aws DEV DAY ONLINE JAPAN

20.10.2020

B-06

AWS CDKにコントリビュートするための具体的な方法

佐藤智樹
クラスメソッド 株式会社

In Partnership with intel

自己紹介

- 佐藤智樹
- クラスメソッド株式会社
- CX事業本部（IoTチームと自社サービス開発）
- 前職：SIerのSEを4年弱
- 現在：サーバーサイドエンジニア
- 好きなAWSサービス：Lambda、CDK

@tmk2154

@tomoki10

In Partnership with intel

アジェンダ

- 対象とする視聴者
- コントリビュートすると良いこと
- AWS CDKの構成と詳細
- コントリビューティングガイドの説明
- 実際にコントリビュートした流れ
- まとめ

In Partnership with intel

対象とする視聴者

- AWS CDK or 大きめのOSSにコントリビュートしてみたい人
  エンジニアやってるんだし一度ぐらいはやってみたい
  せっかくだしドキュメントだけじゃなくてソースもコミットしたい

- AWS CDKは内部構成を少し知りたい
  適宜説明は入れますが、AWS CDKのBlackBeltを見ていると分かりやすいかも

目標

- 「これなら自分でも出来そう！」と思ってもらうこと

In Partnership with intel

コントリビュートすると良いこと

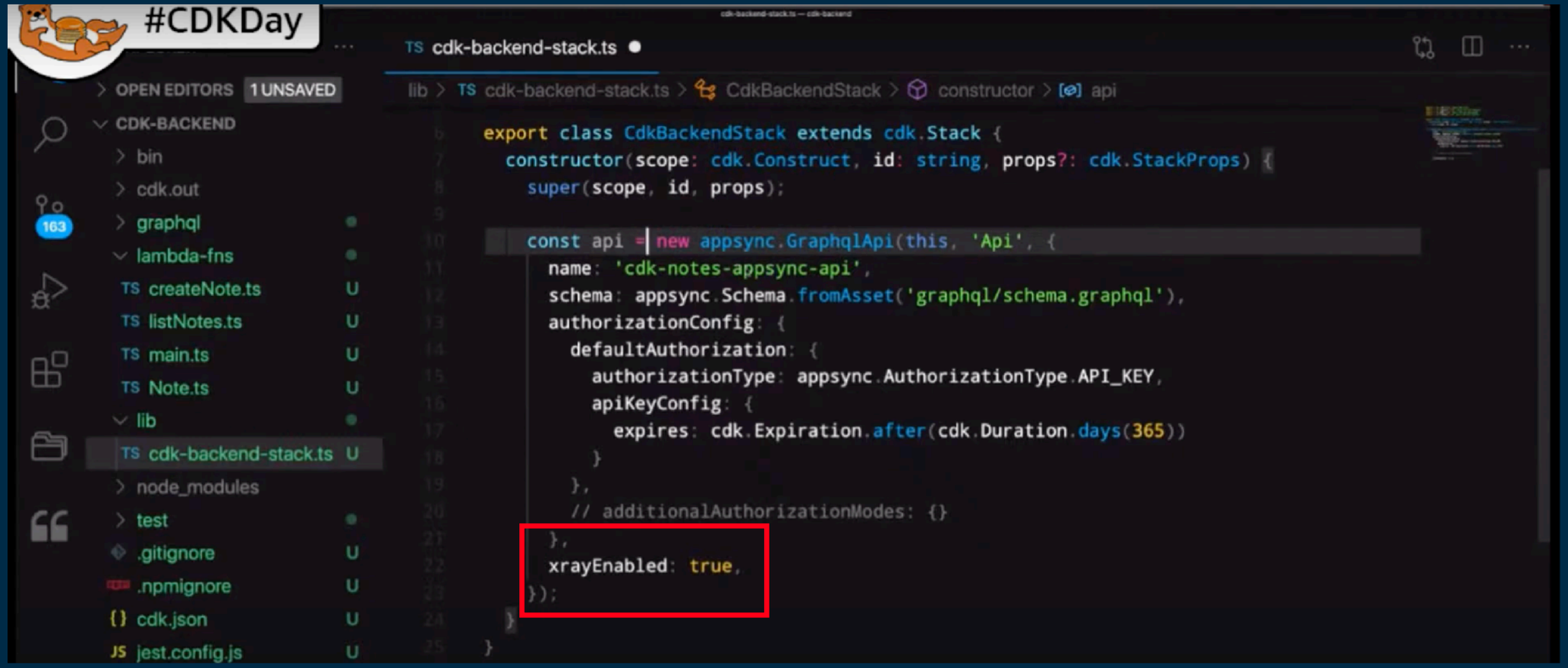
- 勉強になる
  普段CDKを使う際に構成を理解していると少しは早く実装できる
  CloudFormationの制約かCDKの問題かの区別がつきやすい

- 何の機能ができるのかいち早く知れる
  先日発表されたCLFからCDKのマイグレーションやv2の詳細を先に知れる

- 自分のソースが意外なとこで出たりして面白い

CDK Dayの様子

- Building Real-time Backends with AWS AppSync & CDK



https://www.youtube.com/watch?v=qJutZqXMdgM&t=4616s

In Partnership with intel

aws DEV DAY
ONLINE JAPAN

AWS CDKの構成と詳細

In Partnership with intel

AWS CDK(Cloud Developer Kit)とは

- 使い慣れた言語でAWSリソースをプロビジョニングできるツールキット

- 現在はTS/JS、Python、Java、C#で使用可能(今後はGoも?)
  ツールの内部の開発はTypeScriptメイン

- プログラミング言語でリソース管理ができる
  →真のIaCツール(個人の感想です)

- いずれは CDK for TerraformでGCPやAzureも管理?

AWS Cloud
Development Kit

In Partnership with intel

AWS CDKの個人的に思う利点(BlackBeltより一部引用)

- エディタで型チェック、サジェスト、仕様参照ができる
  →Webドキュメントなしでも仕様の確認や実装が可能

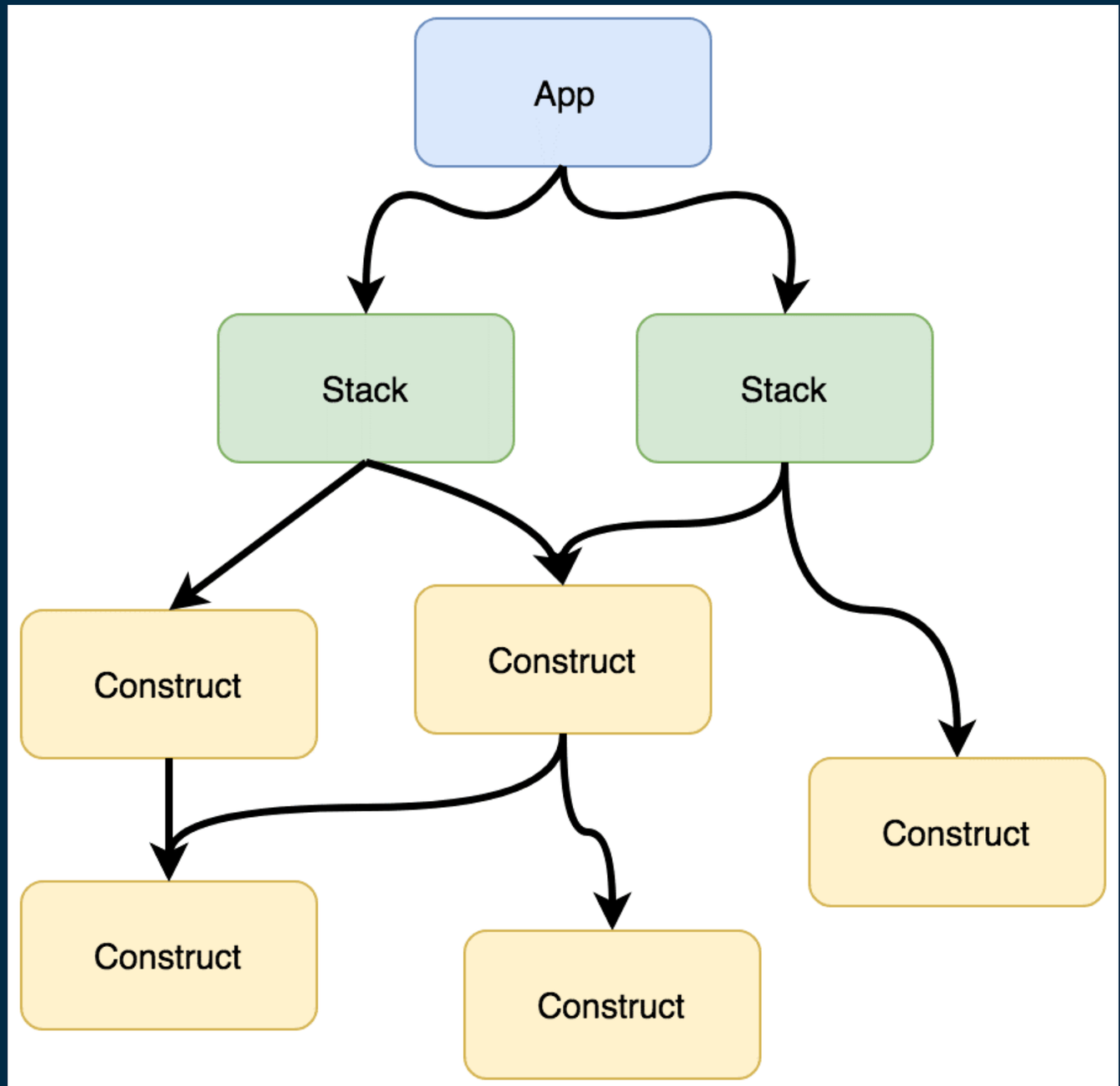
- 一般にコード量が減る
  →SQS+Lambda+SNSがCloudFormationの実装だと340行程
  CDKだと100行程※

