AWS Black Belt Online Seminar

Amazon CodeCatalyst
Workflow 編

田中 創一郎

Partner Solutions Architect
2023/12

自己紹介

名前：田中 創一郎

所属：パートナーアライアンス統括本部

　　ストラテジック SI 技術本部

経歴：事業会社の情シス、

　　SIer の Java エンジニア・IT アーキテクト

好きな AWS サービス：

Amazon CodeCatalyst

AWS Control Tower

AWS Black Belt Online Seminar とは

- 「サービス別」「ソリューション別」「業種別」などのテーマに分け、アマゾン ウェブ サービス ジャパン合同会社が提供するオンラインセミナーシリーズです
- AWS の技術担当者が、 AWS の各サービスやソリューションについてテーマごとに動画を公開します
- 以下の URL より、過去のセミナー含めた資料などをダウンロードすることができます
  - https://aws.amazon.com/jp/aws-jp-introduction/aws-jp-webinar-service-cut/
  - https://www.youtube.com/playlist?list=PLzWGOASvSx6FIwIC2X1nObr1KcMCBBlqY

ご感想は X (Twitter) へ！ハッシュタグは以下をご利用ください
#awsblackbelt

内容についての注意点

- 本資料では資料作成時点のサービス内容および価格についてご説明しています。AWS のサービスは常にアップデートを続けているため、最新の情報は AWS 公式ウェブサイト (https://aws.amazon.com/) にてご確認ください
- 資料作成には十分注意しておりますが、資料内の価格と AWS 公式ウェブサイト記載の価格に相違があった場合、AWS 公式ウェブサイトの価格を優先とさせていただきます
- 価格は税抜表記となっています。日本居住者のお客様には別途消費税をご請求させていただきます
- 技術的な内容に関しましては、有料の AWS サポート窓口へお問い合わせください
- 料金面でのお問い合わせに関しましては、カスタマーサポート窓口へお問い合わせください (マネジメントコンソールへのログインが必要です)

Black Belt CodeCatalyst シリーズ 

- Overview 編
- Spaces 編
- Projects, Blueprints 編
- Source repositories 編
- Dev Environments 編
- Workflow 編
- Issues 編
- Identity, permissions, and access 編
- Extensions 編

