo completamente nuovo di possibilità e ottimizzare i processi di approvvigionamento delle risorse dei tuoi team.

In questa guida:

Costrutti di livello 1

I costrutti L1 sono gli elementi costitutivi di AWS CDK e si distinguono facilmente dagli altri costrutti grazie al prefisso. `Cfn` Ad esempio, il pacchetto Amazon DynamoDB contiene `Table` un costrutto, che AWS CDK è un costrutto L2. Viene chiamato `CfnTable` il costrutto L1 corrispondente che rappresenta direttamente un DynamoDB CloudFormation . `Table` È impossibile utilizzarlo AWS CDK senza accedere a questo primo livello, sebbene un' AWS CDK applicazione in genere non utilizzi mai direttamente un costrutto L1. Tuttavia, nella maggior parte dei casi, i costrutti L2 e L3 che gli sviluppatori sono abituati a utilizzare si basano in larga misura sui costrutti L1. Quindi puoi pensare ai costrutti L1 come al ponte tra e. CloudFormation AWS CDK

L'unico scopo AWS CDK è generare CloudFormation modelli utilizzando linguaggi di codifica standard. Dopo aver eseguito il comando CDK synth CLI e aver generato i CloudFormation modelli risultanti, il lavoro AWS CDKè completo. Il comando cdk deploy è disponibile solo per comodità, ma ciò che fai quando esegui quel comando avviene interamente all'interno. CloudFormation Il pezzo del puzzle che traduce il AWS CDK codice nel formato CloudFormation comprensibile è il costrutto L1.

Il ciclo di vita AWS CDK— per i costrutti L1 CloudFormation

Il processo di creazione e utilizzo dei costrutti L1 consiste nei seguenti passaggi:

1. Il processo di AWS CDK compilazione converte CloudFormation le specifiche in codice programmatico sotto forma di costrutti L1.
2. Gli sviluppatori scrivono codice che fa riferimento direttamente o indirettamente ai costrutti L1 come parte di un'applicazione. AWS CDK
3. Gli sviluppatori eseguono il comando cdk synth per riconvertire il codice programmatico nel formato dettato dalle specifiche (modelli). CloudFormation
4. Gli sviluppatori eseguono il comando cdk deploy per distribuire gli CloudFormation stack all'interno di questi modelli negli ambienti di account. AWS

Facciamo un piccolo esercizio. Vai al repository AWS CDK open source attivo GitHub, scegli un AWS servizio a caso, quindi vai al AWS CDK pacchetto relativo a quel servizio (che si trova nella cartella`packages`,, `aws-cdk-libaws-<servicename>`,`lib`). Per questo esempio scegliamo Amazon S3, ma funziona per qualsiasi servizio. Se guardi il file index.ts principale di quel pacchetto, vedrai una riga che dice:

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

Tuttavia, non vedrai il `s3.generated` file da nessuna parte nella directory corrispondente. Questo perché i costrutti L1 vengono generati automaticamente dalle specifiche CloudFormation delle risorse durante il AWS CDK processo di compilazione. Quindi lo vedrete `s3.generated` nel pacchetto solo dopo aver eseguito il comando AWS CDK build per il pacchetto.

Le specifiche AWS CloudFormation delle risorse

La specifica AWS CloudFormation delle risorse definisce l'infrastruttura come codice (IAC) AWS e determina in che modo il codice contenuto nei CloudFormation modelli viene convertito in risorse in un AWS account. Questa specifica definisce AWS le risorse in formato JSON a livello di regione. A ogni risorsa viene assegnato un nome di tipo di risorsa univoco che segue il formato. `provider::service::resource` Ad esempio, il nome del tipo di risorsa per un bucket Amazon S3 sarebbe `AWS::S3::Bucket` e il nome del tipo di risorsa per un punto di accesso Amazon S3 sarebbe. `AWS::S3::AccessPoint` Questi tipi di risorse possono essere renderizzati in un CloudFormation modello utilizzando la sintassi definita nella specifica della risorsa. AWS CloudFormation Quando viene eseguito il processo di AWS CDK compilazione, ogni tipo di risorsa diventa anche un costrutto L1.

