

中村勉強会
～AWSインフラ編～
Codeサービス系

#2 組み立て工場見学!ざっくり
とパイプラインを外観する

この勉強会の目標

- ・ CodePipelineがどのようにしてCI/CDフローを形成するかを理解する
- ・ CodePipelineの主要概念を理解する
- ・ マネコン/CDKでの実装方法を理解する

本日のお題目

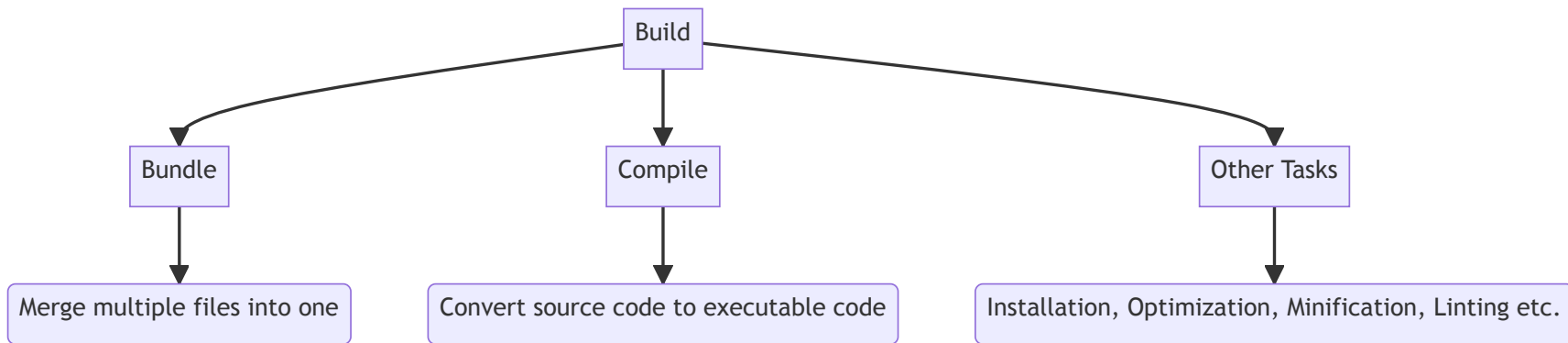
0. 軽く前回のおさらい
1. CodePipelineの役割
2. 工場見学①原料調達
3. 工場見学②加工場
4. 工場見学③配送業者
5. CodePipelineの主要概念
6. 主要概念を反映したCDKコード

0. 前回のおさらい

CI/CDとは？

①CI(Continuous Integration)

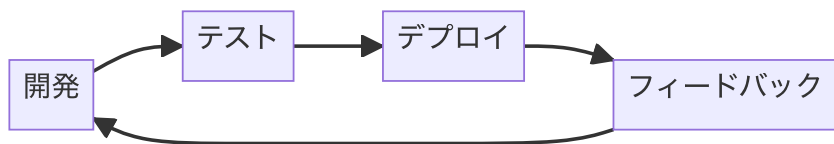
→ コードの変更を継続的に統合する過程(ビルド、テスト)



②CD(Continuous Deployment/Delivery)

→ 統合されたコードを実環境に継続的に反映させる過程(デプロイ)

高速なフィードバックループの実現



- ①アプリケーションの品質安定
- ②手動運用時に起こりうるヒューマンエラーの削減

だけではなく...