- 複数スタック間の依存関係が記述できる
  →簡易なコードでAPI Gateway+Lambdaなどを別スタックに書けるなど

※「Moving from CFN YAML to CDK - Benefits, how to do it in prod and more」
https://www.youtube.com/watch?v=qJutZqXMdgM&t=12485s

AWS CDK アプリケーションの構成

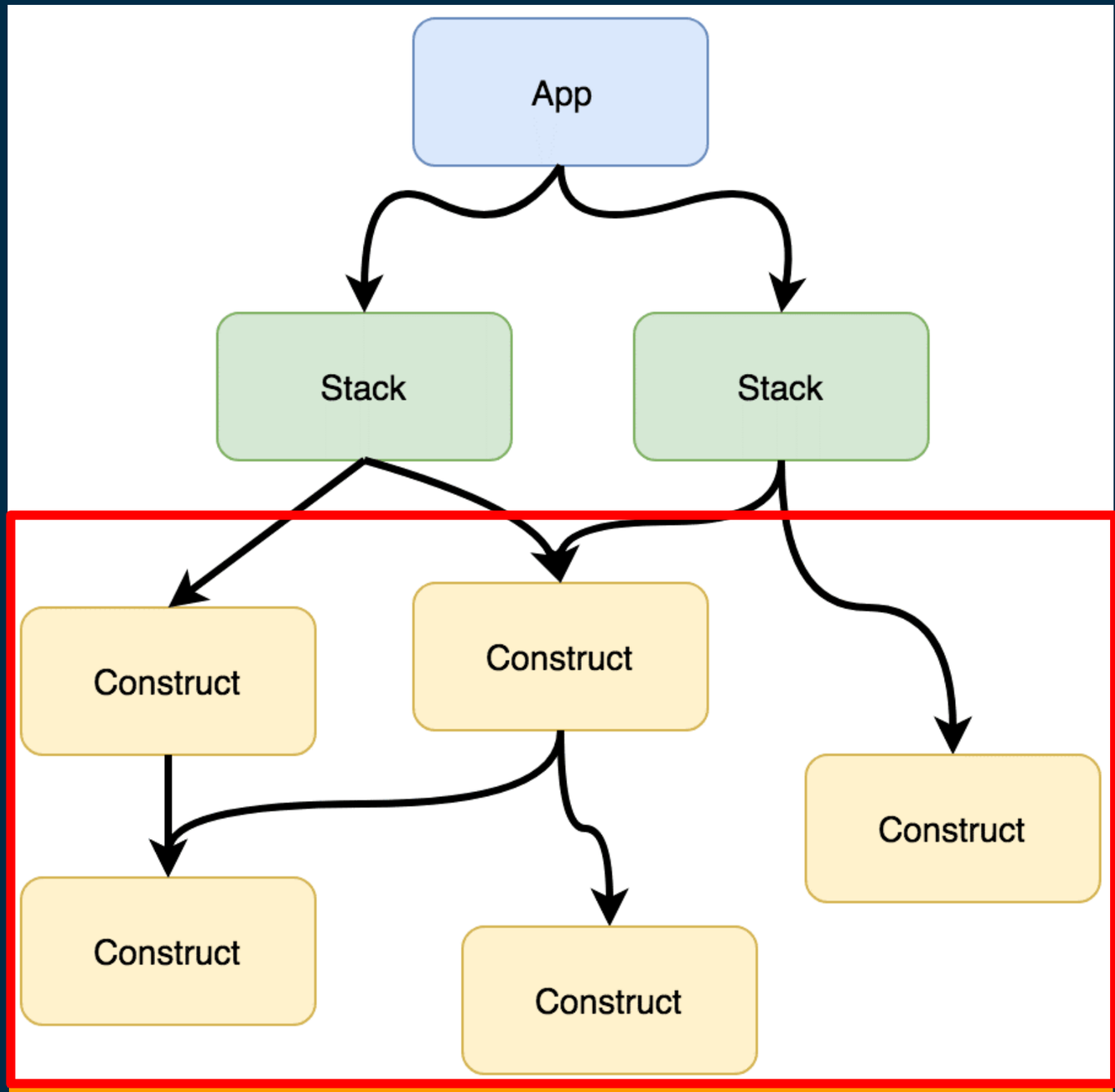
- App
  - CloudFormationテンプレート生成とデプロイに使用される要素
  - スタック間の依存関係を定義
- Stack
  - CloudFormationでいうスタック
  - デプロイの最小単位
- Construct
  - Stack内で生成するAWSリソース
  - Construct Libraryで定義



引用元:
https://aws.amazon.com/jp/blogs/aws/boost-your-infrastructure-with-cdk/

AWS CDK
アプリケーションの構成

- App
  - CloudFormationテンプレート生成とデプロイに使用される要素
  - スタック間の依存関係を定義
- Stack
  - CloudFormationでいうスタック
  - デプロイの最小単位
- Construct
  - Stack内で生成するAWSリソース
  - Construct Libraryで定義



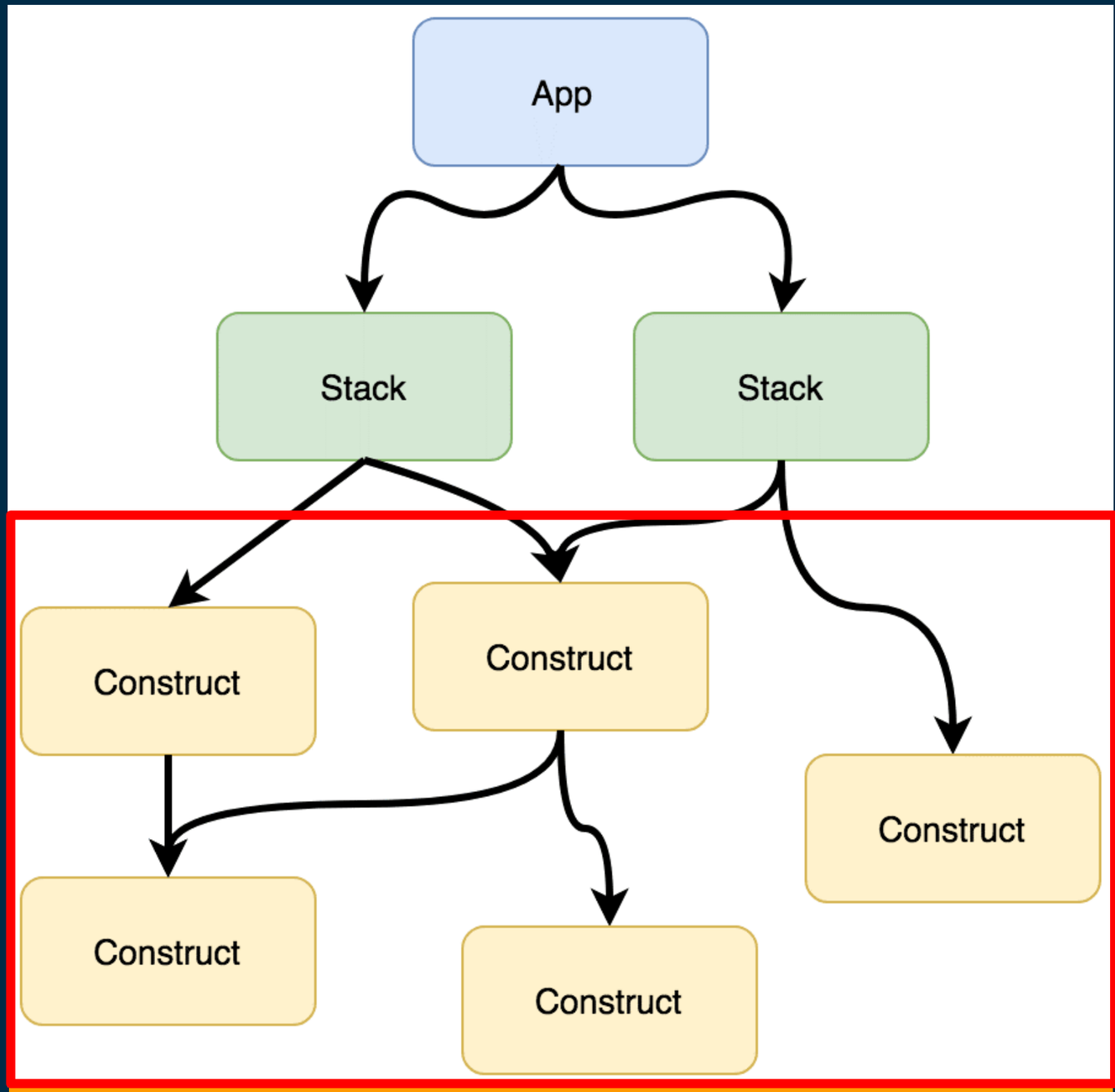
引用元:
https://aws.amazon.com/jp/blogs/aws/boost-your-infrastructure-with-cdk/

Constructの構成

- L1 (Low Level) Construct
  CloudFormationのリソースと
  1対1で対応 (自動生成)

- L2 (High Level) Construct
  デフォルト値や追加の関数を実装し
  てL1 Constructを抽象化

- L3 Construct (Patterns or etc...)
  複数リソースやL2 Constructをさら
  に抽象化して利用する場合



引用元:
https://aws.amazon.com/jp/blogs/aws/boost-your-infrastructure-with-cdk/

TIPS: Constructの構成(AppSyncの具体例)