Di conseguenza, ogni costrutto L1 è un'immagine speculare programmatica della risorsa corrispondente. CloudFormation Ogni proprietà da applicare in un CloudFormation modello è disponibile quando si crea un'istanza di un costrutto L1 e ogni CloudFormation proprietà richiesta è richiesta anche come argomento quando si crea un'istanza del costrutto L1 corrispondente. La tabella seguente confronta un bucket S3 rappresentato in un CloudFormation modello con lo stesso bucket S3 definito come costrutto L1. AWS CDK

CloudFormation modello

```
"amzns3demobucket": {
    "Type": "AWS::S3::Bucket",
    "Properties": {
      "BucketName": "amzn-s3-demo-
bucket",
      "BucketEncryption": {
        "ServerSideEncryptionConfig
uration": [
          {
```

costrutto L1

```
new CfnBucket(this, "amzns3de
mobucket", {
  bucketName: "amzn-s3-demo-bucket",
  bucketEncryption: {
    serverSideEncryptionConfigu
ration: [
      {
        serverSideEncryptionByDefau
lt: {
```

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

```
          sseAlgorithm: "AES256"
        }
      }
    ]
  },
  metricsConfigurations: [
    {
      id: "myConfig"
    }
  ],
  ownershipControls: {
    rules: [
      {
        objectOwnership: "BucketOw
nerPreferred"
      }
    ]
  },
  publicAccessBlockConfiguration: {
    blockPublicAcls: true,
    blockPublicPolicy: true,
    ignorePublicAcls: true,
    restrictPublicBuckets: true
  },
  versioningConfiguration: {
    status: "Enabled"
  }
});
```

Come puoi vedere, il costrutto L1 è l'esatta manifestazione in codice della risorsa. CloudFormation Non esistono scorciatoie o semplificazioni, quindi la quantità di testo standard da scrivere è all'incirca la stessa. Tuttavia, si suppone che uno dei grandi vantaggi dell'utilizzo di AWS CDK sia che aiuta a eliminare gran parte di quella sintassi standard. CloudFormation Allora come succede? È qui che entra in gioco il costrutto L2.

Costrutti di livello 2

Il repository AWS CDK open source è scritto principalmente utilizzando il linguaggio di TypeScriptprogrammazione ed è composto da numerosi pacchetti e moduli. La libreria di pacchetti principale, chiamataaws-cdk-lib, è suddivisa approssimativamente in un pacchetto per AWS servizio, sebbene non sia sempre così. Come discusso in precedenza, i costrutti L1 vengono generati automaticamente durante il processo di compilazione, quindi cos'è tutto quel codice che vedi quando guardi all'interno del repository? Questi sono costrutti L2, che sono astrazioni dei costrutti L1.

I pacchetti contengono anche una raccolta di TypeScript tipi, enumerazioni e interfacce, nonché classi di supporto che aggiungono ulteriori funzionalità, ma questi elementi sono tutti compatibili con costrutti L2. Tutti i costrutti L2 richiamano i costrutti L1 corrispondenti nei loro costruttori al momento dell'istanziazione e il costrutto L1 risultante che viene creato è accessibile dal livello 2 in questo modo:

```
const role = new Bucket(this, "amzn-s3-demo-bucket", {/*...BucketProps*/});
const cfnBucket = role.node.defaultChild;
```

Il costrutto L2 prende le proprietà predefinite, i metodi di convenienza e altri principi sintattici e li applica al costrutto L1. Ciò elimina gran parte della ripetizione e della verbosità necessarie per fornire risorse direttamente all'interno. CloudFormation

Tutti i costrutti L2 costruiscono i costrutti L1 corrispondenti sotto il cofano. Tuttavia, i costrutti L2 in realtà non estendono i costrutti L1. Entrambi i costrutti L1 e L2 ereditano una classe speciale chiamata Construct. Nella versione 1 della AWS CDK`Construct` classe era integrata nel kit di sviluppo, ma nella versione 2 è un pacchetto autonomo separato. In questo modo altri pacchetti come il Cloud Development Kit for Terraform (CDKTF) possono includerlo come dipendenza. Qualsiasi classe che eredita la `Construct` classe è un costrutto L1, L2 o L3. I costrutti L2 estendono direttamente questa classe mentre i costrutti L1 estendono una classe chiamata, come illustrato nella tabella seguente. `CfnResource`

Albero di ereditarietà L1	Albero di ereditarietà L2
costrutto L1	costrutto L2
→ classe CfnResource	→ classe Construct
→ classe astratta CfnRefElement	

→→→ classe astratta CfnElement

→→→→ classe Construct

Se entrambi i costrutti L1 e L2 ereditano la `Construct` classe, perché i costrutti L2 non estendono semplicemente L1? Bene, le classi tra la `Construct` classe e il livello 1 bloccano il costrutto L1 come immagine speculare della risorsa. CloudFormation Contengono metodi astratti (metodi che le classi a valle devono includere) come`_toCloudFormation`, che costringono il costrutto a generare direttamente la sintassi. CloudFormation I costrutti L2 ignorano queste classi ed estendono direttamente la classe. `Construct` Ciò offre loro la flessibilità necessaria per astrarre gran parte del codice necessario per i costrutti L1 costruendoli separatamente all'interno dei rispettivi costruttori.