シリーズ構成

- 全体像をお伝えする Overview 編
- 各機能の詳細についてお伝えする各機能編

Black Belt CodeCatalyst シリーズの対象読者

- チーム開発をするすべてのアプリケーション開発者

Workflow 編の対象読者

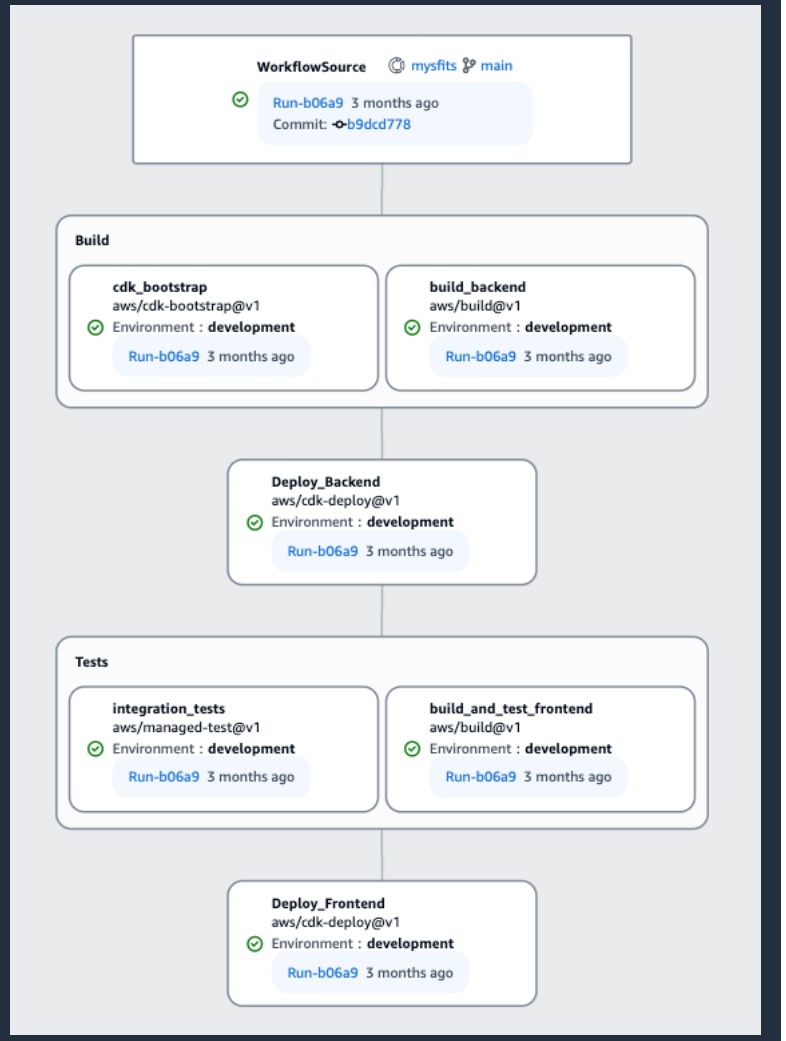
- CI/CD パイプラインの作成・管理方法を知りたい方

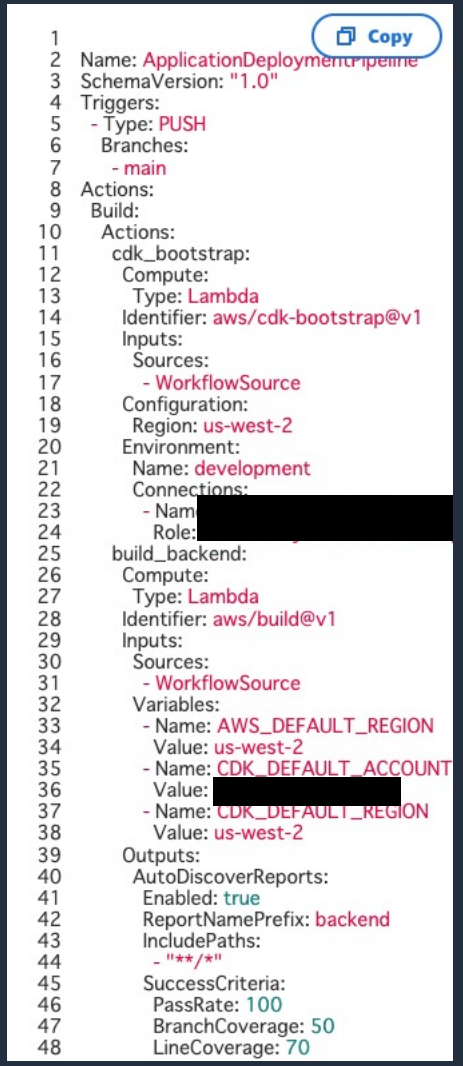
アジェンダ

1. Workflow の全体像
2. Workflow の構成要素
3. Workflow の作成方法
4. Workflow の実行・管理方法