もう一つの幸せスパイラル



実はCI/CDツールはたくさんある

→CircleCI,Jenkins,GitLab,GitHub Actions, Azure,GCP系サービス

結論:AWSのサービス群を選択

1.信頼と実績

- ・すでにプロトが用意してあって、最低限のCI構築はできていた
- ・CodeDeployだけ一応使ったことがあった

2.応用可能性/親和性

- ・その他インフラリソースとの親和性
- ・IaCサービス(AWS Cloud Development Kit)との親和性

3.学習コスト

- ・学習コストのことだけ考えれば、基本的にサードパーティ製のツールはない方がいい

AWSのCode系サービス

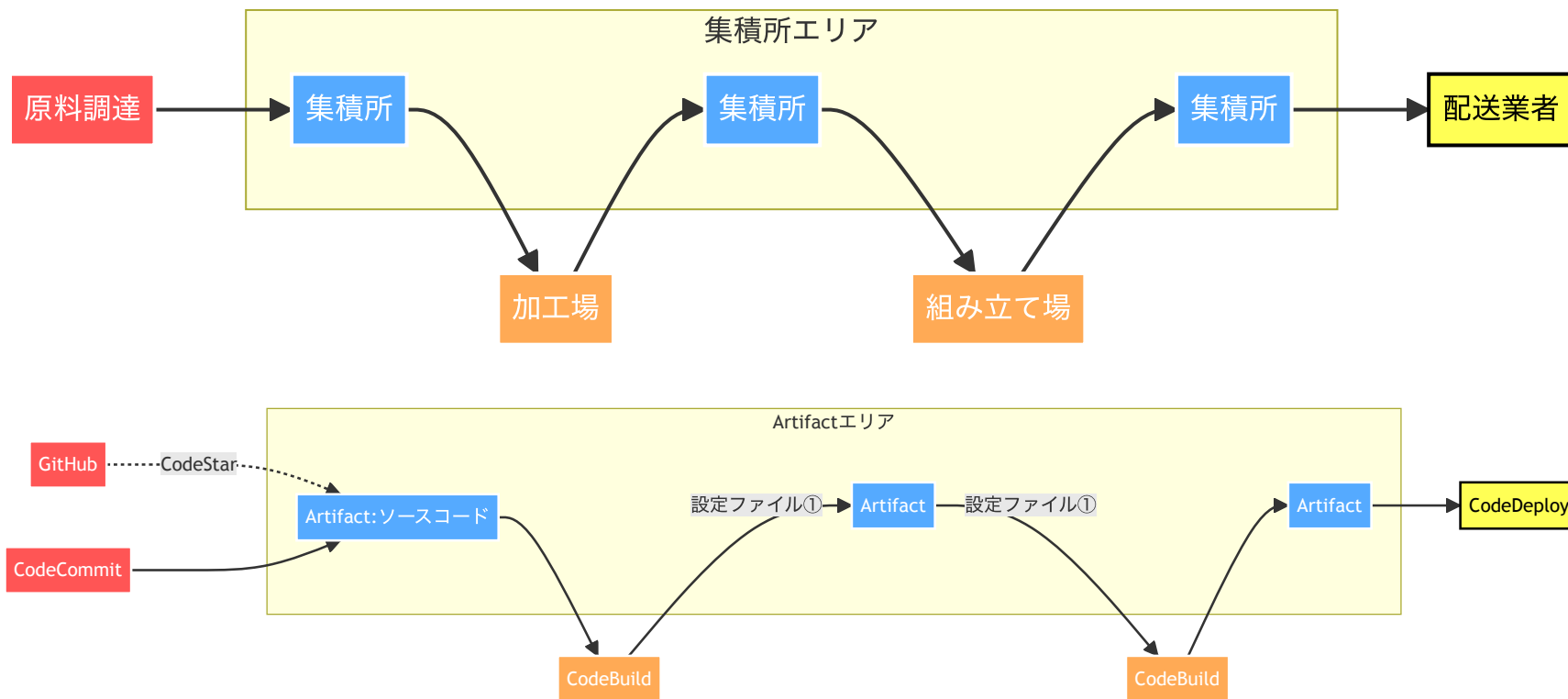
1. CodePipeline/Artifact...パイプライン(流れ)を定義、ソースストレージ
 2. CodeCommit...GitHubのAWS版
 3. CodeBuild...ビルドのための環境を素早く用意、ビルドプロセスの構築
 4. CodeDeploy...デプロイ
-

よりマネージドなCI/CDサービス

5. CodeGuru...リッチでニッチな使い道(機械学習使ったコードレビュー)
6. CodeStar...CI/CDめっちゃマネージド(テンプレートから選べる)
7. CodeCatalyst...メンバーオンボーディング、IDE連携、CodeStar+CDK構築まで一気通貫(マネージドの鬼)、コツを掴めば、爆速開発環境、インフラ、パイプライン構築可能かも