La sezione precedente presentava un side-by-side confronto tra un bucket S3 di un CloudFormation modello e lo stesso bucket S3 reso come un costrutto L1. Tale confronto ha dimostrato che le proprietà e la sintassi sono quasi identiche e che il costrutto L1 salva solo tre o quattro righe rispetto al costrutto. CloudFormation Ora confrontiamo il costrutto L1 con il costrutto L2 per lo stesso bucket S3:

Costrutto L1 per bucket S3

```
new CfnBucket(this, "amzns3de
mobucket", {
  bucketName: "amzn-s3-demo-bucket",
  bucketEncryption: {
    serverSideEncryptionConfigu
ration: [
      {
        serverSideEncryptionByDefau
lt: {
          sseAlgorithm: "AES256"
        }
      }
    ]
  },
  metricsConfigurations: [
    {
      id: "myConfig"
    }
  ],
  ownershipControls: {
```

Costruzione L2 per bucket S3

```
new Bucket(this, "amzns3demobucket",
 {
  bucketName: "amzn-s3-demo-bucket",
  encryption: BucketEncryption.S
3_MANAGED,
  metrics: [
    {
      id: "myConfig"
    },
  ],
  objectOwnership: ObjectOwnership.BU
CKET_OWNER_PREFERRED,
  blockPublicAccess: BlockPubl
icAccess.BLOCK_ALL,
  versioned: true
});
```

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

Come puoi vedere, il costrutto L2 è meno della metà delle dimensioni del costrutto L1. I costrutti L2 utilizzano numerose tecniche per realizzare questo consolidamento. Alcune di queste tecniche si applicano a un singolo costrutto L2, ma altre possono essere riutilizzate su più costrutti in modo da essere separate in una classe separata per la riutilizzabilità. I costrutti L2 consolidano la CloudFormation sintassi in diversi modi, come illustrato nelle sezioni seguenti.

Proprietà predefinite

Il modo più semplice per consolidare il codice per il provisioning di una risorsa consiste nel trasformare le impostazioni delle proprietà più comuni in impostazioni predefinite. AWS CDK Ha accesso a potenti linguaggi di programmazione CloudFormation ma non lo fa, quindi queste impostazioni predefinite sono spesso di natura condizionale. A volte è possibile eliminare diverse righe di CloudFormation configurazione dal AWS CDK codice perché tali impostazioni possono essere dedotte dai valori di altre proprietà che vengono passate al costrutto.

Strutture, tipi e interfacce

Sebbene AWS CDK sia disponibile in diversi linguaggi di programmazione, è scritto in modo nativo TypeScript, quindi il sistema di tipi di quel linguaggio viene utilizzato per definire i tipi che compongono i costrutti L2. L'approfondimento di questo tipo di sistema non rientra nello scopo di

questa guida; consulta la TypeScriptdocumentazione per i dettagli. Riassumendo, a TypeScript `type` descrive il tipo di dati contenuti in una particolare variabile. Potrebbero trattarsi di dati di base come a `string` o dati più complessi come un`object`. A TypeScript `interface` è un altro modo di esprimere il tipo di TypeScript oggetto e a `struct` è un altro nome per un'interfaccia.

TypeScript non usa il termine struct, ma se guardi nell'AWS CDK API Reference, vedrai che una struttura è in realtà solo un'altra TypeScript interfaccia all'interno del codice. L'API Reference si riferisce anche a determinate interfacce come interfacce. Se le strutture e le interfacce sono la stessa cosa, perché la AWS CDK documentazione le distingue?

Ciò che AWS CDK chiamano strutture sono interfacce che rappresentano qualsiasi oggetto utilizzato da un costrutto L2. Ciò include i tipi di oggetto per gli argomenti delle proprietà che vengono passati al costrutto L2 durante l'istanziazione, ad esempio per il costrutto S3 Bucket e `BucketProps` per `TableProps` il costrutto DynamoDB Table, nonché altre interfacce utilizzate all'interno di. TypeScript AWS CDKIn breve, se si tratta di un' TypeScript interfaccia all'interno di AWS CDK e il suo nome non è preceduto dalla lettera, la chiama struttura. *`I`* AWS CDK

Al contrario, AWS CDK usa il termine interfaccia per rappresentare gli elementi di base, un oggetto semplice dovrebbe essere considerato una rappresentazione corretta di un particolare costrutto o classe di supporto. Cioè, un'interfaccia descrive quali devono essere le proprietà pubbliche di un costrutto L2. Tutti i nomi di AWS CDK interfaccia sono nomi di costrutti o classi di supporto esistenti preceduti dalla lettera. `I` Tutti i costrutti L2 estendono la `Construct` classe, ma implementano anche l'interfaccia corrispondente. Quindi il costrutto `Bucket` L2 implementa l'interfaccia. `IBucket`

Metodi statici

Ogni istanza di un costrutto L2 è anche un'istanza dell'interfaccia corrispondente, ma non è vero il contrario. Questo è importante quando si esamina una struttura per vedere quali tipi di dati sono richiesti. Se una struttura ha una proprietà chiamata`bucket`, che richiede il tipo di dati`IBucket`, è possibile passare un oggetto che contiene le proprietà elencate nell'`IBucket`interfaccia o un'istanza di un L2`Bucket`. Entrambi funzionerebbero. Tuttavia, se quella `bucket` proprietà richiedesse un L2`Bucket`, potresti passare solo un'`Bucket`istanza in quel campo.