5. Actions のベストプラクティス
6. Workflow のクォータ
7. まとめ

Workflow の全体像

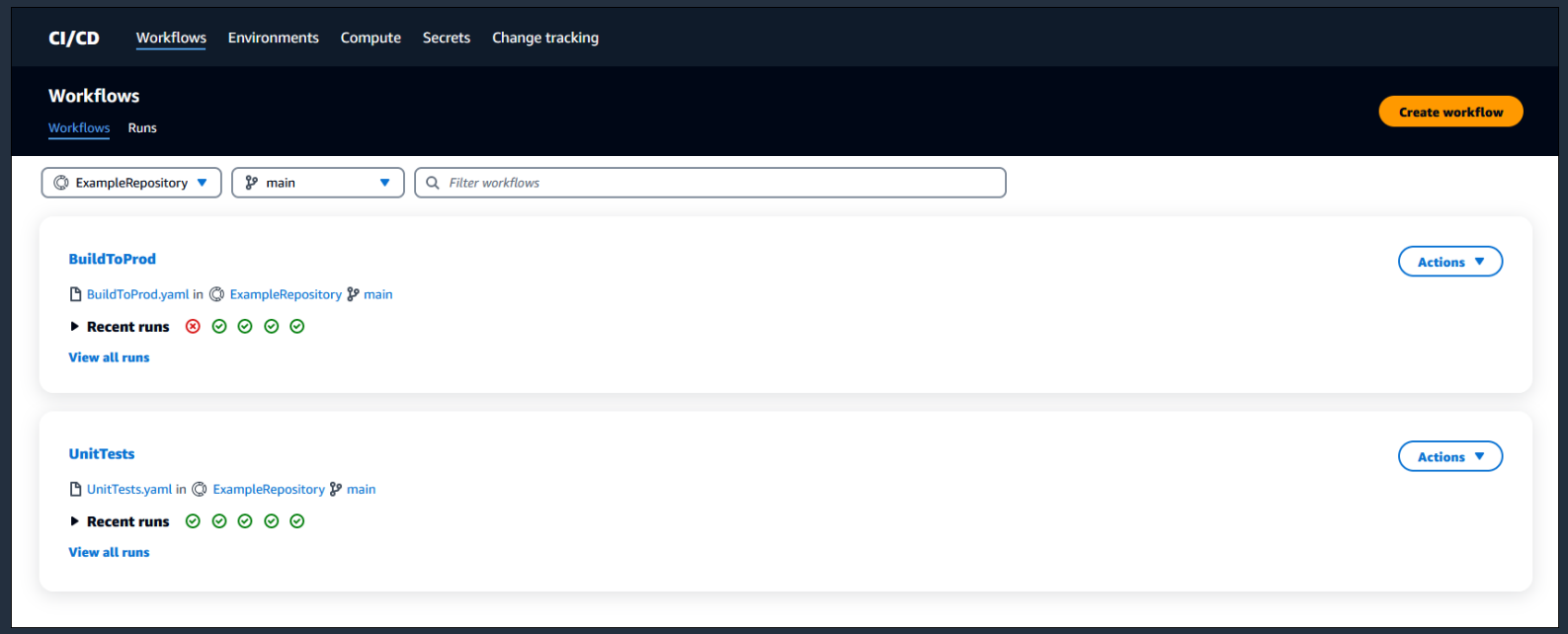
Workflow の定義





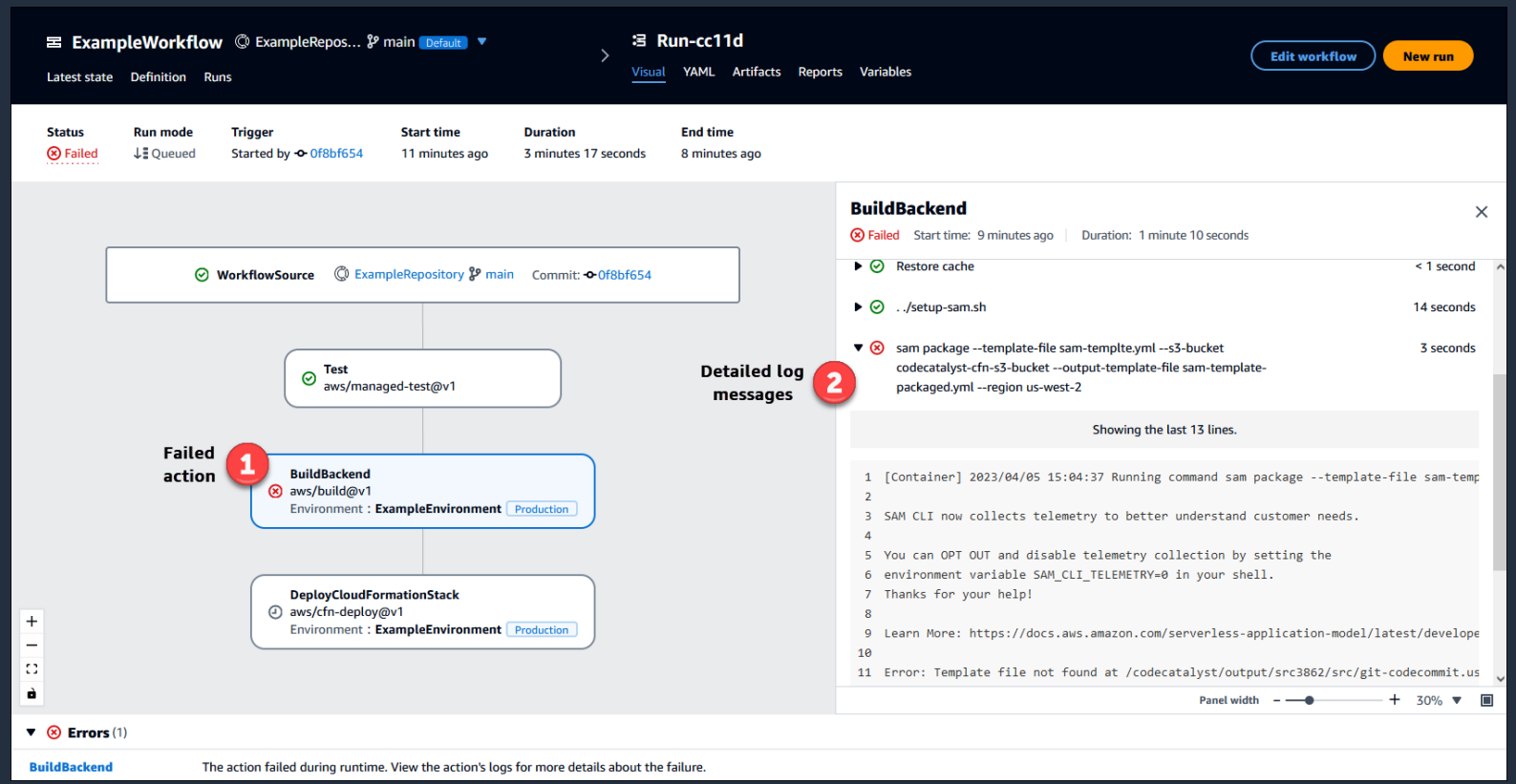
- コードをビルド・テスト・デプロイするパイプラインを定義・実行
- GUI でも YAML ファイル形式でも定義が可能
  - はじめは GUI で作成し、後で YAML ファイルを編集するといった相互利用もできる
- パイプラインの構成も、ビルド・テストの設定も一つの定義で一元管理