- ベースとなるソース

  Githubリポジトリ：https://github.com/aws/aws-cdk

  上記リポジトリの aws-cdk/packages/@aws-cdk/aws-appsync

  @aws-cdk配下に各AWSリソース用のパッケージがある

- L1 (Low-Level) Construct
  aws-appsync/lib 配下の appsync.generated.ts ファイル

  ビルドしないと生成されない(GitHub上で見えない)ので Gitpod などで確認

- L2 (High-Level) Construct
  graphqlapi.ts、schema-base.ts などいくつかのファイルに分かれている

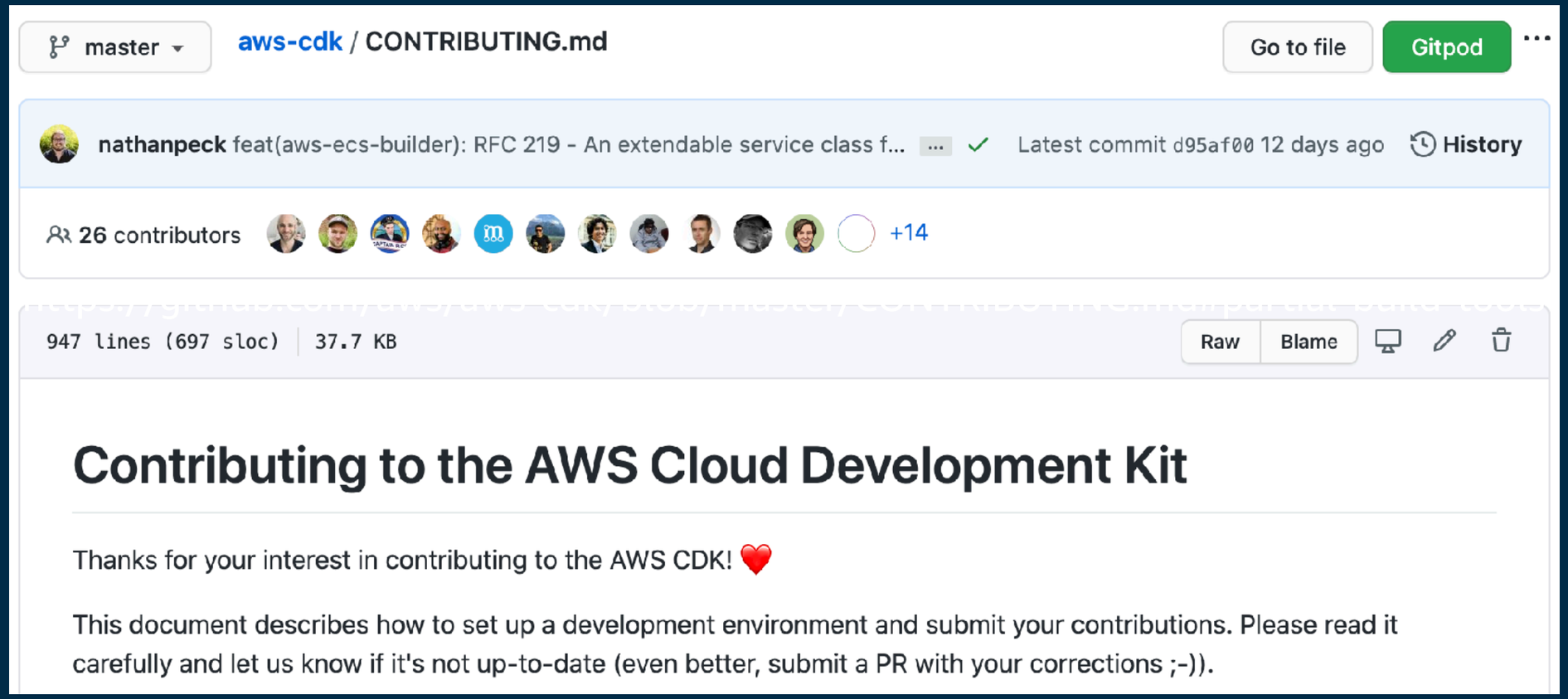
aws DEV DAY ONLINE JAPAN

コントリビューティングガイドの説明

In Partnership with intel

コントリビューティングガイドとは

- OSSへ貢献したい人向けに設計や実装方式を共有するドキュメント



https://github.com/aws/aws-cdk/blob/master/CONTRIBUTING.md

In Partnership with intel

コントリビューティングガイドの内容

- PRを送るまで段階ごとにやるべきことなどが書かれている

  今回は以下の3つを中心に紹介

  - 環境構築
  - issueから実装マージまでの流れ
  - 各種ツールの紹介

環境構築

Gitpod

GitHubからワンクリックで環境構築して使えるWebIDE(今回紹介)
→ソースの確認やちょっとした修正で有用

Docker

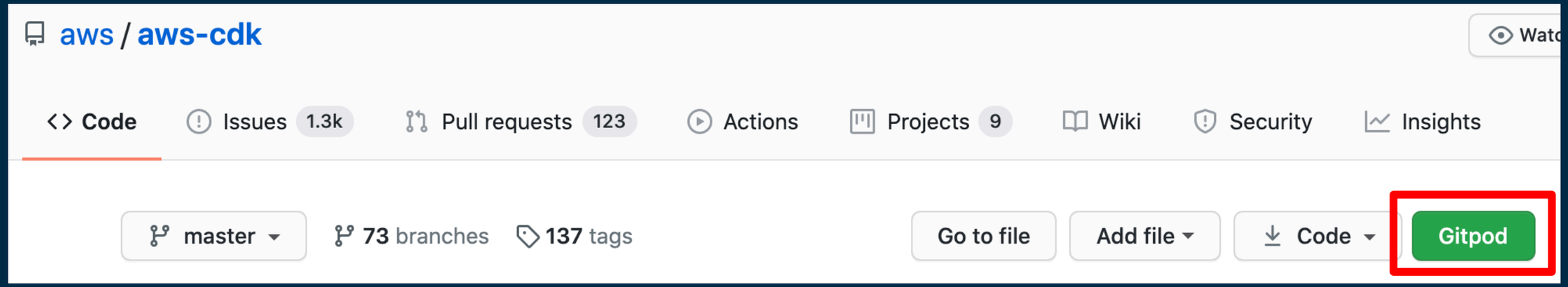
Dockerイメージから環境作成
→継続的な開発で有用

ローカル環境

Node.jsやJavaのバージョンを合わせて構築
→環境の管理が大変なので多分非推奨

Gitpodとは

- GitHubからワンクリックで環境構築して使えるWebIDE
  - VS Code ライクな開発環境がWeb上で使える
  - 初期のビルド設定もあるのでワンクリックで実装が始められる
  - 1ヶ月で50時間までは無料



In Partnership with intel

TIPS: GitpodをCDKの開発で使う場合の注意点

- Gitpod初回起動は1時間弱かかる

  全体ビルドが走るので立ち上がるまでが遅い。 1回ビルドが通れば再起動時は数分程度で動作可能

  2回目以降の起動はGitpodの画面で行う。(先ほどのボタンを押すと毎回新しいインスタンスができる)

- 途中でビルドがこけると別のリソースのせいでテストが通らない時もある

  対象のディレクトリにて scripts/buildup(後述) で関連するリソースを再度ビルドすると多分大丈夫

  最悪新しいGitpodの環境を立ち上げる

- Gitpod上のAccess Control※で権限を与えないとPushできない

  GitpodのWebUIで「write public repos」の権限を与えること

※https://gitpod.io/access-control/

In Partnership with intel

issueから実装マージまでの流れ

- コントリビューティングガイドをベースに少し変えています
  - issueを立てる/探す
  - 設計(オプション)
  - 実装(Magic)
    - 実装、単体/結合テスト
    - 動作確認
  - コミット作成ルールを確認
  - PRの作成
  - レビュー/マージ

コントリビューティングガイドの流れ

- Pull Requests
  - Pull Request Checklist
  - Step 1: Open Issue
  - Step 2: Design (optional)
  - Step 3: Work your Magic
  - Step 4: Commit
  - Step 5: Pull Request
  - Step 6: Merge