Questa distinzione diventa molto importante quando importate risorse preesistenti nel vostro stack. È possibile creare un costrutto L2 per qualsiasi risorsa nativa dello stack, ma se è necessario fare riferimento a una risorsa creata all'esterno dello stack, è necessario utilizzare l'interfaccia del costrutto L2. Questo perché la creazione di un costrutto L2 crea una nuova risorsa se non ne esiste

già una all'interno di quello stack. I riferimenti alle risorse esistenti devono essere oggetti semplici conformi all'interfaccia di quel costrutto L2.

Per semplificare questa operazione nella pratica, alla maggior parte dei costrutti L2 è associata una serie di metodi statici che restituiscono l'interfaccia del costrutto L2. Questi metodi statici di solito iniziano con la parola. `from` I primi due argomenti passati a questi metodi sono gli stessi `scope` e `id` gli argomenti richiesti per un costrutto L2 standard. Tuttavia, il terzo argomento non è `props` altro che un piccolo sottoinsieme di proprietà (o talvolta solo una proprietà) che definisce un'interfaccia. Per questo motivo, quando si passa un costrutto L2, nella maggior parte dei casi sono necessari solo gli elementi dell'interfaccia. In questo modo è possibile utilizzare anche le risorse importate, ove possibile.

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

Tuttavia, non dovresti fare molto affidamento sulle interfacce. Dovreste importare risorse e utilizzare le interfacce direttamente solo quando assolutamente necessario, perché le interfacce non forniscono molte delle proprietà, come i metodi di supporto, che rendono un costrutto L2 così potente.

Metodi di supporto

Un costrutto L2 è una classe programmatica piuttosto che un semplice oggetto, quindi può esporre metodi di classe che consentono di manipolare la configurazione delle risorse dopo che è avvenuta l'istanziazione. Un buon esempio di ciò è il costrutto (IAM) L2 Role AWS Identity and Access Management . I seguenti frammenti mostrano due modi per creare lo stesso ruolo IAM utilizzando il costrutto L2. `Role`

Senza un metodo di supporto:

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

Con un metodo di supporto:

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

La possibilità di utilizzare metodi di istanza per manipolare la configurazione delle risorse dopo l'istanziazione offre ai costrutti L2 molta flessibilità aggiuntiva rispetto al livello precedente. I costrutti L1 ereditano anche alcuni metodi relativi alle risorse (come`addPropertyOverride`), ma solo al secondo livello si ottengono metodi progettati specificamente per quella risorsa e le sue proprietà.

enumerazioni;

CloudFormation La sintassi spesso richiede di specificare molti dettagli per fornire correttamente una risorsa. Tuttavia, la maggior parte dei casi d'uso è spesso coperta solo da una manciata di configurazioni. La rappresentazione di tali configurazioni utilizzando una serie di valori enumerati può ridurre notevolmente la quantità di codice necessaria.

Ad esempio, nell'esempio di codice L2 del bucket S3 riportato in precedenza in questa sezione, è necessario utilizzare la `bucketEncryption` proprietà del CloudFormation modello per fornire

tutti i dettagli, incluso il nome dell'algoritmo di crittografia da utilizzare. AWS CDK Fornisce invece l'`BucketEncryption`enum, che utilizza le cinque forme più comuni di crittografia a bucket e consente di esprimerle ciascuna utilizzando nomi di singole variabili.

Che dire dei casi limite che non sono coperti dalle enumerazioni? Uno degli obiettivi di un costrutto L2 è semplificare il compito di approvvigionamento di una risorsa di livello 1, quindi alcuni casi limite che sono meno comuni potrebbero non essere supportati nel livello 2. Per supportare questi casi limite, AWS CDK consente di manipolare direttamente le proprietà CloudFormation delle risorse sottostanti utilizzando il metodo. addPropertyOverride Per ulteriori informazioni sulle sostituzioni delle proprietà, consultate la sezione Best practice di questa guida e la sezione Abstractions and escape hatches nella documentazione. AWS CDK

Classi di supporto

A volte un enum non è in grado di soddisfare la logica programmatica necessaria per configurare una risorsa per un determinato caso d'uso. In queste situazioni, AWS CDK spesso offre invece una classe di supporto. Un enum è un oggetto semplice che offre una serie di coppie chiave-valore, mentre una classe helper offre tutte le funzionalità di una classe. TypeScript Una classe helper può comunque agire come un'enum esponendo proprietà statiche, ma tali proprietà potrebbero quindi avere i loro valori impostati internamente con la logica condizionale nel costruttore della classe helper o in un metodo di supporto.