Workflow の参照



- Workflow の画面で一覧表示
- 検索も可能

Workflow の実行結果



- 実行された Workflow の履歴と詳細が参照できる
- ログや実行時間が記録される
  - 左の例では①の失敗した action 内のエラーになった処理の結果を②のログで確認している

Workflow の構成要素

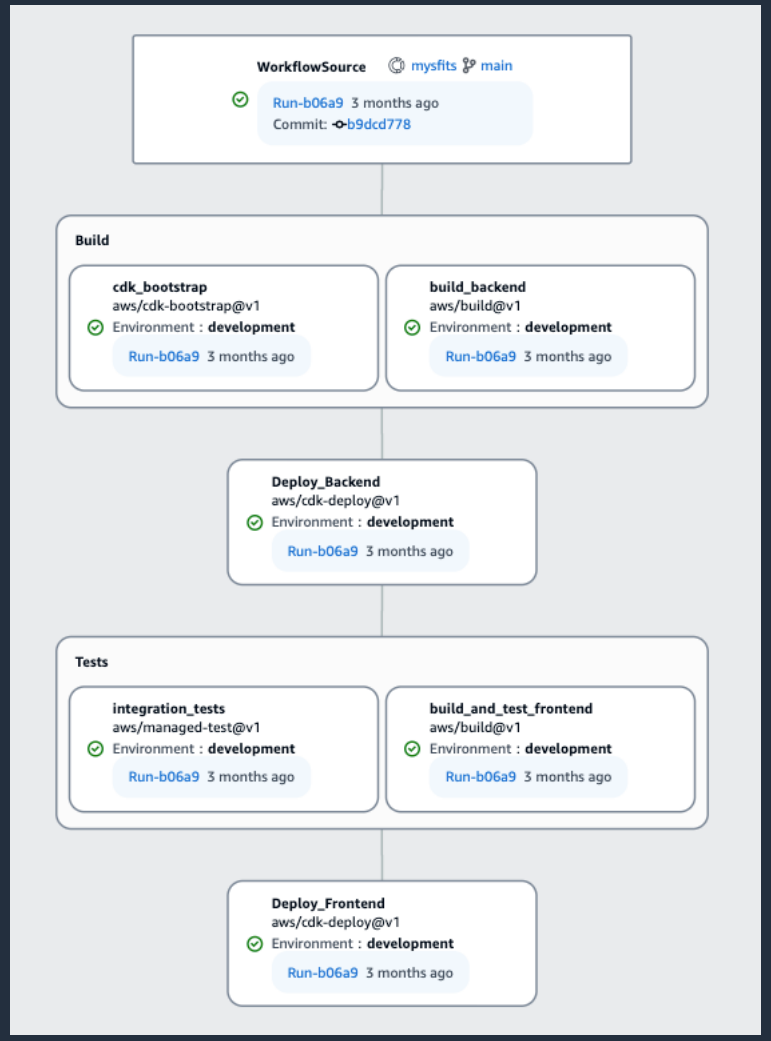
Workflow の構成要素

1. 主要な構成要素
2. 代表的な Actions
3. Actions の設定項目

Workflow の構成要素

1. 主要な構成要素
2. 代表的な Actions
3. Actions の設定項目

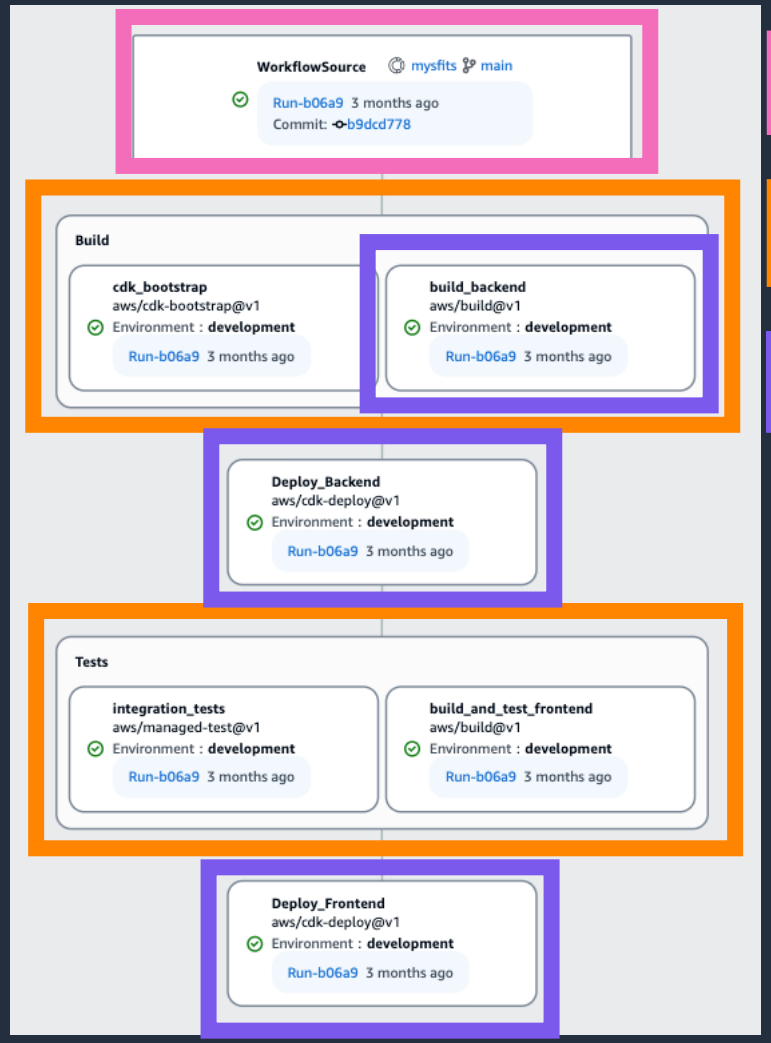
全体構成



以下の要素の組み合わせでパイプラインが構成される

- Triggers：Workflow を起動するトリガー
- Action Groups：action を管理しやすいようにまとめるもの
- Actions：具体的な処理

全体構成



Triggers

Action Groups

Actions

以下の要素の組み合わせでパイプラインが構成される

- Triggers：Workflow を起動するトリガー
- Action Groups：action を管理しやすいようにまとめるもの
- Actions：具体的な処理

その他の Top Level で定義できる項目

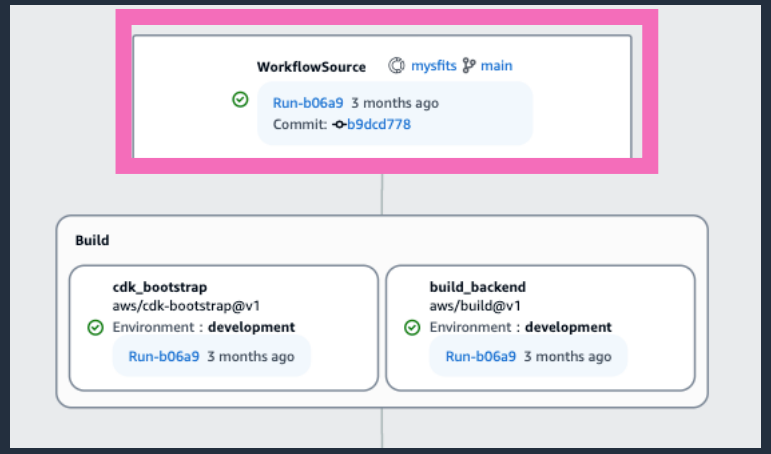
```
# Name
Name: workflow-name

# Schema version
SchemaVersion: 1.0

# Run mode
RunMode: QUEUED|SUPERSEDED|PARALLEL

# Compute
Compute:
...

# Triggers
Triggers:
...

# Actions
Actions:
...
```

ワークフロー図としては登場しないが、
ワークフロー定義の Top Level では以下の項目も定義する

- Name：Workflow の名称
- SchemaVersion：Workflow 定義のスキーマのバージョン
- RunMode：同時実行時の処理方法
- Compute：action を実行するときの実行環境
  - 全体で一律定義 or action 単位での設定が可能

※ RunMode と Compute は任意設定

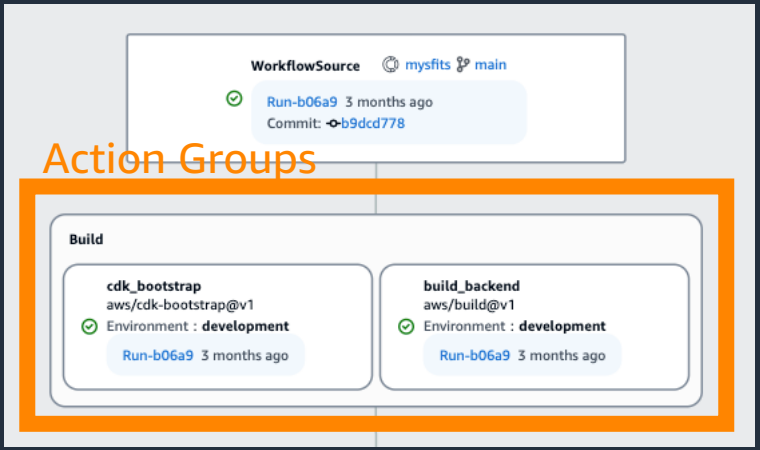
Triggers




```
Triggers:
  - Type: PUSH
    Branches:
      - main
  - Type: PULLREQUEST
    Branches:
      - feature\-.*
      - bugfix\-.*
      - release\-.*
      - docs\-.*
    Events:
      - OPEN
      - REVISION
```

- Workflow を起動するトリガー
- リポジトリへの Push と Pull Request にて発動（Type: Push, PULLREQUEST）
  - Pull Request の場合はさらに細かい条件指定
    - Pull Request が作成されたとき（Events: OPEN）
    - Pull Request が閉じられたとき（Events: CLOSE）
    - Pull Request が作成されるか、Pull Request に新しいリビジョンが作成されたとき（Events: REVISION）
- 対象のブランチ名、もしくはブランチ名のパターン（例：feature- ではじまるブランチ）を指定
- ブランチ内の特定のファイル、ディレクトリのみの変更を条件にすることもできる

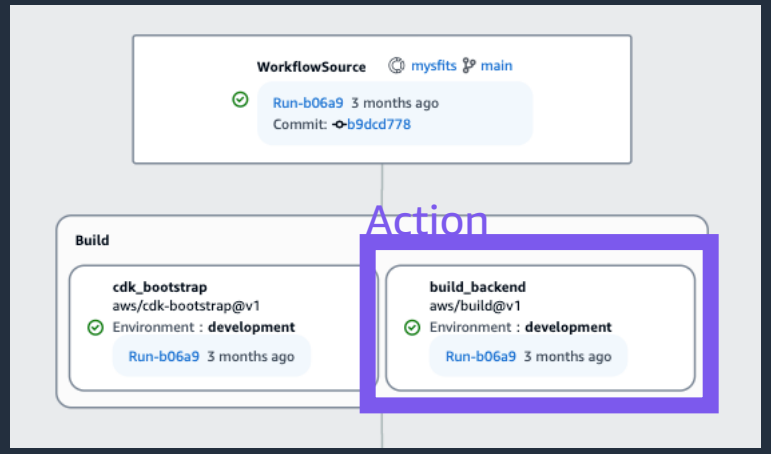
Action Groups




```
Actions:
  BuildAndTestGroup: # Action group 1
    Actions:
      BuildAction:
      ...
      TestAction:
      ...
  DeployGroup: #Action group 2
    DependsOn:
      - BuildAndTestGroup
    Actions:
      DeployAction1:
      ...
      DeployAction2:
      ...
```

