# サーバレスタスク管理アプリケーション

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

## 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

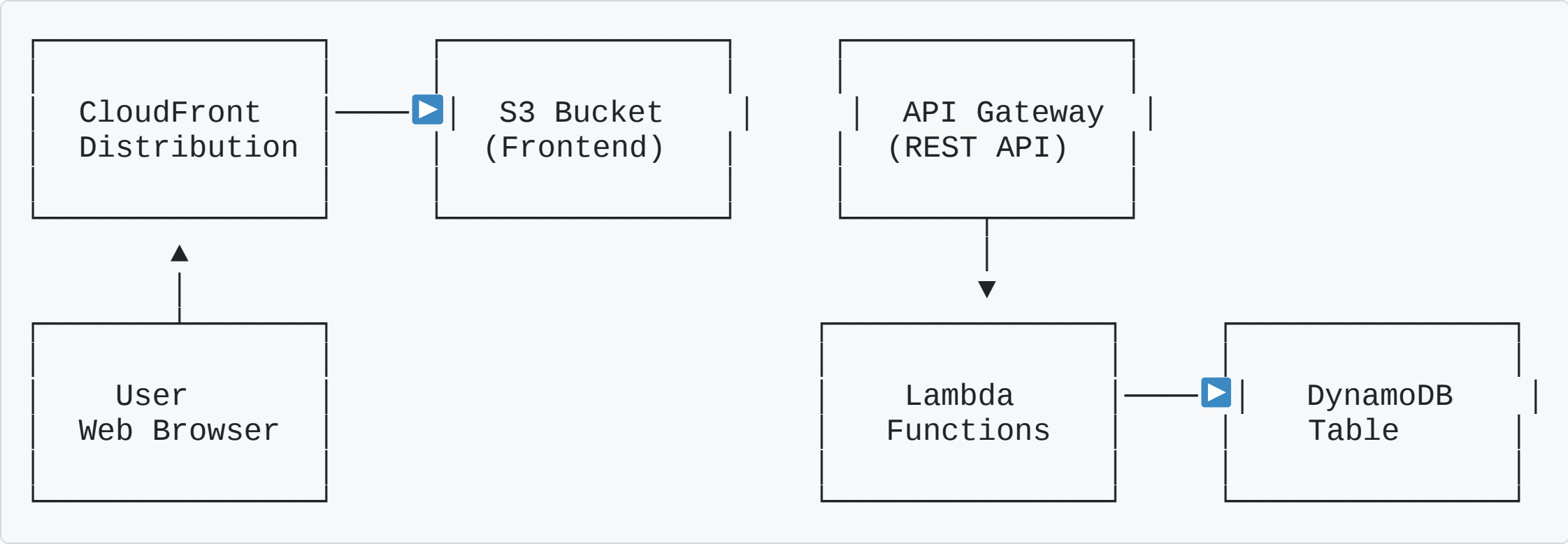
## フロントエンド

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

## インフラ

- **AWS CDK**: TypeScriptでインフラ定義

# アーキテクチャ図


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

# 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>
```

```
body {
  font-family: Arial, sans-serif;
  padding: 20px;
  background-color: #f4f4f9;
}
.container {
  max-width: 600px;
  margin: 0 auto;
```

# 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: |
```

# 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 && \
    # ...
```

# デモ

[ここでアプリケーションのデモを行います]

# 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

# ご質問・フィードバック

ありがとうございました！