Quindi, sebbene l'`BucketEncryption`enum possa ridurre la quantità di codice necessaria per impostare un algoritmo di crittografia su un bucket S3, la stessa strategia non funzionerebbe per impostare le durate temporali perché ci sono semplicemente troppi valori possibili tra cui scegliere. Creare un enum per ogni valore sarebbe molto più difficile che utile. Per questo motivo, viene utilizzata una classe helper per le impostazioni di configurazione S3 Object Lock predefinite di un bucket S3, rappresentate dalla classe. ObjectLockRetention `ObjectLockRetention`contiene due metodi statici: uno per il mantenimento della conformità e l'altro per il mantenimento della governance. Entrambi i metodi utilizzano un'istanza della classe helper Duration come argomento per esprimere la quantità di tempo per cui il blocco deve essere configurato.

Un altro esempio è la classe AWS Lambda helper Runtime. A prima vista, potrebbe sembrare che le proprietà statiche associate a questa classe possano essere gestite da un enum. Tuttavia, sotto il cofano, ogni valore di proprietà rappresenta un'istanza della `Runtime` classe stessa, quindi la logica eseguita nel costruttore della classe non può essere ottenuta all'interno di un enum.

Costrutti di livello 3

Se i costrutti L1 eseguono una traduzione letterale delle CloudFormation risorse in codice programmatico e i costrutti L2 sostituiscono gran parte della CloudFormation sintassi dettagliata con metodi di supporto e logica personalizzata, cosa fanno i costrutti L3? La risposta a questa domanda è limitata solo dalla tua immaginazione. È possibile creare il livello 3 per adattarlo a qualsiasi caso d'uso specifico. Se il progetto necessita di una risorsa con un sottoinsieme specifico di proprietà, è possibile creare un costrutto L3 riutilizzabile per soddisfare tale esigenza.