In Partnership with intel

issueを立てる/探す

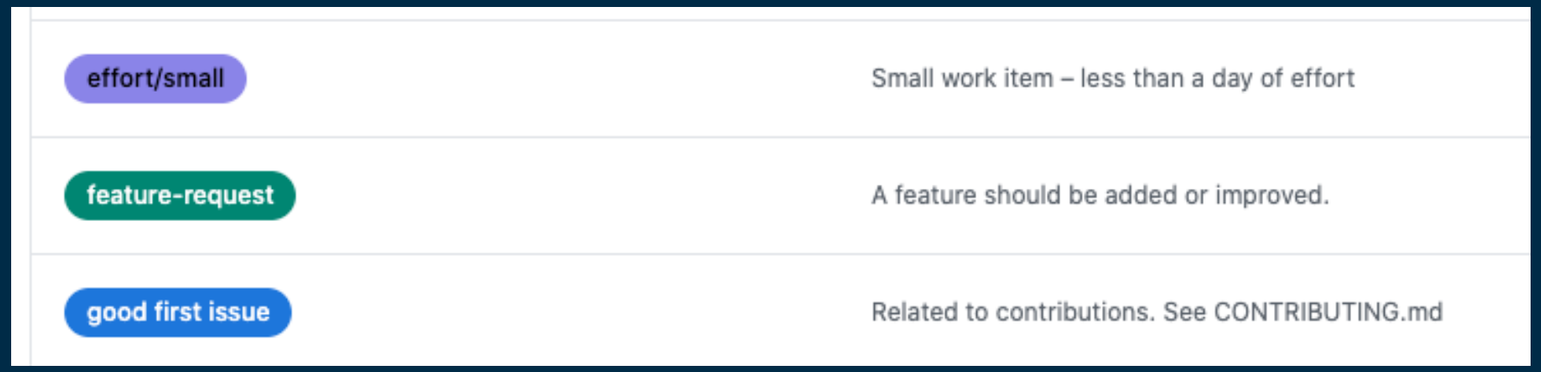
- 既存のissueを確認して、新しい問題であればissueを立てる

  着手できそうであればissueに書く

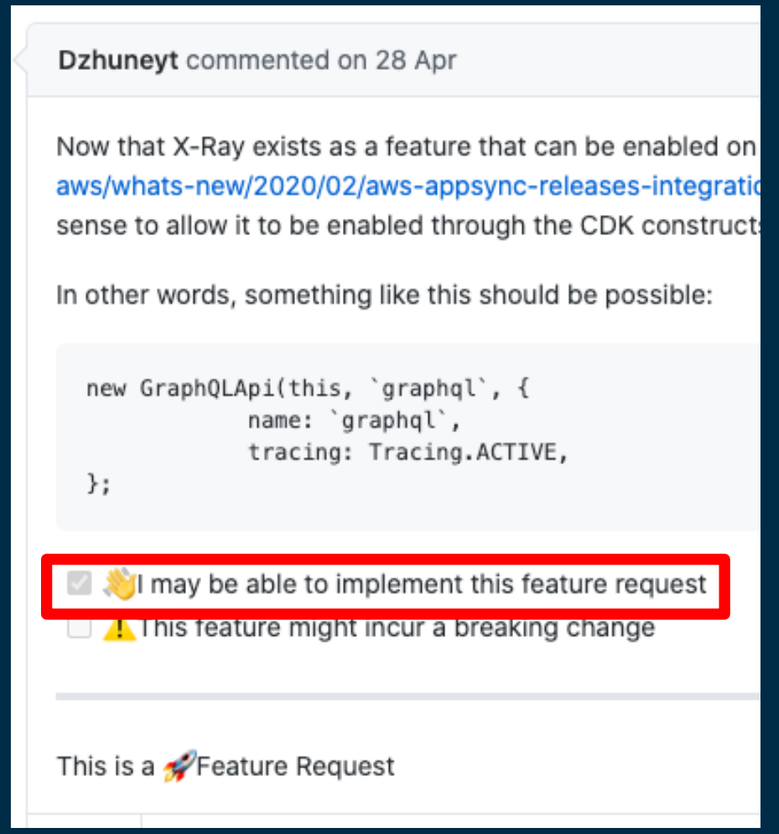
- 既存のissueで貢献できそうな内容を探す

  effort/small、good first issue ラベルが付いているissueは比較的簡単

  AWSリソースの改修の場合は「@aws-cdk/aws-xxx」のタグを探す



label 検索結果



Feature Request の例

In Partnership with intel

設計

- 中~大規模な変更の場合はあらかじめ設計を行う

  設計ガイドラインを確認の上でissueにコメントする

  →設計思想があってないと修正だらけに... 最初はわかりやすいissueから着手の方が良いかも

- 設計ガイドラインの確認は有用

  CDK全体の設計思想について書かれています（少し長い）

  https://github.com/aws/aws-cdk/blob/master/DESIGN_GUIDELINES.md

実装(Magic)

- 一般的なコーディングスタイルで実装

  2 space indent、横幅120文字など詳細はガイド参照

- すべての変更には単体テストが必要

  test配下のソースに単体/結合テストを追加する(Jestで実装)

- コメント追加も場合によって必要（修正しないとLintでwarningになる）

  外部から呼び出す際のパラメータ変更などAPIの変更はREADMEを更新

  パラメータ追加などの場合はjsdocsも更新

実装(単体テスト)

- テスト実装にはJestを使用(一部古いものはnode-unit)
  - JavaScript/TypeScriptで使えるテスティングフレームワーク
  - 他のフレームワーク(JUnitなど)を使ったことがあれば分かりやすい
- 単体テストの構成

  以下のテストが大半
  - Stackを生成して、StackにConstructを作成
  - Stackに想定したResourceが含まれているかチェック

In Partnership with intel

実装(単体テスト: appsync.test.tsの場合)

- Stackを生成して、StackにConstructを作成

```
let stack: cdk.Stack;
let api: appsync.GraphqlApi;
beforeEach(() => {
  stack = new cdk.Stack(); // 普段のbin配下の実装と同様
  api = new appsync.GraphqlApi(stack, 'api', { // lib配下と同様
    authorizationConfig: {},
    name: 'api',
    schema:
      appsync.Schema.fromAsset(path.join(__dirname,
        'appsync.test.graphql')),
  });
});
```

実装(単体テスト: appsync.test.tsの場合)

- テストしたいConstructを作成
- Stackに想定したResourceが含まれているかチェック

```
test('appsync should configure pipeline … contents’, () => {
  // WHEN
  new appsync.Resolver(stack, 'resolver', {
    api: api,
    typeName: 'test',
    fieldName: 'test2',
    pipelineConfig: ['test', 'test'],
  });
  // THEN
  expect(stack).toHaveResourceLike('AWS::AppSync::Resolver', {
    Kind: 'PIPELINE',
    PipelineConfig: { Functions: ['test', 'test'] },
  });
});
```

実装(結合テスト)

- 複数サービスに跨る変更やCLFを使った新しい機能を作る時に必要
- 結合テストの種類
  - CloudFormationテンプレートを生成して比較
    cdk synthを実行してtest配下の *.expected.json ファイルと比較
  - デプロイして検証
    yarn integ コマンドを実行するとAWSアカウントにデプロイ（AWSの認証情報設定は必要）

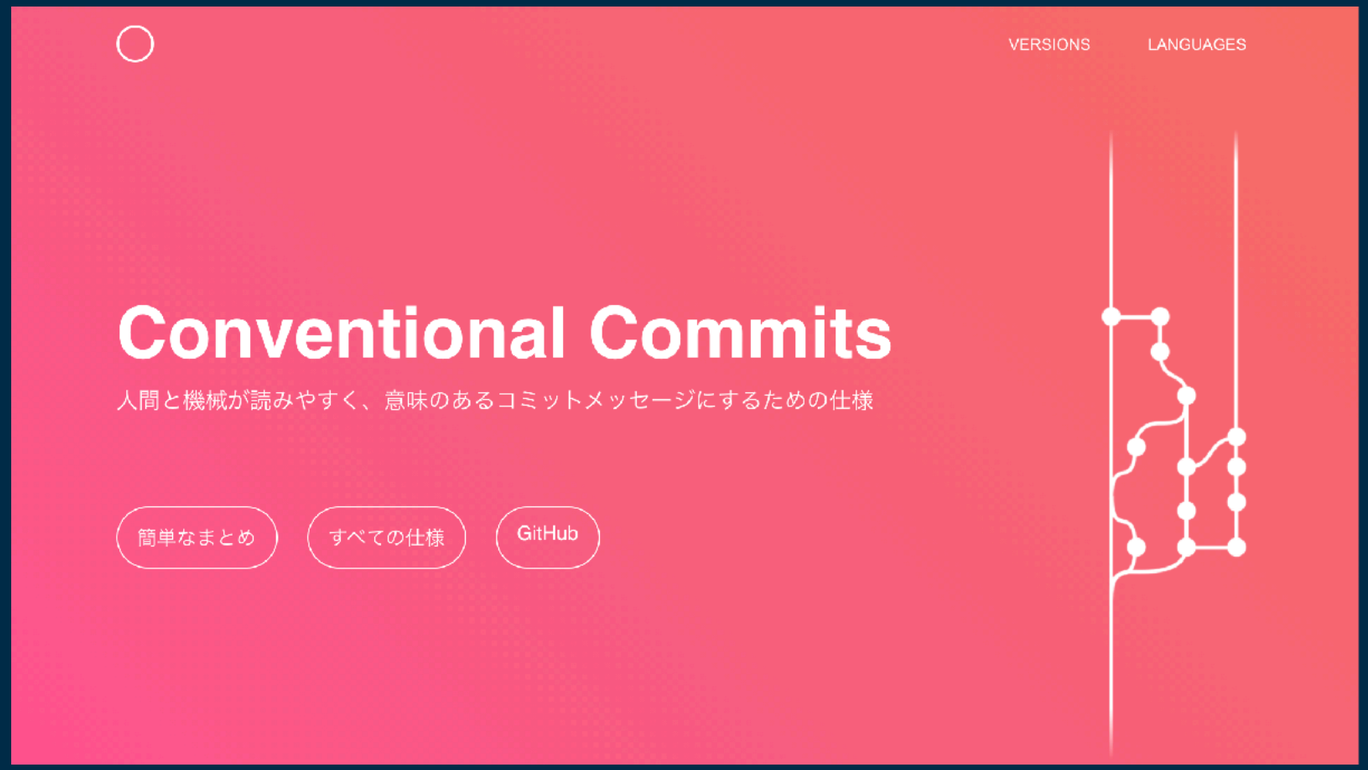
コミット

- タイトルは以下の文言で始める

  feat(module): title
  fix(module): title
  refactor(module): title
  chore(module): title

- 変更が自明でない場合は
  確認内容を記載

- 重大な変更がある場合は記載



コミットメッセージ作成のベースは「Conventional Commits」
https://www.conventionalcommits.org/ja/v1.0.0-beta.4/