- Action をまとめることで、Workflow を整理することができる
- Action・Action Group の依存関係を設定することで、それぞれの前後関係を定義する
- 依存する Action・Action Group は複数定義できる
- GUI では編集できず、YAML ファイルにて設定する必要がある

Actions



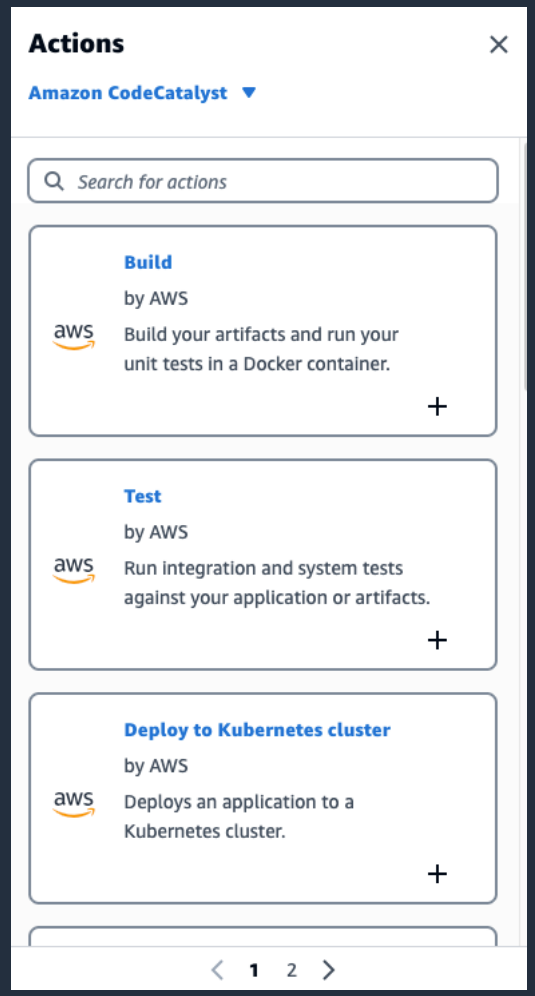

```
Actions:
  BuildAndTestGroup: # Action group 1
    Actions:
      BuildAction:
      ...
      TestAction:
      ...
  DeployGroup: #Action group 2
    DependsOn:
      - BuildAndTestGroup
    Actions:
      DeployAction1:
      ...
      DeployAction2:
      ...
```

- Workflow 内で実行される処理
- 複数の action を直列に実行することも、並列で実行することもできる
- action 間の依存関係を設定することで、それぞれの前後関係を定義する
  - 依存する action は複数定義できる
- 以下の 4つの種類がある
  - CodeCatalyst actions
  - CodeCatalyst Labs actions
  - GitHub Actions
  - Third-party actions
- 独自の custom action を作成することもできる
  - 作成方法は Developer Guide を参照

CodeCatalyst actions

- CodeCatalyst 開発チームによって作成・保守、フルサポートされるビルド、テスト、デプロイの他、様々なタスクを実行する action

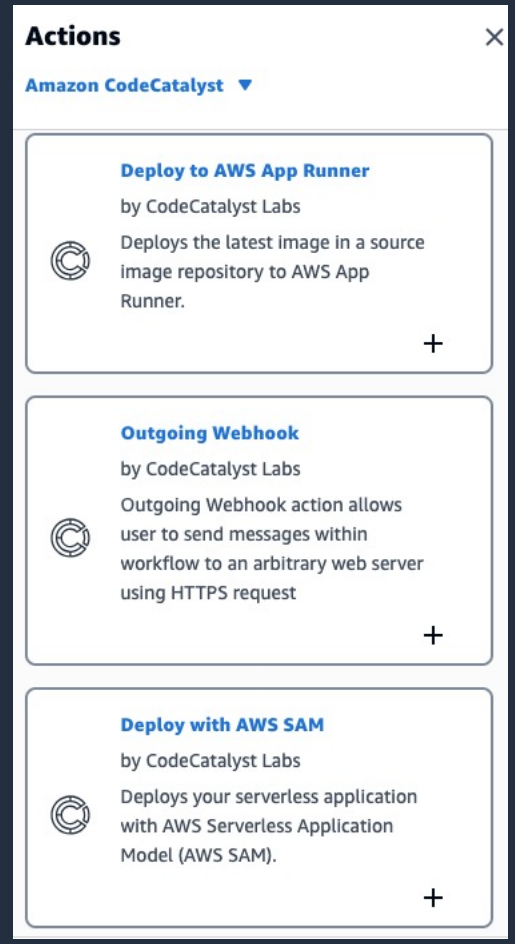
- 以下のような action が用意されている（一部抜粋）
  - build
  - test
  - Deploy AWS CloudFormation stack
  - Deploy to Amazon ECS
  - AWS CDK deploy
  - Amazon S3 publish



CodeCatalyst Labs actions

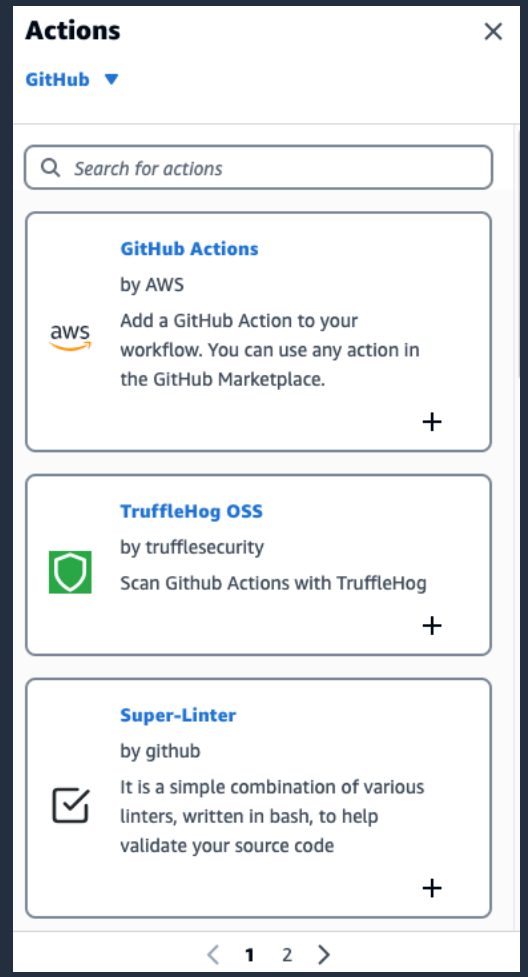
- 実験的なアプリケーションのためのラボである CodeCatalyst Labs の一部で、AWS はベストエフォートで保守・サポートするという位置づけ