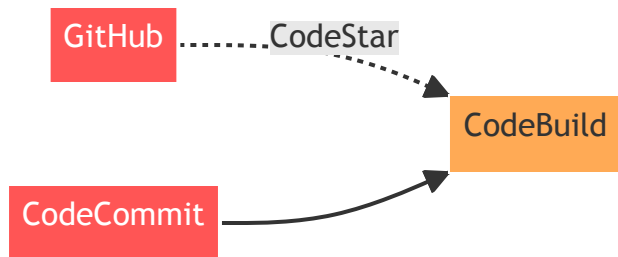
1.CodePipelineの役割

ある機械製品が出来上がるまでの流れ



2.工場見学①原料調達

原料調達



①接続先の設定

②外部プロバイダーを挟む場合、一工夫必要(GitHubとの連携にはCodeStarが簡単)

三

デベロッパー用ツール > CodePipeline > パイプライン > 新規のパイプラインを作成する

ソースステージを追加する 情報

Step 2 of 5

ソース

ソースプロバイダー

ここでは、パイプラインの入力アーティファクトを保存します。プロバイダ接続の詳細を指定します。

AWS CodeCommit

リポジトリ名

ソースコードをプッシュしたところで既に作成したリポジトリを選択します。

Q

ブランチ名

リポジトリのブランチを選択します

Q

検出オプションを変更する

検出モードを選択して、ソースコードに変更が発生したときにパイプラインを自動的に開始させます。

☒ Amazon CloudWatch Events (推奨)

Amazon CloudWatch Events を使用して、変更が発生したときにパイプラインを自動的に開始させます。

☐ AWS CodePipeline

AWS CodePipeline を使用して、変更を定期的に確認する

出力アーティファクト形式

出力アーティファクト形式を選択します。

☒ CodePipeline のデフォルト

AWS CodePipeline では、パイプラインのアーティファクトにデフォルトの zip 形式が使用されます。リポジトリに関する Git メタデータは含まれません。

☐ 完全クローン

AWS CodePipeline はリポジトリに関するメタデータを渡し、後続の操作で完全な Git クローンを作成できます。AWS CodeBuild の操作に対してのみサポートされます。

キャンセル

戻る

次に

CodeStar Connectionsを利用してGitHubと接続する

デベロッパー用ツール > CodePipeline > パイプライン > 新規のパイプラインを作成する

ソースステージを追加する [情報](#)

Step 2 of 5

ソース

ソースプロバイダー
ここでは、パイプラインの入力アーティファクトを保存します。プロバイダ接続の詳細を指定します。

GitHub (バージョン 2) ▼

新しい GitHub バージョン 2 (アプリーベース) アクション

CodePipeline で GitHub バージョン 2 アクションを追加するには、OAuth アプリを使用してリポジトリにアクセスする接続を作成します。以下のオプションを使用して、既存の接続を選択するか、新しい接続を作成してください。 [詳細](#)

接続
すでに設定済みの既存の接続を選択するか、新しい接続を作成してこのタスクに戻ります。

arn:aws:codestar-connections:ap-northeast-1:242325799897:conne... または **GitHub に接続する**

接続する準備が完了しました
GitHub 接続を使用する準備ができました。

リポジトリ名
GitHub アカウントのリポジトリを選択します。

Scala-partners/facility-reservation ✕

<account>/<repository-name>

ブランチ名
リポジトリのブランチを選択します。

dev ✕

検出オプションを変更する

☒ ソースコードの変更時にパイプラインを開始する

①Sourceステージでトリガー先の設定をする

②CodeStar Connectionsで取得した接続ARN(後述)を入力

③トリガー先リポジトリ、ブランチを選択

CodeStar Connectionsの接続を作成する

①接続の作成

デベロッパー用ツール ×
設定

- ▶ ソース • CodeCommit
- ▶ アーティファクト • CodeArtifact
- ▶ ビルド • CodeBuild
- ▶ デプロイ • CodeDeploy
- ▶ パイプライン • CodePipeline
- ▼ 設定
 - 通知ルール
 - 接続**
- 🔍 リソースへ移動する
- 📧 フィードバック