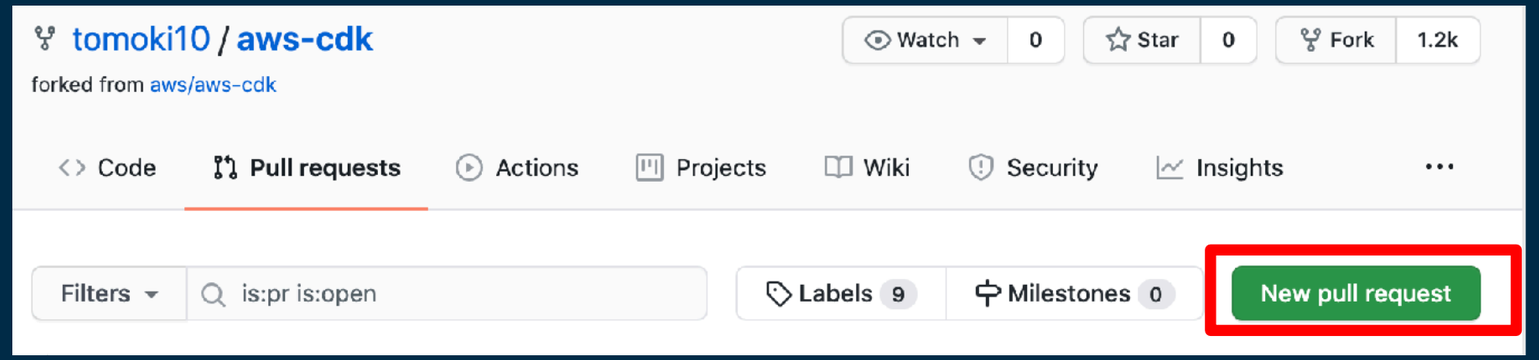
In Partnership with intel

Pull Request作成

- GitHubの画面上などからでPull Requestを作成

  普段GitHubフローなどで開発している場合はそんなに変わらない

- PRチェックリストに従ってセルフチェック
  - 単体テストは追加したか
  - jsdocsは書いたか
  - READMEは適宜更新したか etc...



In Partnership with intel

レビュー/マージ

- メンテナからレビューを受ける

  PRに対して指摘があれば適宜修正する

  早いとPR作成した当日にレビュー内容がコメントされる

- ソースを修正してPush

  修正に問題がなければメンテナがapproveする

  ※他の修正が並行で走っていてテストがコケる場合もあるのでその際は修正が必要

  →完了🎉

TIPS: 内部のツール（一部抜粋）

- Linter
  - ESLint
  - pkglint
  - awslint
  - prlint
- scripts/buildup
- cfn2ts

TIPS: 内部のツール（Linter）

- ESLint

  tools内の eslintrc.js の設定が全体に反映。GitpodでもESLint Pluginの追加で警告を出せる

- pkglint

  tools内の rules.ts の設定に応じて、ドキュメント形式 や License の整合性を検証

- awslint

  aws construct library用のlinter。設計ガイドラインに準拠しているかを検証

  (ExperimentalなパッケージだとJSDocがちゃんと付いてなくてwarnが出てることも)

- prlint(ベータ)

  PRの命名規則やコメントに応じて必要な項目が足りているのか検証

TIPS: 内部のツール（script/buildup）

- scripts/buildup
  パッケージのディレクトリ内で実行すると、関連するパッケージだけ再ビルドできる

  初回ビルドは大体1時間くらいかかってコケることがあるので使える

```
lerna success - cdk-build-tools
lerna success - cdk-integ-tools
lerna success - cfn2ts
lerna success - nodeunit-shim
lerna success - pkglint
lerna success - pkgtools
lerna success - prlint
lerna success - yarn-cling

real    46m36.159s
user    93m8.947s
sys     5m19.583s
Done in 2843.95s.
```

実際に初回起動した際の結果

TIPS: 内部のツール（cfn2ts）

- CloudFormationリソースからTSのコードを生成できるツール

  ***.generated.tsファイルはこのツールで生成されている。

  L1 Constructの対応だけやけに早いのはこのツールのおかげ（Chatbotなど）

- ビルド時に jsonファイル から TSファイルを生成

  @aws-cdk/cfnspec 配下にビルドの際生成される specification.json を見ている？

aws DEV DAY ONLINE JAPAN

実際コントリビュートした際の流れ

In Partnership with intel

実際コントリビュートした際の流れ

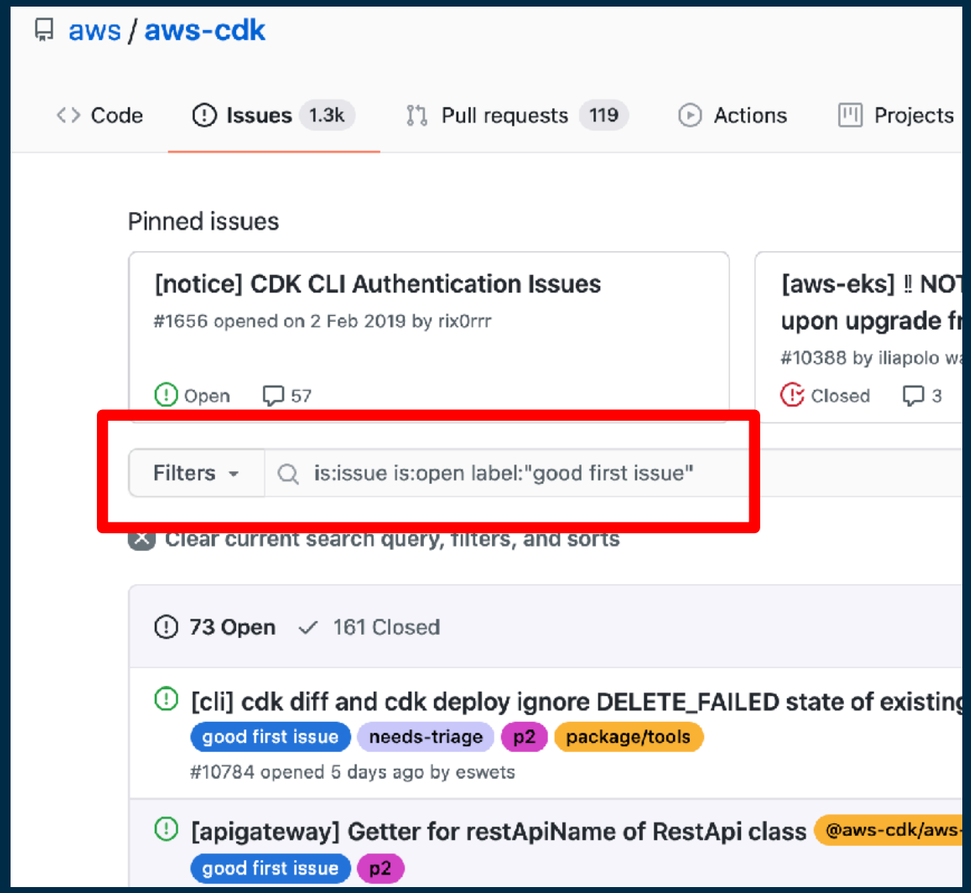
- issueを探す/コメントする
- Fork/Gitpod設定
- 実装
- 単体/結合テスト作成
- Branch作成/Push
- Pull Request作成
- レビュー/修正
- メンテナがマージ