- Amazon CodeCatalyst カテゴリーのアクションから選択
  - by CodeCatalyst Labs と記載されている action が該当する



GitHub Actions

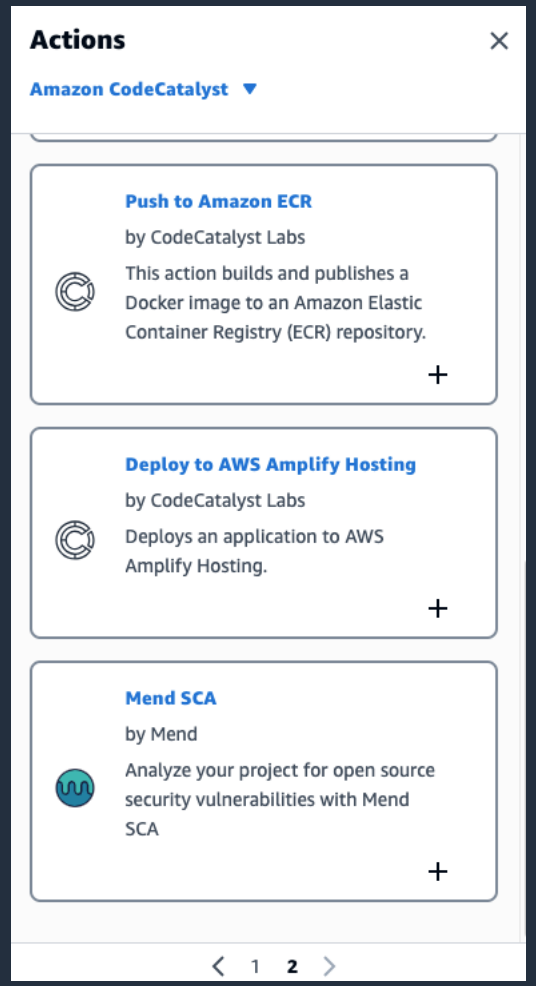
- GitHub カテゴリーのアクションから選択
  - その他、GitHub Marketplace に存在する action も利用できる
- CodeCatalyst Actions に比べて、以下の制約がある
  - Lambda Compute type が利用できない
  - 内部で github context や GitHub 固有のリソースを参照している action は動作しない
    - 例：GitHub リソースを追加・変更するアクション
  - Docker container actions は利用できるが、デフォルトの Docker ユーザー（root）で実行する必要がある



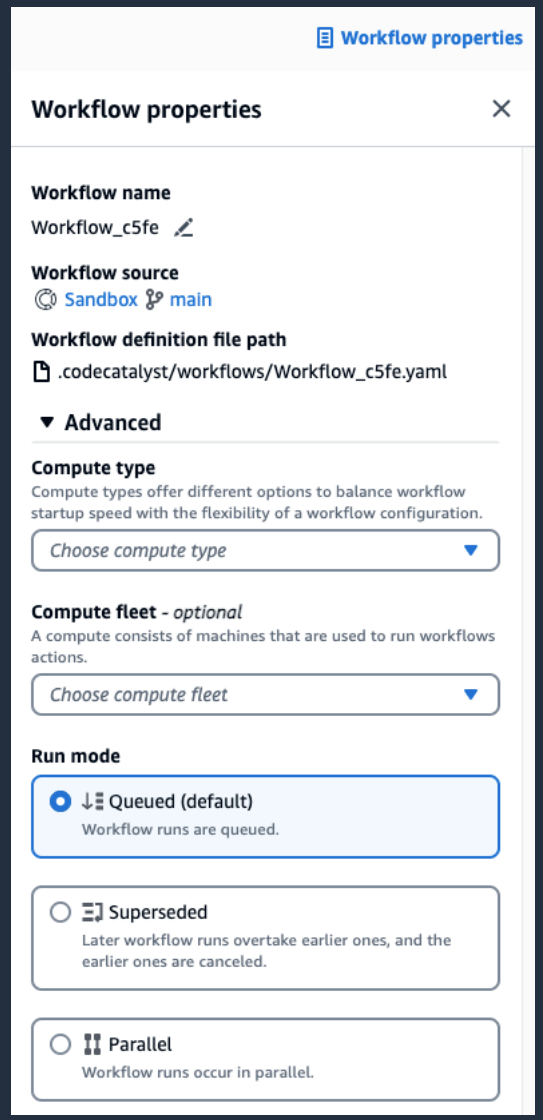
Third-party actions

- Third-party ベンダーが作成し、CodeCatalyst コンソールで利用できるようにした action
  - 例：Mend 社から提供されている Mend SCA action

- Amazon CodeCatalyst カテゴリーのアクションから選択する
  - by “*Third-party vender-name*” と記載されている action が該当