デベロッパー用ツール > 接続

接続 | ホスト

接続 情報

🔄 詳細を表示 保留中の接続を更新 削除

接続を作成

🔍

< 1 > ⚙️

	接続名	プロバイダー	ステータス	
<input type="radio"/>	facility-reservation-app-repo	GitHub	🟢 利用可能	a 1 a

②GitHubを選択し、接続名を入力

デベロッパー用ツール > 接続 > 接続を作成

接続を作成する 情報

プロバイダーを選択する

☐ Bitbucket ☒ GitHub ☐ GitHub Enterprise Server

☐ GitLab

GitHub アプリ接続を作成する 情報

接続名

SCP_Organizatin

▶ タグ - オプション

GitHub に接続する

使用する認証アプリの設定

③ どのアカウントにインストールされたアプリを使うか選択

④ 新しいアプリをインストールを選択すると設定にいける

デベロッパー用ツール > 接続 > 接続を作成

GitHub に接続する

GitHub 接続設定 情報

接続名

GitHub アプリ

GitHub アプリは、GitHub との間に接続リンクを作成します。開始するには、新しいアプリをインストールし、この接続を保存します。

または **新しいアプリをインストールする**

yutaro0602
38208028

Scala-partners
12882003

新しく監視対象のリポジトリを追加したい

既存の設定を利用

接続

aws

Install AWS Connector for GitHub

Where do you want to install AWS Connector for GitHub?

yutaro0602

Configure >

Scala-partners


Configure >

Terms Privacy Security Status Docs Contact GitHub Pricing API Training Blog About


© 2023 GitHub, Inc.

設定確認

⑤ パスワード入力



Confirm access

 Signed in as @yutaro0602

Password

.....

[Forgot password?](#)

Confirm

Tip: You are entering **sudo mode**. After you've performed a sudo-protected action, you'll only be asked to re-authenticate again after a few hours of inactivity.

[Terms](#) [Privacy](#) [Docs](#) [Contact GitHub Support](#)

⑥ インストールアプリの設定変更

Overview

Repositories


Projects

Packages

Teams

People

Settings

 **Scala-partners**
Organization account [Switch to another account](#)

General

Features

Access

Billing and plans

Repository roles

Member privileges

Import/Export

Moderation

Code, planning, and automation

Repository

Codespaces

Copilot

Actions

Webhooks

Discussions

Packages

Pages


Projects

Security

Authentication security

Code security and analysis

Verified and approved domains

 **AWS Connector for GitHub**
Installed 3 years ago · Developed by aws
<https://docs.aws.amazon.com/dtconsole/latest/userguide/welcome-connections.html>

Enables you to connect GitHub with AWS

Permissions

Repository access

⚠️ アカウント単位の設定であることに注意

Go to your organization profile

☒ **Only select repositories**
Select at least one repository.
Also includes public repositories (read-only).

Select repositories

Selected 7 repositories.

Scala-partners/pytorch-YOLOv4

Scala-partners/lpsystem

対象リポジトリの追加とARNの確認

⑤ 対象リポジトリの追加

Access

- Billing and plans
- Repository roles
- Member privileges
- Import/Export
- Moderation
- Code, planning, and automation
 - Repository
 - Codespaces
 - Copilot
 - Actions
 - Webhooks
 - Discussions
 - Packages
 - Pages
 - Projects
- Security
 - Authentication security
 - Code security and analysis
 - Verified and approved domains
 - Secrets and variables
- Third-party Access
 - OAuth application policy
 - GitHub Apps
 - Personal access tokens
- Integrations
 - Scheduled reminders

Enables you to connect GitHub with AWS

Permissions

- ✓ Read access to issues, members, and metadata
- ✓ Read and write access to administration, code, commit statuses, and pull requests

Repository access

☐ All repositories
This applies to all current and future repositories owned by the resource owner. Also includes public repositories (read-only).