※あくまで参考

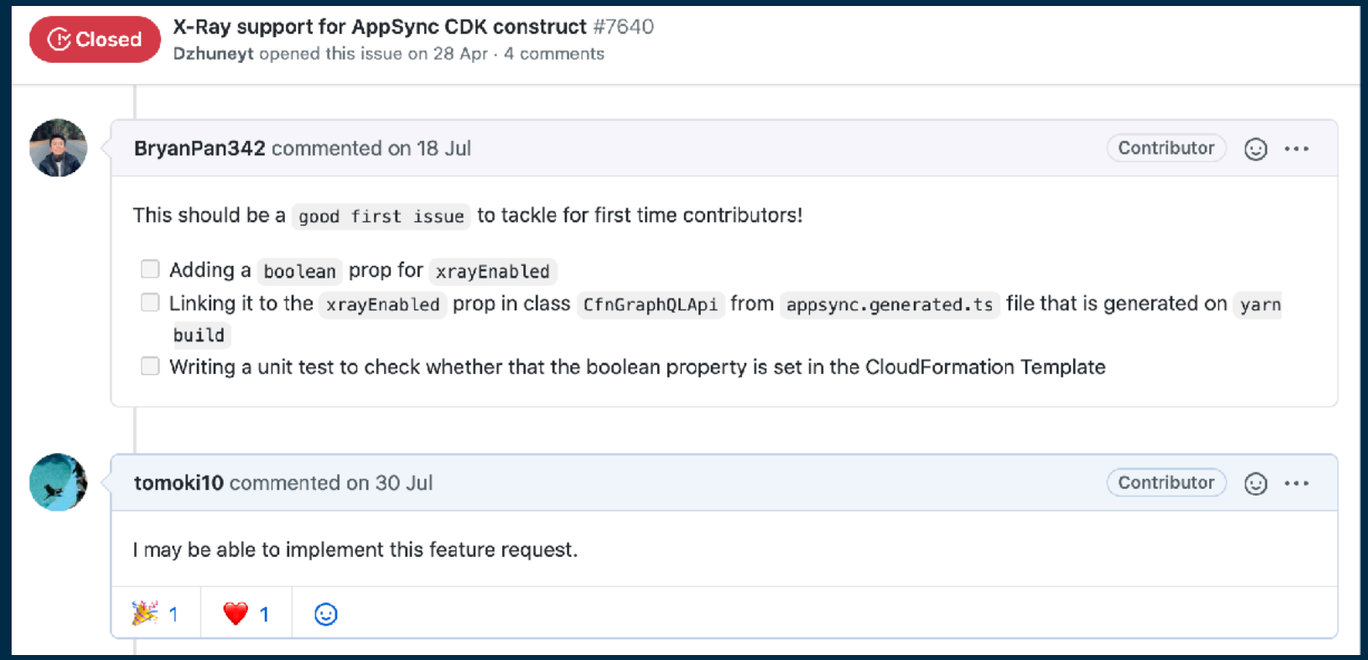
この通りでなくても大丈夫です

issueを探す/コメントする

- issueを探す



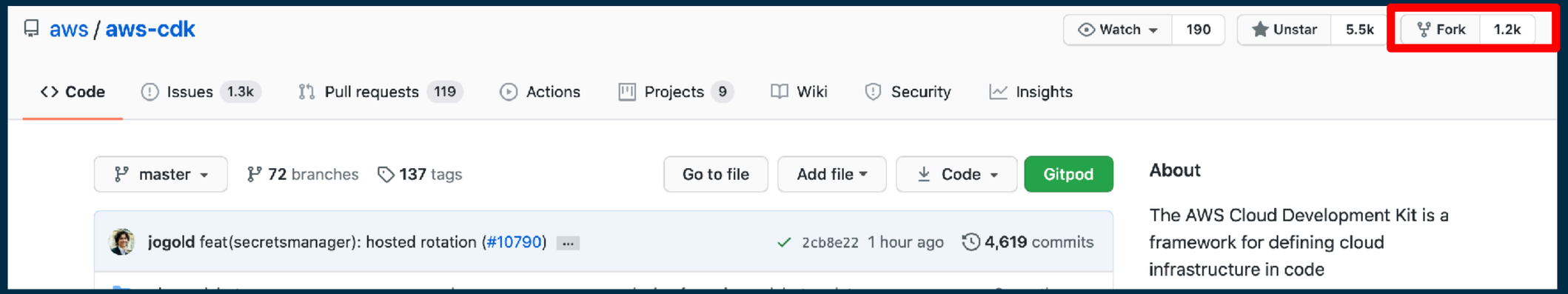
- issueにコメントする



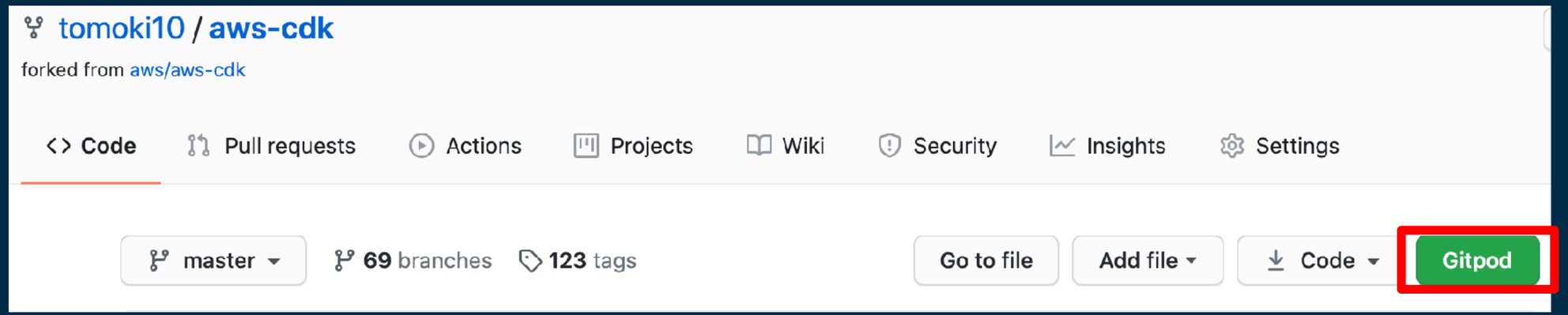
In Partnership with intel

Fork/Gitpod設定

- リポジトリからForkする



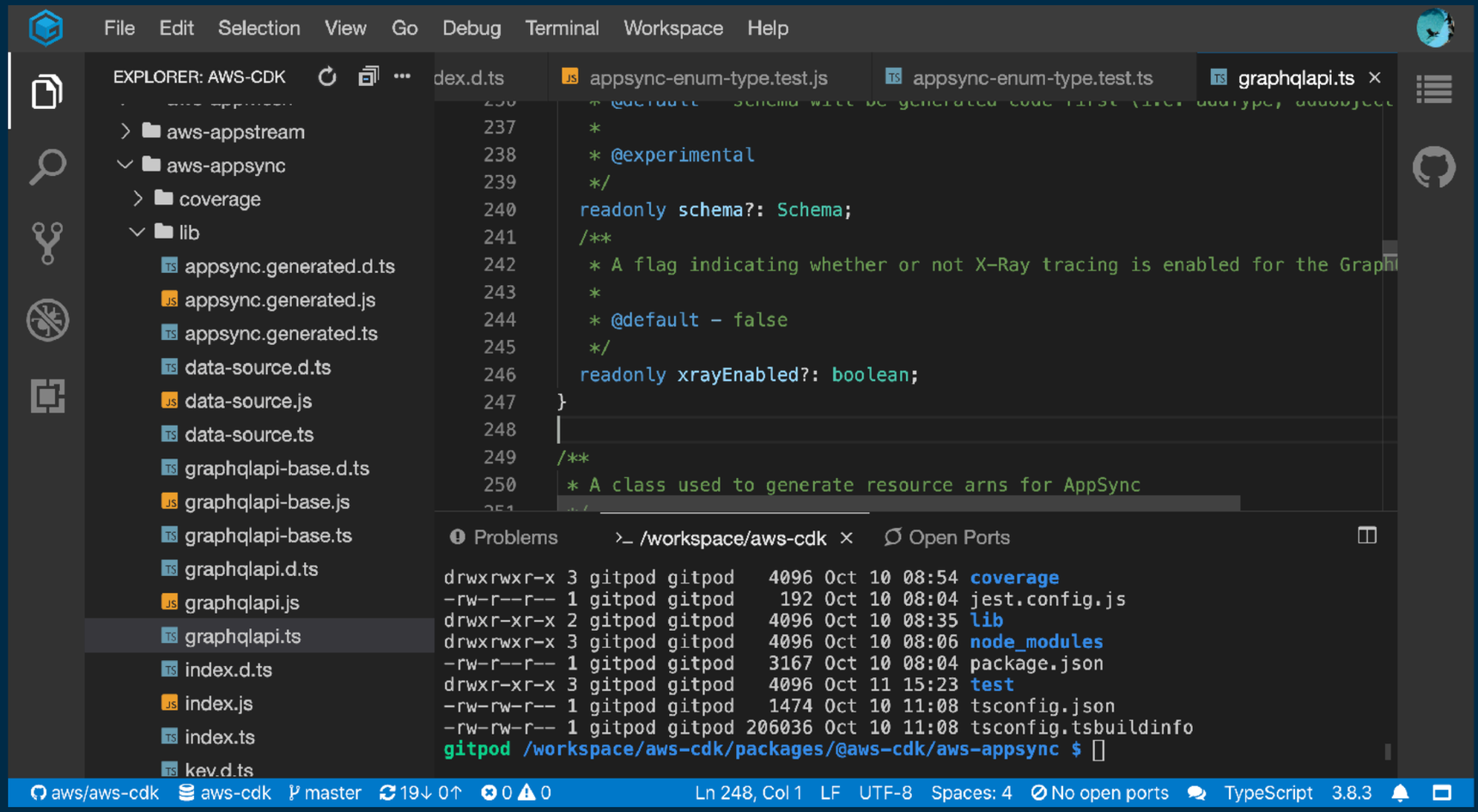
- ForkしたリポジトリでGitpodを起動



In Partnership with intel

実装

- Gitpod上でソースを確認して実装



In Partnership with intel

単体/結合テスト

- package.jsonを確認してテスト（ yarn build+test の実行結果 ）

  動作がわからない時は、意図的にテストを壊して影響範囲を確認

- 個人用のAWS環境で動作確認

```
>_ gitpod /workspace/aws-cdk/packages/@aws-cdk/aws-appsync ×

------------------------------ Coverage summary ------------------------------
Statements   : 85.8% ( 435/507 )
Branches     : 91.14% ( 391/429 )
Functions    : 77.36% ( 164/212 )
Lines        : 86.38% ( 406/470 )
------------------------------------------------------------------------------

Test Suites: 16 passed, 16 total
Tests:       172 passed, 172 total
Snapshots:   0 total
Time:        18.965 s
Ran all test suites.
Verifying integ.api-import.js against integ.api-import.expected.json ... OK.
Verifying integ.auth-apikey.js against integ.auth-apikey.expected.json ... OK.
Verifying integ.graphql-iam.js against integ.graphql-iam.expected.json ... OK.
Verifying integ.graphql-schema.js against integ.graphql-schema.expected.json ... OK.
Verifying integ.graphql.js against integ.graphql.expected.json ... OK.
Tests successful. Total time (24.2s) | /workspace/aws-cdk/node_modules/jest/bin/jest
.js (20.2s) | cdk-integ-assert (4.0s)
Done in 49.15s.
gitpod /workspace/aws-cdk/packages/@aws-cdk/aws-appsync $
```