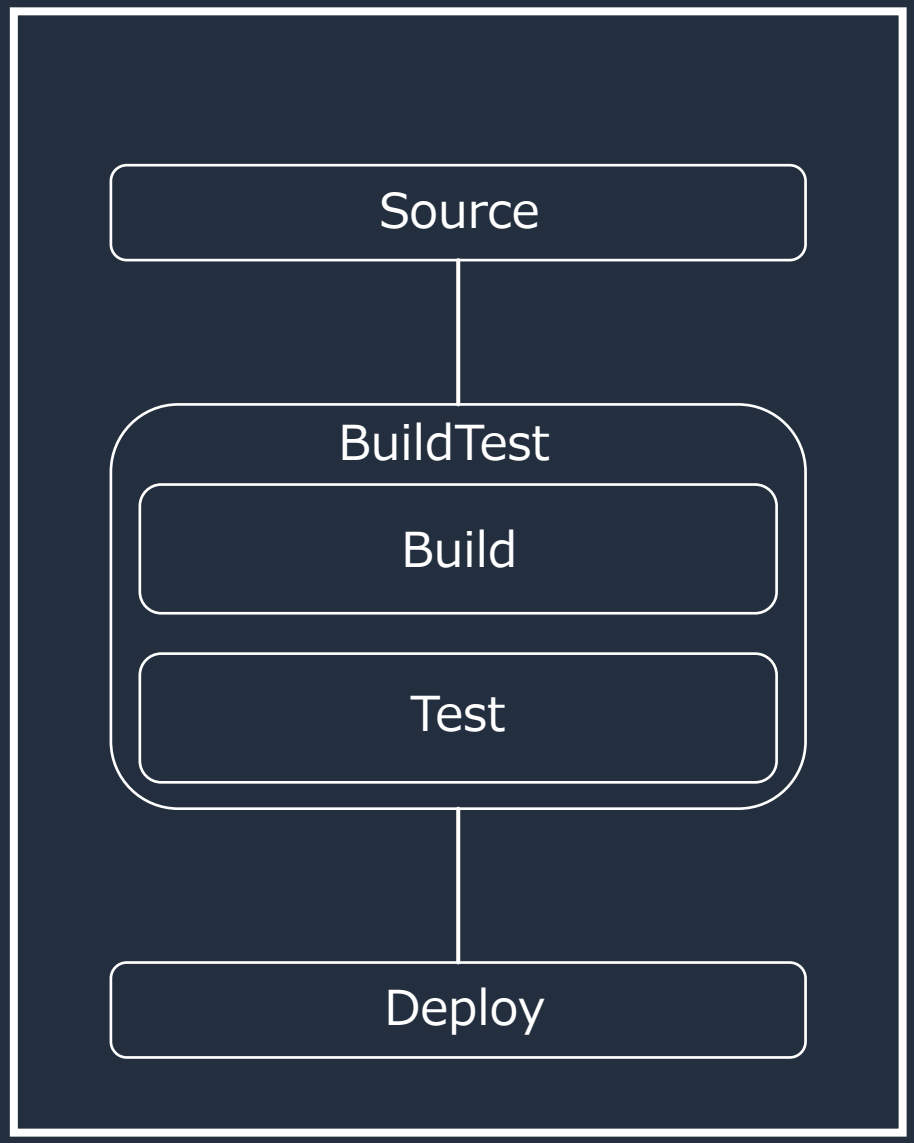


RunMode



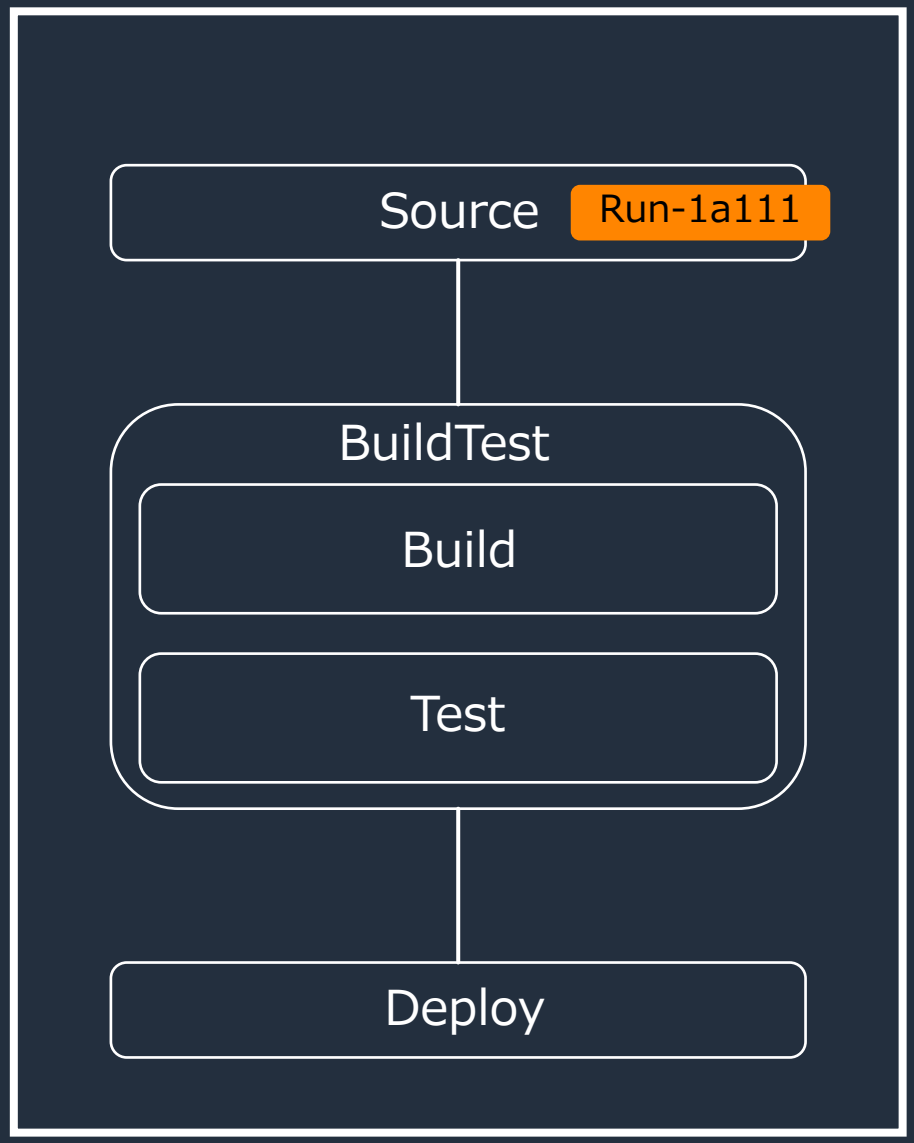
- 同時実行時の処理方法
- 以下の 3つから選択
  - QUEUED
    - 複数の実行がキューに入れられ、順に実行される
    - キューには最大 50 件の実行を含めることができる
    - デフォルト値となっている
  - SUPERSEDED
    - 複数の実行がキューに入れられ、順に実行される
    - 既に待機中の処理があり、後から実行されたワークフローの処理が追いついた場合、後の処理が優先され、既に待機中の処理はキャンセルされる
  - PARALLEL
    - 複数のワークフローが同時実行される
    - Space あたり 1000 の同時実行が上限