☒ Only select repositories
Select at least one repository.
Also includes public repositories (read-only).

Select repositories

Search for a repository

- Scala-partners/amplify-docker-action
no description
- Scala-partners/amplify-prediction
no description
- Scala-partners/AmplifyBBSHandsOn
no description
- Scala-partners/amplify_test
no description
- Scala-partners/AppSyncDataInsertTool
no description
- Scala-partners/aws-secrets-test
no description
- Scala-partners/cloudformation-handson
no description

監視対象に入れたいリポジトリの追加

⑥ ARN確認

デベロッパー用ツール > 接続

接続 | ホスト

接続 情報

接続名 | プロバイダ | ステータス | ARN

facility-reservation-app-repo	GitHub	利用可能	arn:aws:codestar-connections:ap-northeast-1:242325799897:connection/
-------------------------------	--------	------	--

3.工場見学②加工場

加工場(CodeBuild)

Artifact : ソースコード → CodeBuild → Artifact : 設定ファイル、成果物

①ビルド、テスト環境の設定

②CodeBuildではビルドごとにプロジェクトというものを作成する

③ビルド環境に入れておく環境変数の設定

デベロッパー用ツール > CodePipeline > パイプライン > 新規のパイプラインを作成する

ビルドステージを追加する 情報

Step 1
パイプラインの設定を選択する

Step 2
ソースステージを追加する

Step 3
ビルドステージを追加する

Step 4
デプロイステージを追加する

Step 5
レビュー

構築する - オプション

プロバイダーを構築する
これはビルドプロジェクトのツールです。オペレーティングシステム、ビルドスペックファイル、および出力ファイル名のようなビルドアーティファクトの詳細を提供してください。

AWS CodeBuild ▼

リージョン
アジアパシフィック (東京) ▼

プロジェクト名
AWS CodeBuild コンソールで既に作成したビルドプロジェクトを選択します。または AWS CodeBuild コンソールでビルドプロジェクトを作成してこのタスクに戻ります。

Q test-dev-project X または プロジェクトを作成する [🔗](#)

環境変数 - オプション
CodeBuild 環境変数のキー、値、タイプを選択します。[value] フィールドで、CodePipeline によって生成された変数を参照できます。 [詳細](#) [🔗](#)

環境変数の追加

ビルドタイプ

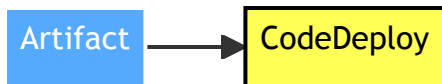
☒ 単一ビルド
単一ビルドをトリガー

☐ バッチビルド
複数のビルドを 1 つの実行としてトリガーします。

キャンセル 戻る ビルドステージをスキップ 次に

4.工場見学③配送業者

配送業者(CodeDeploy)



①特定のプロバイダーを選択すると配送業者として選択可能

②デプロイ先(お届け先)の設定、デプロイ内容(お届け物)の設定

③お届け先や中身によって設定や配送形態が結構変わる

ソースステージを追加する

Step 3
ビルドステージを追加する

Step 4
デプロイステージを追加する

Step 5
レビュー

デプロイプロバイダー
インスタンスにデプロイする方法を選択します。プロバイダを選択し、プロバイダの設定の詳細を入力します。

Amazon ECS(ブルー/グリーン) ▼

リージョン
アジアパシフィック (東京) ▼

AWS CodeDeploy アプリケーション名
既存のアプリケーションを一つ選択します。または AWS CodeDeploy に新たに作成します。

アプリケーションを作成する

AWS CodeDeploy デプロイグループ
既存のデプロイグループを一つ選択します。または AWS CodeDeploy に新たに作成します。

Amazon ECS タスク定義
Amazon ECS タスク定義ファイルが保存されている入力アーティファクトを選択します。デフォルトのファイルパス以外の場合は、タスク定義ファイルのパスとファイル名を指定します。

入力アーティファクトを... ▼ taskdef.json

デフォルトのパスは taskdef.json です。

AWS CodeDeploy AppSpec ファイル
AWS CodeDeploy AppSpec ファイルが保存されている入力アーティファクトを選択します。デフォルトのファイルパス以外の場合は、AppSpec ファイルのパスとファイル名を指定します。

