作って学ぶ！サーバレスアプリ開発

AWS CDK を活用したサーバレスアーキテクチャの実装例

プロジェクト概要

シンプルなタスク管理アプリを AWS CDK を使用してサーバレスアーキテクチャで実装

- **完全サーバレス構成**: インフラ管理やサーバーメンテナンスが不要
- **AWS CDK によるインフラのコード化**: インフラを TypeScript で定義
- **CI/CD 自動化**: GitHub Actions による自動デプロイ
- **シンプルなフロントエンド**: 直感的に操作できる UI

技術スタック

概要

分野	使用技術
インフラ	AWS CDK (IaC), AWS Lambda, API Gateway, DynamoDB, S3, CloudFront
開発環境	VSCode（DevContainer）, Docker, GitHub
言語・ライブラリ	TypeScript, AWS SDK
CI/CD	GitHub Actions

技術スタック

バックエンド

- **DynamoDB**: タスクデータの永続化
- **Lambda 関数**: サーバレス API エンドポイント
- **API Gateway**: RESTful API インターフェース

フロントエンド

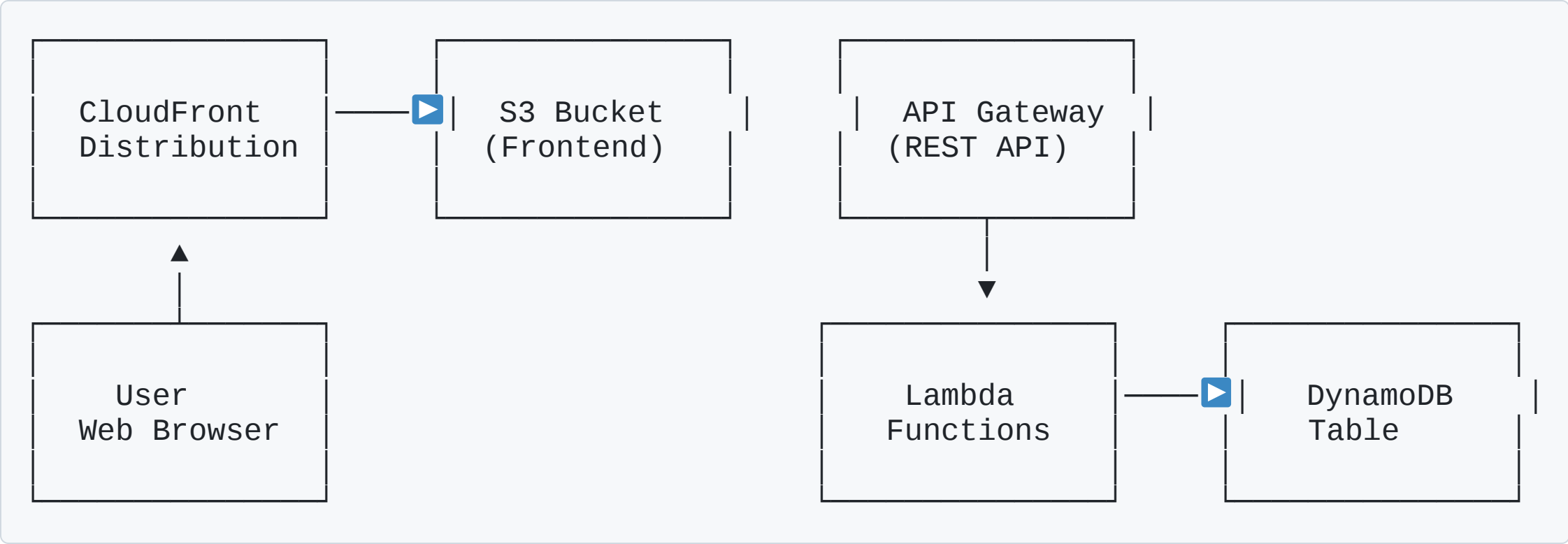
- HTML/CSS/JavaScript
- S3 でホスティング
- CloudFront で配信

技術スタック

インフラ

- **AWS CDK**: TypeScript でインフラ定義
- **GitHub Actions**: CI/CD 自動化

アーキテクチャ図


CloudFront Distribution
S3 Bucket (Frontend)
API Gateway (REST API)
User Web Browser
Lambda Functions
DynamoDB Table

アーキテクチャ図

開発の流れと工夫ポイント

- 開発環境
  - DevContainer で環境を統一
- インフラ構築
  - CDK スタックを分割（DB, API, フロント）
  - コードでインフラを管理（再現性向上）
- CI/CD
  - main ブランチにプッシュするだけで自動デプロイ
  - Lambda コード変更時もすぐ反映

CDK のスタック構成

プロジェクトは 3 つの主要なスタックで構成：

```
// infra/bin/main.ts
const databaseStack = new TaskDatabaseStack(app, "TaskDatabaseStack");
const apiStack = new TaskApiStack(app, "TaskApiStack", { databaseStack });
const frontendStack = new TaskFrontendStack(app, "TaskFrontendStack");
```

TaskDatabaseStack

DynamoDB テーブルの定義：

```
// infra/lib/task-database-stack.ts
export class TaskDatabaseStack extends cdk.Stack {
  public readonly taskTable: dynamodb.Table;

  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    this.taskTable = new dynamodb.Table(this, "TaskTable", {
      partitionKey: { name: "taskId", type: dynamodb.AttributeType.STRING },
      removalPolicy: cdk.RemovalPolicy.DESTROY,
    });
  }
}
```