I costrutti L3 sono chiamati pattern all'interno di. AWS CDK Un pattern è qualsiasi oggetto che estende la `Construct` classe in AWS CDK (o estende una classe che estende la `Construct` classe) per eseguire qualsiasi logica astratta oltre il livello 2. Quando si utilizza la AWS CDK CLI per eseguire cdk init per avviare un nuovo AWS CDK progetto, è necessario scegliere tra tre tipi di AWS CDK applicazione:`app`, e. `lib` `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]
```

`apped` `sample-app` entrambe rappresentano AWS CDK applicazioni classiche in cui si creano e distribuiscono CloudFormation stack in ambienti AWS. Quando sceglilib, scegli di creare un costrutto L3 nuovo di zecca. `app`e ti `sample-app` consentono di scegliere qualsiasi lingua AWS CDK supportata, ma puoi scegliere TypeScript solo con. `lib` Questo perché è scritto in modo nativo TypeScript e utilizza un sistema open source chiamato JSii per tradurre il codice originale nelle altre lingue supportate. AWS CDK Quando scegli `lib` di avviare il tuo progetto, scegli di creare un'estensione per. AWS CDK

Qualsiasi classe che estende la `Construct` classe può essere un costrutto L3, ma i casi d'uso più comuni per il livello 3 sono le interazioni con le risorse, le estensioni delle risorse e le risorse personalizzate. La maggior parte dei costrutti L3 utilizza uno o più di questi tre casi per estendere le funzionalità. AWS CDK

Interazioni con le risorse

Una soluzione utilizza in genere diversi servizi AWS che funzionano insieme. Ad esempio, una CloudFront distribuzione Amazon utilizza spesso un bucket S3 come origine e AWS WAF per proteggersi dagli exploit comuni. AWS AppSync e Amazon API Gateway utilizzano spesso le tabelle Amazon DynamoDB come fonti di dati per le proprie API. Una pipeline utilizza AWS CodePipeline spesso Amazon S3 come sorgente AWS CodeBuild e per le sue fasi di costruzione. In questi casi è spesso utile creare un singolo costrutto L3 che gestisca il provisioning di due o più costrutti L2 interconnessi.

Ecco un esempio di costrutto L3 che fornisce una CloudFront distribuzione insieme alla sua origine S3, un record Amazon Route 53 e un certificato AWS Certificate Manager (ACM) AWS WAF per aggiungere un endpoint personalizzato con crittografia in transito, il tutto in un unico costrutto riutilizzabile:

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

Nota che Amazon S3 CloudFront, Route 53 e ACM utilizzano tutti costrutti L2, ma l'ACL Web (che definisce le regole per la gestione delle richieste Web) utilizza un costrutto L1. Questo perché si tratta di un pacchetto open source in evoluzione che non AWS CDK è completamente completo e non esiste ancora un costrutto L2. `WebAcl` Tuttavia, chiunque può contribuire alla creazione di nuovi AWS CDK costrutti L2. Quindi, fino a quando non AWS CDK offre un costrutto L2 per`WebAcl`, devi usare un costrutto L1. Per creare un nuovo sito Web utilizzando il costrutto L3`CloudFrontWebsite`, si utilizza il codice seguente:

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

In questo esempio, il costrutto CloudFront `Distribution` L2 è esposto come proprietà pubblica del costrutto L3. Ci saranno comunque casi in cui sarà necessario esporre proprietà L3 come questa, se necessario. In effetti lo vedremo di `Distribution` nuovo più avanti, nella sezione Risorse personalizzate.

AWS CDK Include alcuni esempi di modelli di interazione con le risorse come questo. Oltre al `aws-ecs` pacchetto che contiene i costrutti L2 per Amazon Elastic Container Service (Amazon ECS), AWS CDK ha un pacchetto chiamato aws-ecs-patterns. Questo pacchetto contiene diversi costrutti L3 che combinano Amazon ECS con Application Load Balancers, Network Load Balancer e gruppi target, offrendo al contempo diverse versioni preimpostate per Amazon Elastic Compute Cloud (Amazon EC2) e. AWS Fargate Poiché molte applicazioni serverless utilizzano Amazon ECS solo con Fargate, questi costrutti L3 offrono una praticità che può far risparmiare tempo agli sviluppatori e denaro ai clienti.

Estensioni di risorse

Alcuni casi d'uso richiedono che le risorse abbiano impostazioni predefinite specifiche che non sono native del costrutto L2. A livello di stack, questo può essere gestito utilizzando gli aspetti, ma un altro modo conveniente per assegnare nuovi valori predefiniti a un costrutto L2 è estendere il livello 2. Poiché un costrutto è qualsiasi classe che eredita la classe e i costrutti L2 estendono quella `Construct` classe, potete anche creare un costrutto L3 estendendo direttamente un costrutto L2.

Ciò può essere particolarmente utile per una logica aziendale personalizzata che supporta le esigenze specifiche di un cliente. Supponiamo che un'azienda disponga di un repository che memorizza tutto il codice di AWS Lambda funzione in un'unica directory chiamata `src/lambda` e che la maggior parte delle funzioni Lambda riutilizzi ogni volta lo stesso runtime e lo stesso nome del gestore. Invece di configurare il percorso del codice ogni volta che configuri una nuova funzione Lambda, puoi creare un nuovo costrutto L3:

```
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
     });
}
```

È quindi possibile sostituire il `Function` costrutto L2 in qualsiasi punto del repository nel modo seguente:

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

Le impostazioni predefinite consentono di creare nuove funzioni Lambda su una singola riga e il costrutto L3 è configurato in modo da poter comunque sovrascrivere le proprietà predefinite, se necessario.

L'estensione diretta dei costrutti L2 funziona meglio quando si desidera semplicemente aggiungere nuovi valori predefiniti ai costrutti L2 esistenti. Se hai bisogno anche di altre logiche personalizzate, è meglio estendere la classe. `Construct` La ragione di ciò deriva dal `super` metodo, che viene chiamato all'interno del costruttore. Nelle classi che estendono altre classi, il `super` metodo viene utilizzato per chiamare il costruttore della classe madre e questa deve essere la prima cosa che accade all'interno del costruttore. Ciò significa che qualsiasi manipolazione degli argomenti passati o di altra logica personalizzata può avvenire solo dopo la creazione del costrutto L2 originale. Se è necessario eseguire una di queste logiche personalizzate prima di creare un'istanza del costrutto L2, è meglio seguire lo schema descritto in precedenza nella sezione Interazioni con le risorse.

Risorse personalizzate

Le risorse personalizzate sono una potente funzionalità CloudFormation che consente di eseguire la logica personalizzata da una funzione Lambda attivata durante la distribuzione dello stack. Ogni volta che durante la distribuzione sono necessari processi che non sono direttamente supportati da CloudFormation, puoi utilizzare una risorsa personalizzata per realizzarli. AWS CDK Offre classi che consentono di creare risorse personalizzate anche a livello di codice. Utilizzando risorse personalizzate all'interno di un costruttore L3, è possibile creare un costrutto praticamente con qualsiasi cosa.

Uno dei vantaggi dell'utilizzo di Amazon CloudFront è la sua forte capacità di caching globale. Se desideri reimpostare manualmente la cache in modo che il tuo sito web rifletta immediatamente le nuove modifiche apportate alla tua origine, puoi utilizzare l'CloudFront invalidazione. Tuttavia, le invalidazioni sono processi che vengono eseguiti su una CloudFront distribuzione anziché essere proprietà di una distribuzione. CloudFront Possono essere creati e applicati a una distribuzione esistente in qualsiasi momento, quindi non fanno parte nativa del processo di provisioning e distribuzione.

In questo scenario, potresti voler creare ed eseguire un'invalidazione dopo ogni aggiornamento dell'origine di una distribuzione. Grazie alle risorse personalizzate, puoi creare un costrutto L3 simile al seguente:

```
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')
                     }
                  }
             }
          }
     }
}
```

Utilizzando la distribuzione che abbiamo creato in precedenza nel costrutto `CloudFrontWebsite` L3, è possibile farlo molto facilmente:

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

Questo costrutto L3 utilizza un costrutto AWS CDK L3 chiamato AwsCustomResourceper creare una risorsa personalizzata che esegue una logica personalizzata. `AwsCustomResource`è molto comodo quando devi effettuare esattamente una chiamata SDK AWS, perché ti consente di farlo senza dover scrivere alcun codice Lambda. Se hai requisiti più complessi e desideri implementare la tua logica, puoi usare direttamente la CustomResourceclasse base.

Un altro buon esempio di AWS CDK utilizzo di un costrutto L3 di risorse personalizzato è l'implementazione di bucket S3. La funzione Lambda creata dalla risorsa personalizzata all'interno del costruttore di questo costrutto L3 aggiunge funzionalità che altrimenti non CloudFormation sarebbero in grado di gestire: aggiunge e aggiorna oggetti in un bucket S3. Senza l'implementazione del bucket S3, non saresti in grado di inserire contenuti nel bucket S3 che hai appena creato come parte del tuo stack, il che sarebbe molto scomodo.

Il miglior esempio di come AWS CDK eliminare la necessità di scrivere grandi quantità di sintassi è questo: CloudFormation `S3BucketDeployment`

```
new BucketDeployment(this, 'BucketObjects', {
  sources: [Source.asset('./path/to/amzn-s3-demo-bucket')],
  destinationBucket: amzn-s3-demo-bucket
});
```

Confrontalo con il CloudFormation codice che dovresti scrivere per ottenere la stessa cosa:

```
"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": "amzn-s3-demo-bucket77F80CC0"
```

```
    },
    "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": [
           "amzns3demobucket77F80CC0",
           "Arn"
          ]
         },
         {
          "Fn::Join": [
           "",
           [
            {
             "Fn::GetAtt": [
              "amzns3demobucket77F80CC0",
              "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 righe contro 241 righe è un'enorme differenza! E questo è solo un esempio di ciò che è possibile fare sfruttando il livello 3 per personalizzare gli stack.

Best practice

Costrutti L1

- Non è sempre possibile evitare di utilizzare direttamente i costrutti L1, ma è consigliabile evitarlo ogni volta che è possibile. Se uno specifico costrutto L2 non supporta il vostro edge case, potete esplorare queste due opzioni invece di utilizzare direttamente il costrutto L1:
  - Accesso`defaultChild`: se la CloudFormation proprietà di cui hai bisogno non è disponibile in un costrutto L2, puoi accedere al costrutto L1 sottostante utilizzando. `L2Construct.node.defaultChild` È possibile aggiornare qualsiasi proprietà pubblica del costrutto L1 accedendovi tramite questa proprietà invece di dover creare autonomamente il costrutto L1.
  - Usa le sostituzioni delle proprietà: cosa succede se la proprietà che desideri aggiornare non è pubblica? L'ultima via di fuga che consente di AWS CDK fare tutto ciò che un CloudFormation modello può fare è utilizzare un metodo disponibile in ogni costrutto L1:. addPropertyOverride Puoi manipolare lo stack a livello di CloudFormation modello passando il nome e il valore della CloudFormation proprietà direttamente a questo metodo.

Costrutti L2

- Ricorda di sfruttare i metodi di supporto che spesso offrono i costrutti L2. Con il livello 2, non è necessario passare tutte le proprietà al momento dell'istanziazione. I metodi di supporto L2 possono rendere il provisioning delle risorse in modo esponenziale più conveniente, specialmente quando è necessaria la logica condizionale. Uno dei metodi di supporto più convenienti è derivato dalla classe Grant. Questa classe non viene utilizzata direttamente, ma molti costrutti L2 la utilizzano per fornire metodi di supporto che semplificano notevolmente l'implementazione delle autorizzazioni. Ad esempio, se desideri autorizzare una funzione Lambda L2 ad accedere a un bucket L2 S3, puoi `s3Bucket.grantReadWrite(lambdaFunction)` chiamare invece di creare un nuovo ruolo e una nuova politica.