入力アーティファクトを... ▼ appspec.yaml

タスク定義の動的な更新イメージ - オプション
ブレースホルダーおよび入力アーティファクトを指定することができ、コンテナ定義に使用するイメージの名前を動的にタスク定義を更新します。複数の入力アーティファクトおよびブレースホルダーを指定できます。

入力アーティファクトを持つイメージの詳細

入力アーティファクトを選択する ▼

タスク定義のブレースホルダー文字

画像

結果的に

[デベロッパー用ツール](#) > [CodePipeline](#) > [パイプライン](#) > 新規のパイプラインを作成する

Step 1
パイプラインの設定を選択する

Step 2
ソースステージを追加する

Step 3
ビルドステージを追加する

Step 4
デプロイステージを追加する

Step 5
レビュー

レビュー 情報

ステップ 1: パイプラインの設定を選択する

パイプラインの設定

パイプライン名
nakamuray

Artifact の場所
新しい Amazon S3 バケットがパイプラインのデフォルトアーティファクトストアとして作成されました

サービスロール名
AWSCodePipelineServiceRole-ap-northeast-1-nakamuray

ステップ 2: ソースステージを追加する

ソースアクションプロバイダー

Source アクションプロバイダー
AWS CodeCommit

RepositoryName
facility-booking

BranchName
feature-622

PollForSourceChanges
false

ステップ 2: ソースステージを追加する

ソースアクションプロバイダー

Source アクションプロバイダー
AWS CodeCommit

RepositoryName
facility-booking

BranchName
feature-622

PollForSourceChanges
false

OutputArtifactFormat
CODE_ZIP

ステップ 3: ビルドステージを追加する

アクションプロバイダーを構築する

Build アクションプロバイダー
AWS CodeBuild

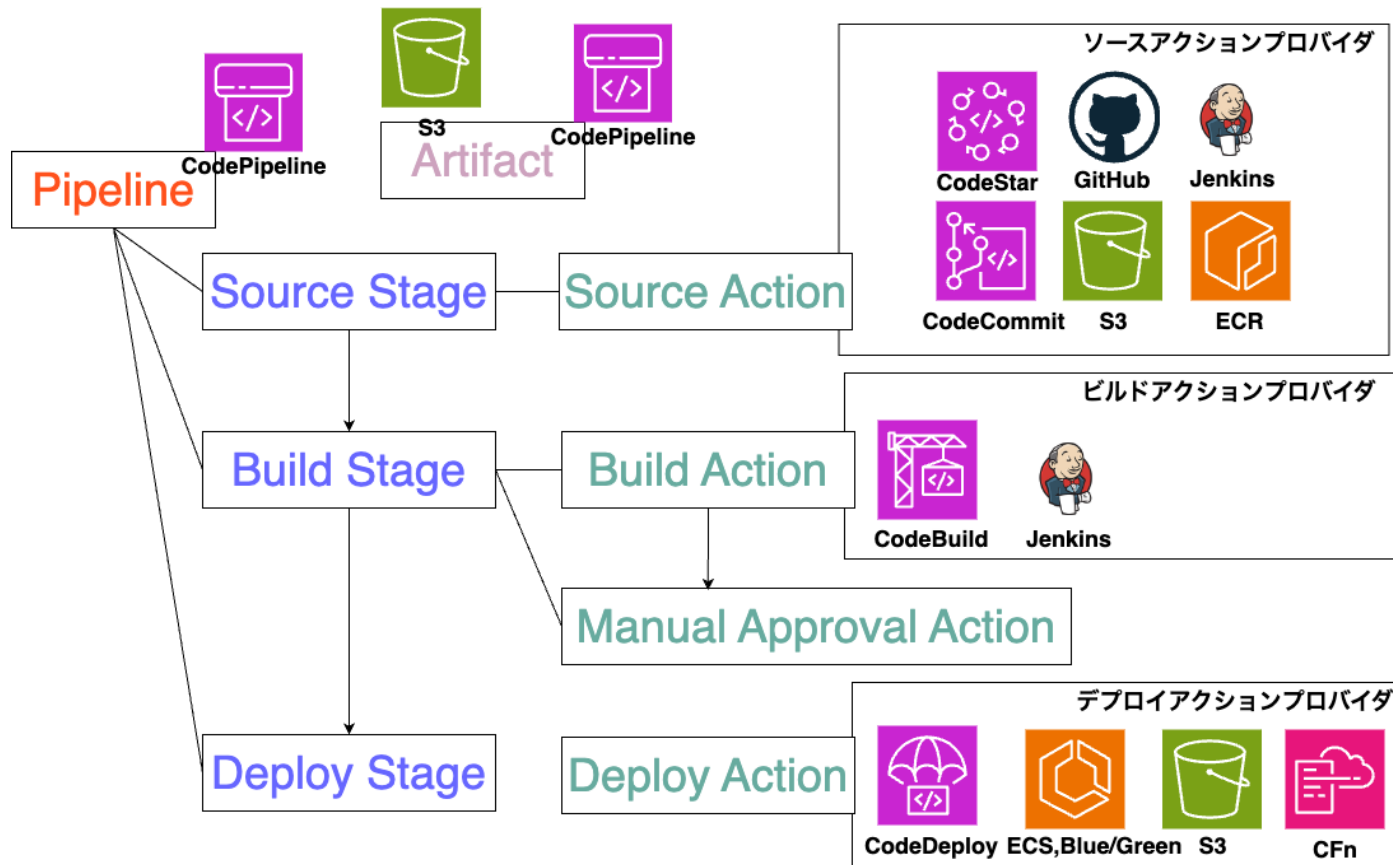
ProjectName
test-dev-project

ステップ 4: デプロイステージを追加する

アクションプロバイダーをデプロイする

5.CodePipelineの主要概念

CodePipelineの主要概念



6.主要概念を反映したCDKコード

主要概念を反映したCDKコード

```
import { Artifact, Pipeline } from 'aws-cdk-lib/aws-codepipeline';
import { CodeBuildAction, CodeStarConnectionsSourceAction,

const pipeline: Pipeline = new Pipeline(this, `${prefix}Pipeline`, {