TaskApiStack

Lambda 関数と API Gateway の定義：

```
// API Gatewayを作成
const api = new apigateway.RestApi(this, "TaskApi", {
  restApiName: "Task Management API",
  description: "This service handles task operations.",
  defaultCorsPreflightOptions: {
    allowOrigins: apigateway.Cors.ALL_ORIGINS,
    allowMethods: apigateway.Cors.ALL_METHODS,
  },
});
```

```
const tasksResource = api.root.addResource("tasks");
tasksResource.addMethod(
  "POST",
  new apigateway.LambdaIntegration(createTaskLambda)
);
tasksResource.addMethod(
  "GET",
  new apigateway.LambdaIntegration(listTasksLambda)
);

const taskResource = tasksResource.addResource("{taskId}");
taskResource.addMethod(
  "PATCH",
  new apigateway.LambdaIntegration(updateTaskLambda)
);
taskResource.addMethod(
  "DELETE",
  new apigateway.LambdaIntegration(deleteTaskLambda)
);
```

TaskFrontendStack

S3 と CloudFront の設定：

```
// S3バケットを作成
const frontendBucket = new s3.Bucket(this, "FrontendBucket", {
  bucketName: "haji-task-manager-frontend-bucket",
  publicReadAccess: false,
  blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
  websiteIndexDocument: "index.html",
  removalPolicy: cdk.RemovalPolicy.DESTROY,
});
```

```
// CloudFrontディストリビューションを作成
const distribution = new cloudfront.Distribution(
  this,
  "Haji-FrontendDistribution",
  {
    defaultRootObject: "index.html",
    defaultBehavior: {
      origin:
        cdk.aws_cloudfront_origins.S3BucketOrigin.withOriginAccessControl(
          frontendBucket
        ),
      viewerProtocolPolicy: cloudfront.ViewerProtocolPolicy.REDIRECT_TO_HTTPS,
      cachePolicy: cloudfront.CachePolicy.CACHING_DISABLED,
    },
  }
);
```

Lambda 関数の実装

例：タスク作成機能

```
// lambda/src/create-task.ts
export const handler = async (event: any) => {
  try {
    const body = JSON.parse(event.body);
    const taskId = generateUuid();
    const createdAt = new Date().toISOString();

    const taskItem = {
      taskId: { S: taskId },
      title: { S: body.title },
      description: { S: body.description || "" },
      createdAt: { S: createdAt },
    };
```

```
    await dynamoDb.send(
      new PutItemCommand({
        TableName: tableName,
        Item: taskItem,
      })
    );

    return { statusCode: 201 /* 省略 */ };
  } catch (error) {
    // エラー処理
  }
};
```

フロントエンドの実装

シンプルな HTML と CSS：

```
<div class="container">
  <h1>Task Manager</h1>
  <div>
    <input id="task-input" type="text" placeholder="Enter a new task" />
    <button id="add-task-btn">Add Task</button>
  </div>
  <ul id="task-list"></ul>
</div>
```

```css
body {
  font-family: Arial, sans-serif;
  padding: 20px;
  background-color: #f4f4f9;
}
.container {
  max-width: 600px;
  margin: 0 auto;
  background: white;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
```

CI/CD パイプライン

GitHub Actions による自動デプロイ：

```yaml
# .github/workflows/deploy.yml
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Build Lambda
        working-directory: lambda
        run: |
          npm install
          npm run build
```

```
- name: Deploy CDK stack
  working-directory: infra
  run: |
    npx cdk deploy --require-approval never
```

開発環境の構築

DevContainer による一貫した開発環境：

```
# .devcontainer/Dockerfile
FROM debian:bookworm-slim

# バージョンの指定
ARG PYTHON_VERSION=3.11
ARG NODEJS_VERSION=22.11.0
ARG AWS_CDK_VERSION=2.177.0
# ...

# AWS CLIのインストール
RUN case ${TARGETARCH} in \
        "amd64") \
            AWS_CLI_URL="https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" ;; \
        "arm64") \
            AWS_CLI_URL="https://awscli.amazonaws.com/awscli-exe-linux-aarch64.zip" ;; \
        *) \
            echo "Unsupported architecture: ${TARGETARCH}" >&2; \
            exit 1 ;; \
    esac && \
    # ...
```

今後の改善点

- ユーザー認証の追加
- タスクの優先度やカテゴリ分類の実装
- タスク完了の通知機能
- モバイルレスポンシブデザインの強化

雑感

- 本資料に記載のコードと本資料は 99% AI 製（ChatGPT+Claude）。
- つらみポイント
  - フロントエンド、フレームワークのバージョンアップが早過ぎて AI が追いついてない
  - それぞれの互換性を意識した環境構築、適切なコード…
  - エラーの対処をしたら別のエラー → ループ
  - 頭使わなさすぎて馬鹿になりそう

雑感

- よかったポイント
  - DevContainer や Git/GitHub といった開発ツールの活用
    - めっちゃ便利。
- 他
  - テストやセキュリティでの品質保証は依然として必要だろう

資料

- 成果物（GitHub リポジトリ）
- おまけ