Costrutti L3

- Sebbene i costrutti L3 possano essere molto comodi quando si desidera rendere gli stack più riutilizzabili e personalizzabili, si consiglia di utilizzarli con attenzione. Considerate il tipo di costrutto L3 di cui avete bisogno o se avete bisogno di un costrutto L3:

- Se non interagite direttamente con AWS le risorse, spesso è più appropriato creare una classe di supporto invece di estendere la classe. `Construct` Questo perché la `Construct` classe esegue per impostazione predefinita molte azioni necessarie solo se interagisci direttamente con le risorse. AWS Quindi, se non hai bisogno che queste azioni vengano eseguite, è più efficiente evitarle.
- Se ritenete che la creazione di un nuovo costrutto L3 sia appropriata, nella maggior parte dei casi vorrete estendere direttamente la `Construct` classe. Estendete altri costrutti L2 solo quando desiderate aggiornare le proprietà predefinite del costrutto. Se sono coinvolti altri costrutti L2 o logica personalizzata, estendete `Construct` direttamente e istanziate tutte le risorse all'interno del costruttore.

Domande frequenti

Non posso utilizzarli AWS CDK senza comprendere i livelli?

Puoi assolutamente farlo. Ma come con gli strumenti più potenti, più potente AWS CDK diventa quanto più ne sai. Imparare come interagiscono i diversi livelli consente AWS CDK di raggiungere un nuovo livello di comprensione che aiuta a semplificare le implementazioni degli stack ben oltre ciò che si può fare con una semplice conoscenza di base. AWS CDK

Posso creare costrutti L2 da L1 nello stesso modo in cui creo costrutti L3 da L2?

Se una risorsa ha già un costrutto L2, ti consigliamo di utilizzare quel costrutto e di effettuare le personalizzazioni nel livello 3. Questo perché sono già state condotte molte ricerche per capire i modi migliori per configurare i costrutti L2 esistenti per una particolare risorsa. Tuttavia, ci sono diversi costrutti L1 i cui costrutti L2 non esistono ancora. In questi casi, ti incoraggiamo a creare i tuoi costrutti L2 e condividerli con gli altri diventando un collaboratore della libreria open source. AWS CDK Puoi trovare tutto ciò di cui hai bisogno per iniziare nelle linee guida per i contributi di. AWS CDK

Quali AWS risorse non dispongono ancora di costrutti L2 ufficiali?

Il numero di AWS risorse che non hanno costrutti L2 diminuisce di giorno in giorno, ma se sei interessato a contribuire alla creazione di un costrutto L2 per una di queste risorse, visita l'API Reference.AWS CDK Guarda l'elenco delle risorse nel riquadro a sinistra. Le risorse che hanno l'apice 1 accanto ai loro nomi non hanno costrutti L2 ufficiali.

Posso creare un costrutto L2 o L3 in qualsiasi lingua supportata? AWS CDK

AWS CDK Supporta diversi linguaggi di programmazione TypeScript, tra cui Python JavaScript, Java, C# e Go. È possibile creare costrutti L3 personali utilizzando il AWS CDK codice compilato nel linguaggio pertinente. Tuttavia, se si desidera contribuire AWS CDK o creare AWS CDK costrutti nativi, è necessario utilizzare. TypeScript Questo perché TypeScript è l'unica lingua nativa di AWS

CDK. Le AWS CDK versioni per altre lingue sono create a partire dal TypeScript codice nativo utilizzando una AWS libreria chiamata JSii.

Dove posso trovare i costrutti L3 esistenti al di fuori di? AWS CDK

Ci sono troppe posizioni da condividere qui, ma puoi trovare molti dei costrutti più popolari sul sito Web di AWS Solutions Constructs e nella AWS CDK sezione del Construct Hub.

Risorse

- AWS CDK Documentazione di riferimento delle API
- AWS CloudFormation specificazione delle risorse
- AWS CDK costruisce la documentazione
- AWS CDK astrazioni e vie di fuga
- Sfrutta i costrutti L2 per ridurre la complessità della tua AWS CDK applicazione (post sul blog)AWS
- AWS CloudFormation risorse personalizzate
- AWS Costrutti di soluzioni
- Costruisci Hub
- AWS CDK Esempi (GitHub repository)

Cronologia dei documenti

La tabella seguente descrive le modifiche significative apportate a questa guida. Per ricevere notifiche sugli aggiornamenti futuri, puoi abbonarti a un feed RSS.

Modifica	Descrizione	Data
Pubblicazione iniziale	—	4 dicembre 2023

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.