  pipelineName: `${prefix}-app-pipeline`,
  crossAccountKeys: false,
});

// Artifactの設定
const sourceOutput: Artifact = new Artifact('sourceArtifact');
const buildspecOutput: Artifact = new Artifact('buildspecArtifact');
const buildspecForDeployOutput: Artifact = new Artifact('buildspecForDeployArtifact');

// ソースアクションの設定
const sourceAction: CodeStarConnectionsSourceAction =
  new CodeStarConnectionsSourceAction(this, 'SourceAction', {
    actionName: 'GitHubSource',
    connectionArn: SCP_REPO_CONNECTION_ARN,
    owner: PROJECT_OWNER,
    repo: APP_REPO,
    branch: envConf.appSettings.deployTriggerBranch,
    output: sourceOutput,
  });
```

```
// ソースステージの追加
pipeline.addStage({
  stageName: 'Source',
  actions: [sourceAction],
});

// ビルドステージの追加
pipeline.addStage({
  stageName: 'Build',
  actions: [new CodeBuildAction({
    actionName: 'DockerEcspressoBuild',
    project: project,
    input: sourceOutput,
    outputs: [buildspecOutput],
  })],
});

// ecspressoによるデプロイ
pipeline.addStage({
  stageName: 'Deploy',
  actions: [new CodeBuildAction({
    actionName: 'EcspressoDeploy',
    project: deployProject,
    input: buildspecOutput,
    outputs: [buildspecForDeployOutput],
  })],
});
```

まとめ

まとめ

- ・ CodePipelineは一つの加工製品の原料調達→加工→配送の流れを形成すると考えるとわかりやすいかもしれない
- ・ CodePipelineにはパイプライン、ステージ、アクションという重要な概念がある
- ・ AWSサービス自体への理解が捗ると、CDKを書くのが上手になる
- ・ 次回はCodeBuildを深掘りしたい