Brunch作成/Push

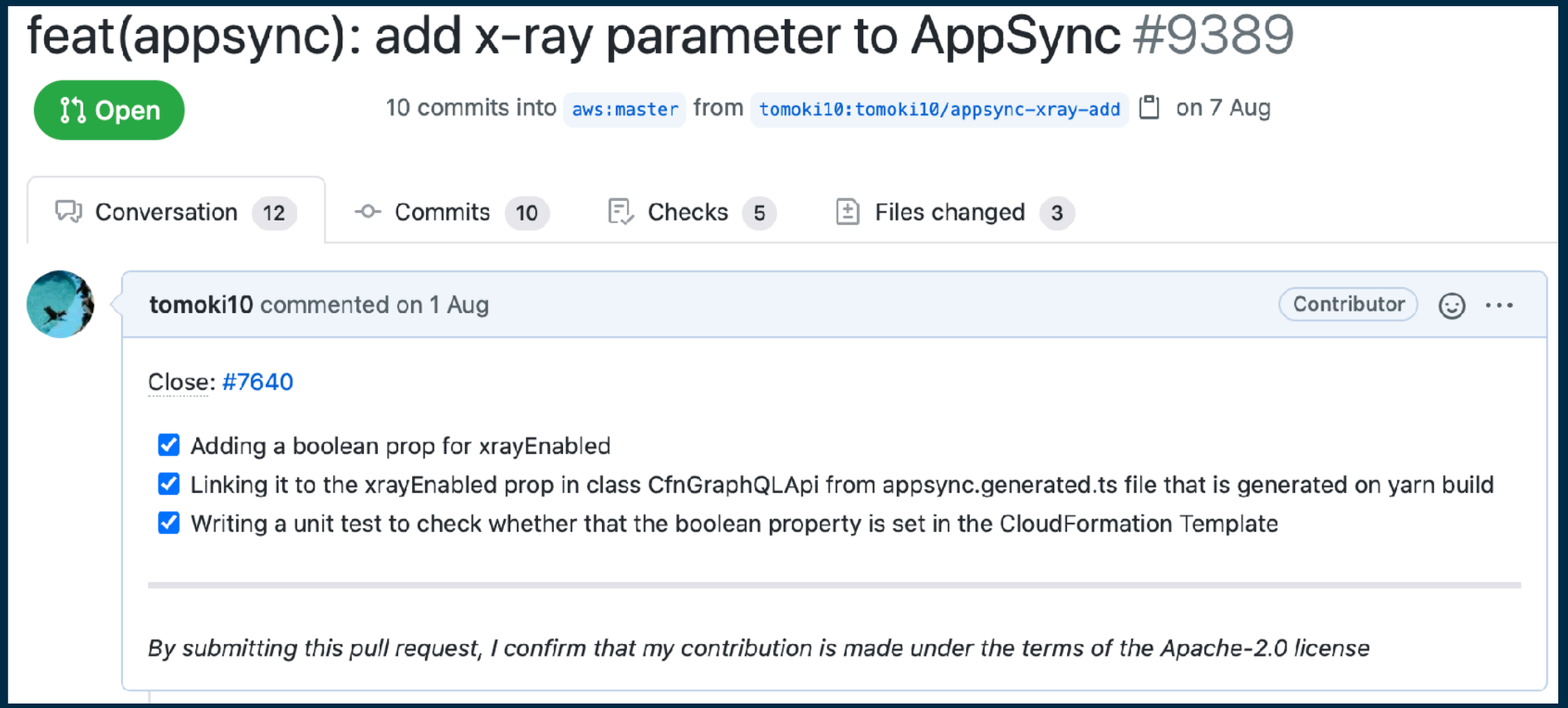
- GitpodでBrunch作成してcommitしてPush

  Gitを使用した普段の開発と特に変わらないです（Gitpodの権限だけは注意！）

```
Verifying integ.api-import.js against integ.api-import.expected.json ... OK.
Verifying integ.auth-apikey.js against integ.auth-apikey.expected.json ... OK.
Verifying integ.graphql-iam.js against integ.graphql-iam.expected.json ... OK.
Verifying integ.graphql-schema.js against integ.graphql-schema.expected.json ... OK.
Verifying integ.graphql.js against integ.graphql.expected.json ... OK.
Tests successful. Total time (24.2s) | /workspace/aws-cdk/node_modules/jest/bin/jest
.js (20.2s) | cdk-integ-assert (4.0s)
Done in 49.15s.
gitpod /workspace/aws-cdk/packages/@aws-cdk/aws-appsync $ git checkout -b tomoki10/a
ppsync-xray-add
```

Pull Request作成

- GitHub上でPull Requestを作成

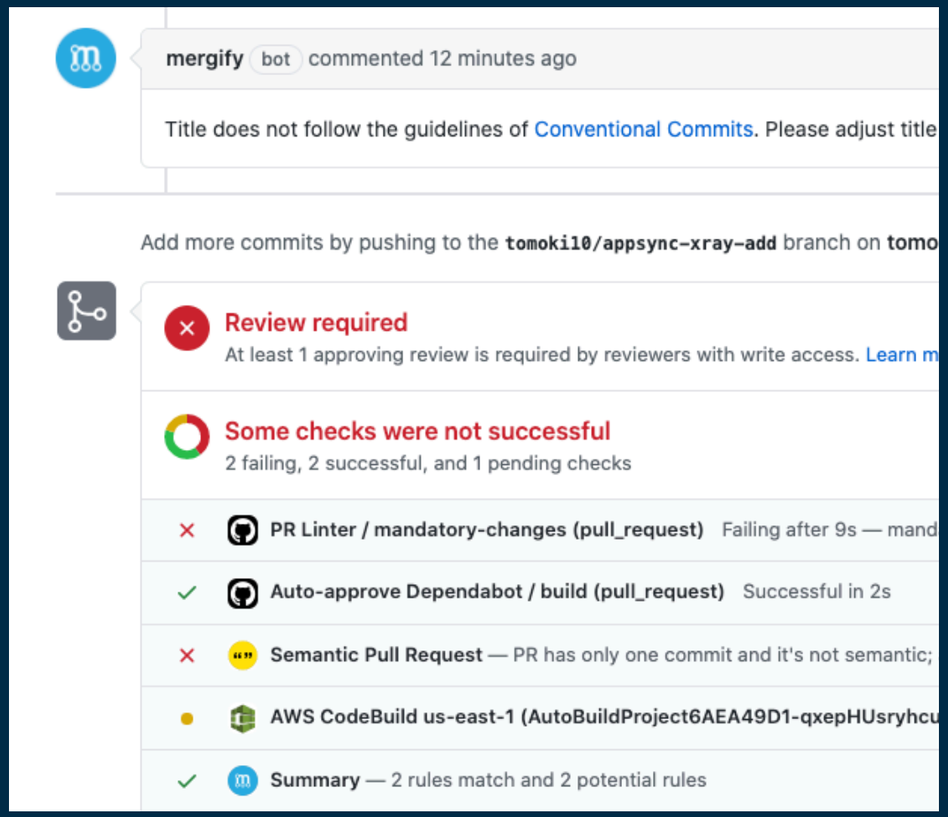


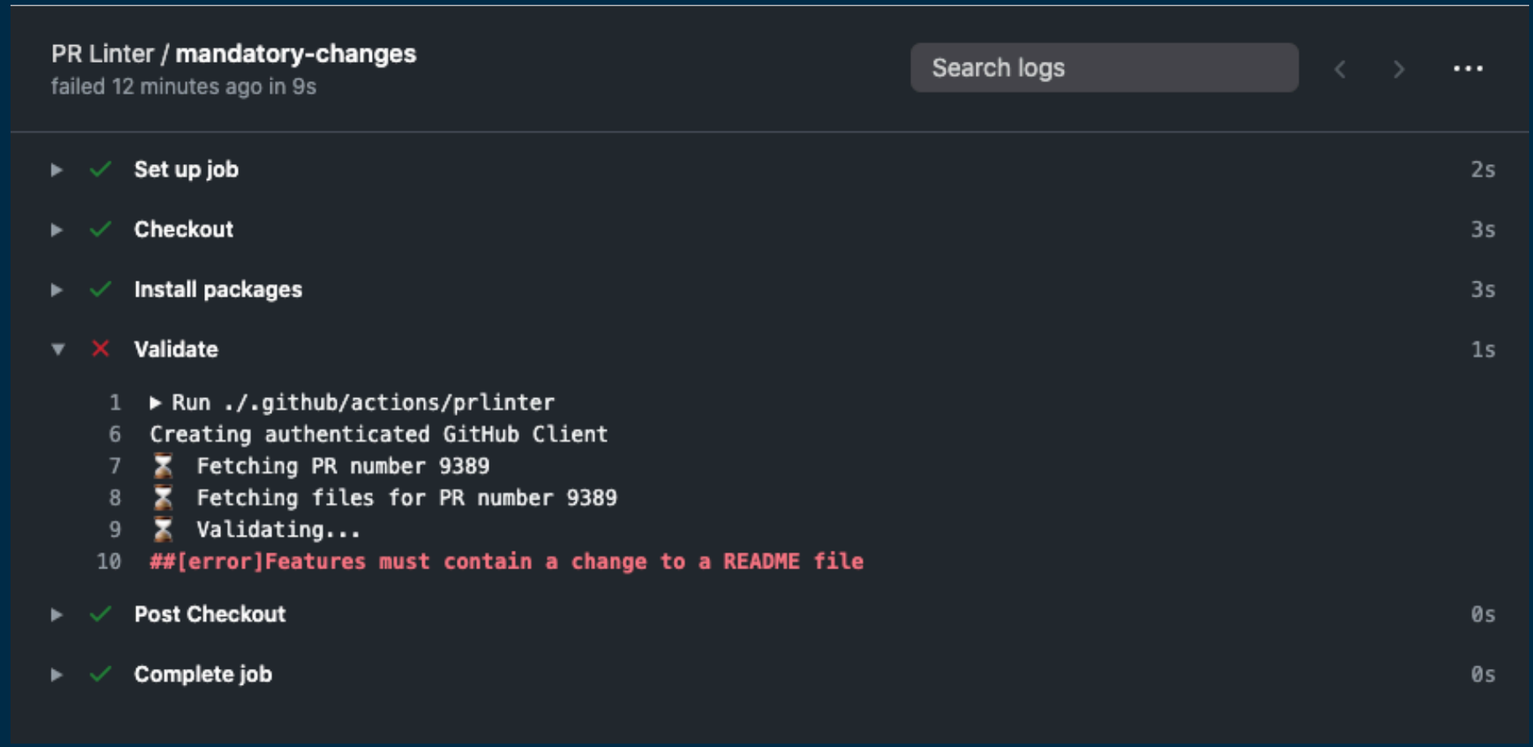
In Partnership with intel

mergify bot のlintで弾かれる

- 以下の原因でリジェクト

  1コミットでPR作成、機能追加なのにREADME変更なし

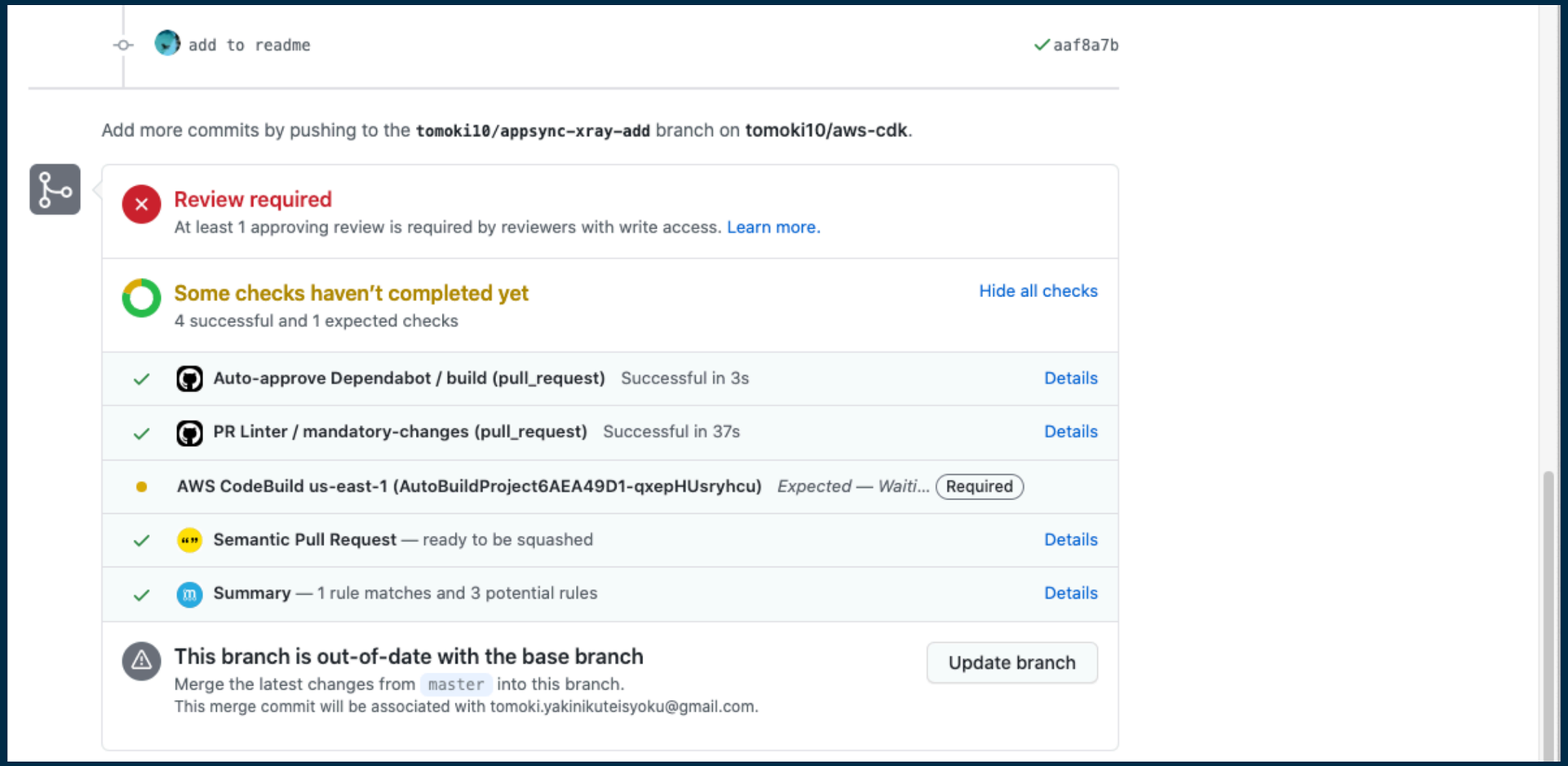




In Partnership with intel

merging bot のlintなどで弾かれる

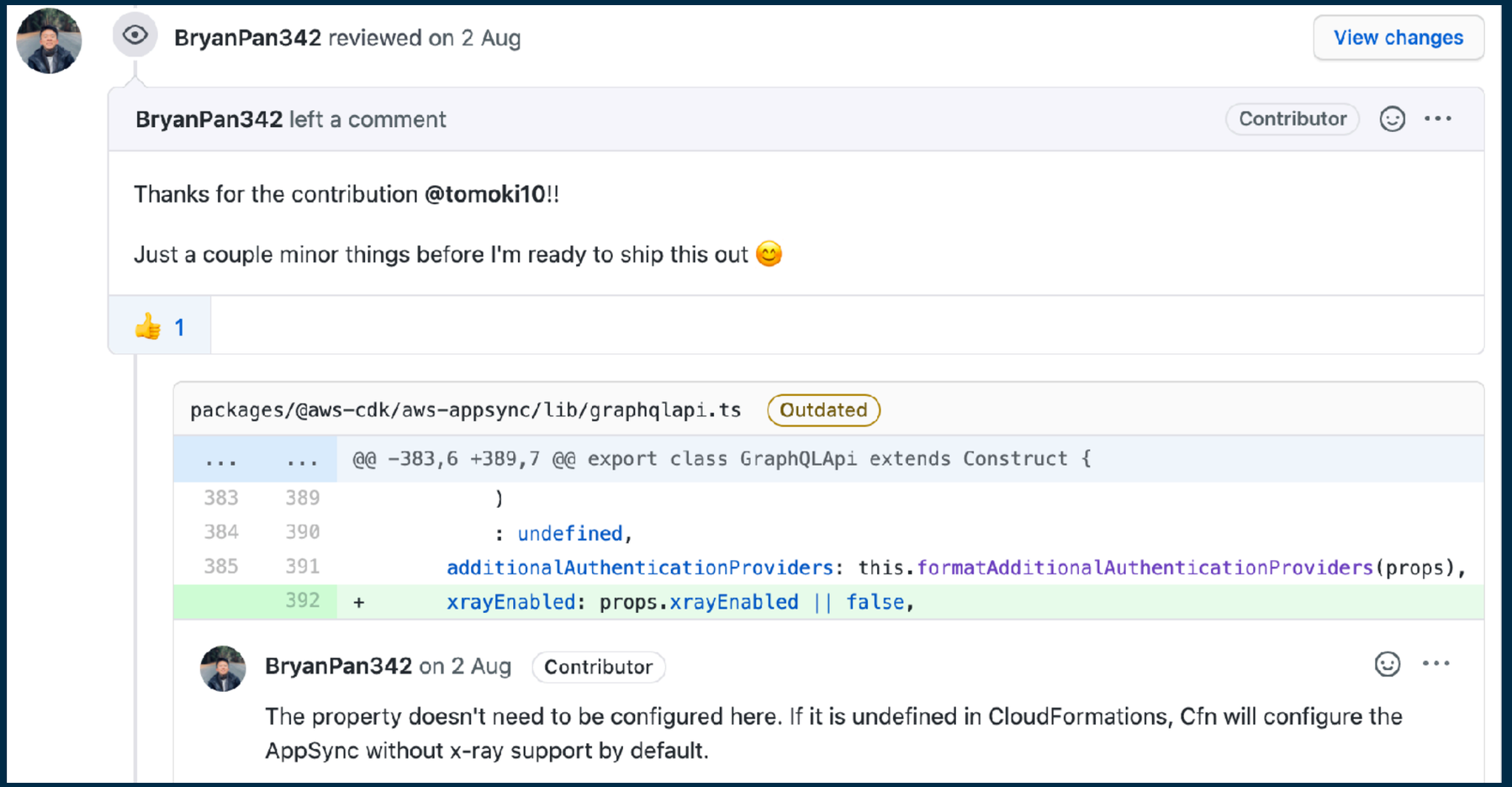
- エラーの原因を修正して再度Pushすれば完了



In Partnership with intel

レビュー/修正

- メンテナの方から指摘が来るので修正



In Partnership with intel

メンテナがマージ

- メンテナの方1人以上からapproveされればマージ🎉

BryanPan342 left a comment — Contributor

LGTM 🎉

Thanks for the contribution **@tomoki10**!!

• • •

BryanPan342 approved these changes on 7 Aug — View changes

• • •

mergify bot commented on 7 Aug — Contributor

Thank you for contributing! Your pull request will be updated from master and then merged automatically (do not update manually, and be sure to allow changes to be pushed to your fork).

mergify bot merged commit **51921ad** into `aws:master` on 7 Aug — View details — Revert

6 of 7 checks passed

aws DEV DAY
ONLINE JAPAN

まとめ

In Partnership with intel

まとめ

- AWS CDKの構成と詳細を紹介

  CDKはAWSに留まらず発展する可能性がある！ 乗るしかない、このビッグウェーブに！

- コントリビューティングガイドの説明

  やり方はコントリビューティングガイドに全て書いてある。読もう！

- 実際コントリビュートした際の流れの紹介

  失敗した内容も含めてコントリビュートまでの全体像を紹介

→明日からあなたもAWS CDKのコントリビューターになろう！！！

aws DEV DAY
ONLINE JAPAN

Thank you!

Tomoki Sato
sato.tomoki@classmethod.jp

In Partnership with intel