RunMode の動作（QUEUED）



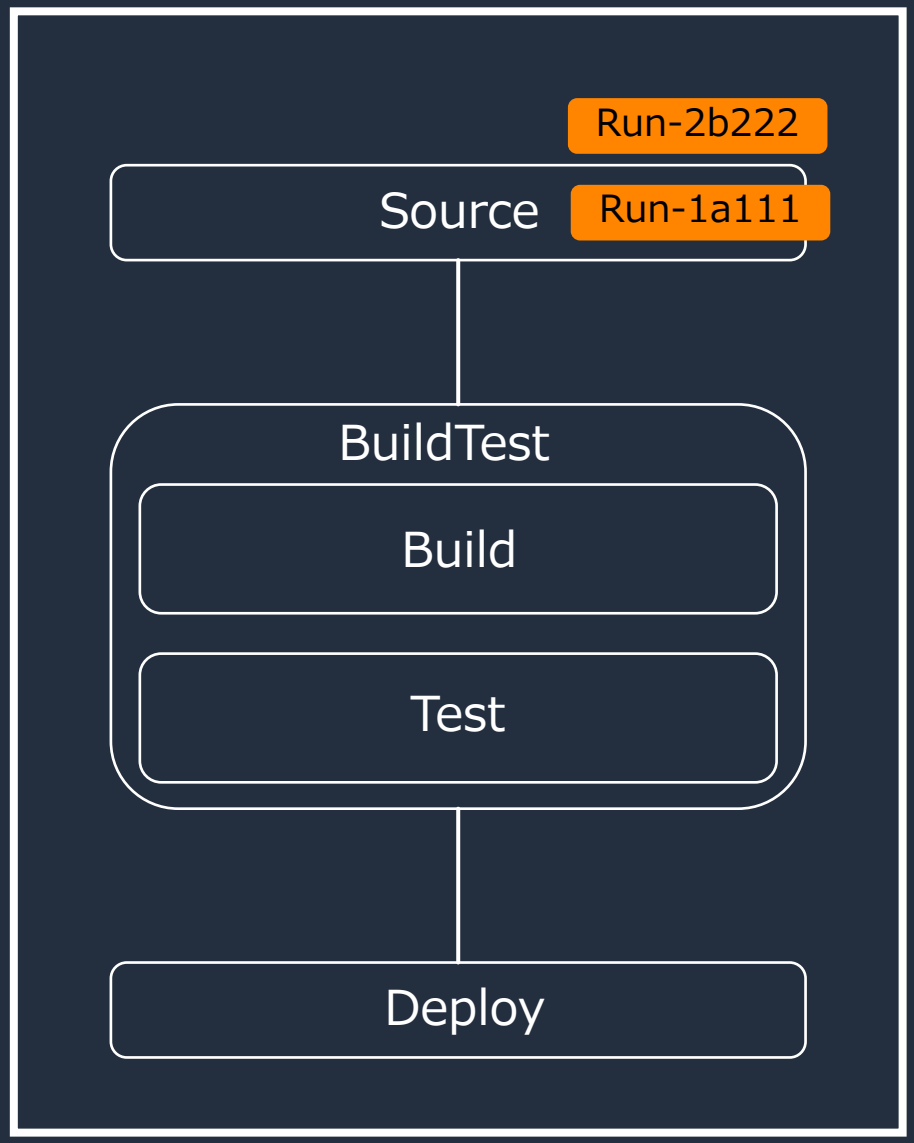
QUEUED の場合は順に実行される

RunMode の動作（QUEUED）



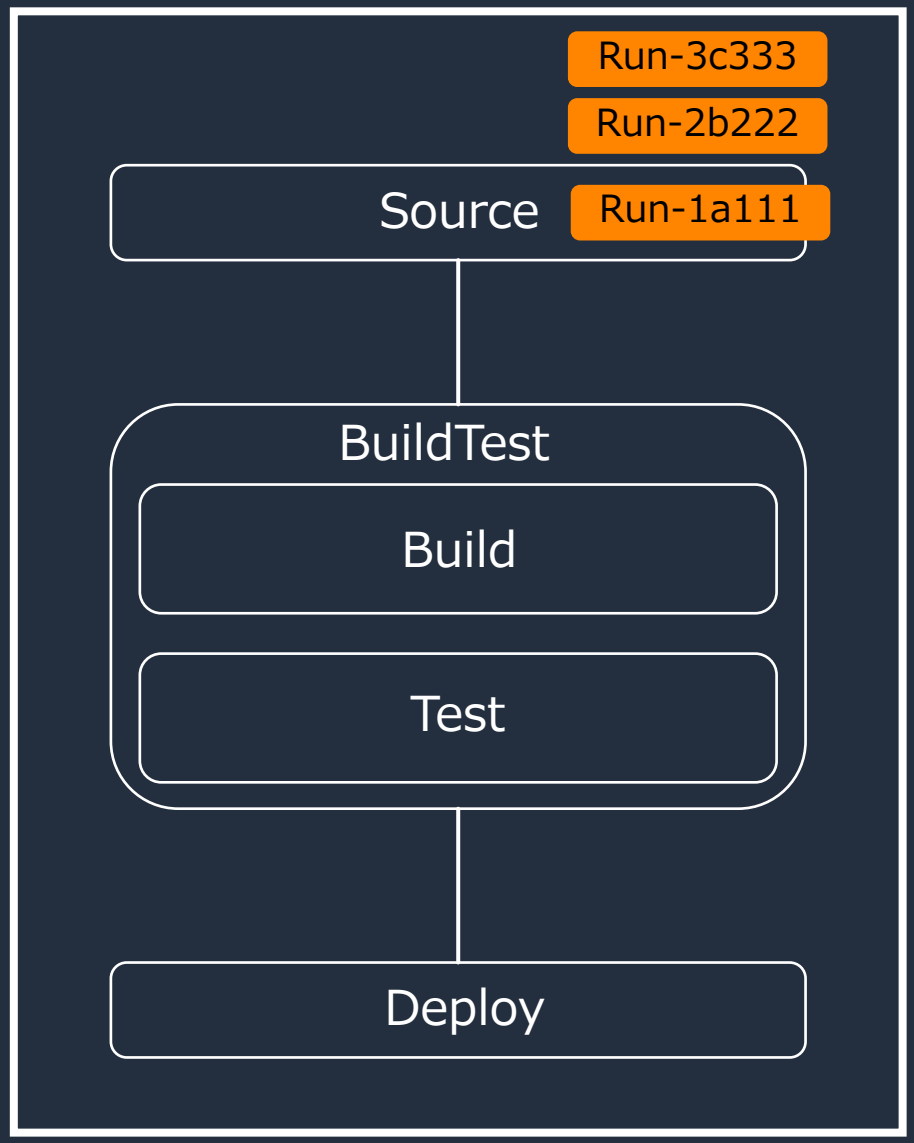
QUEUED の場合は順に実行される

RunMode の動作（QUEUED）



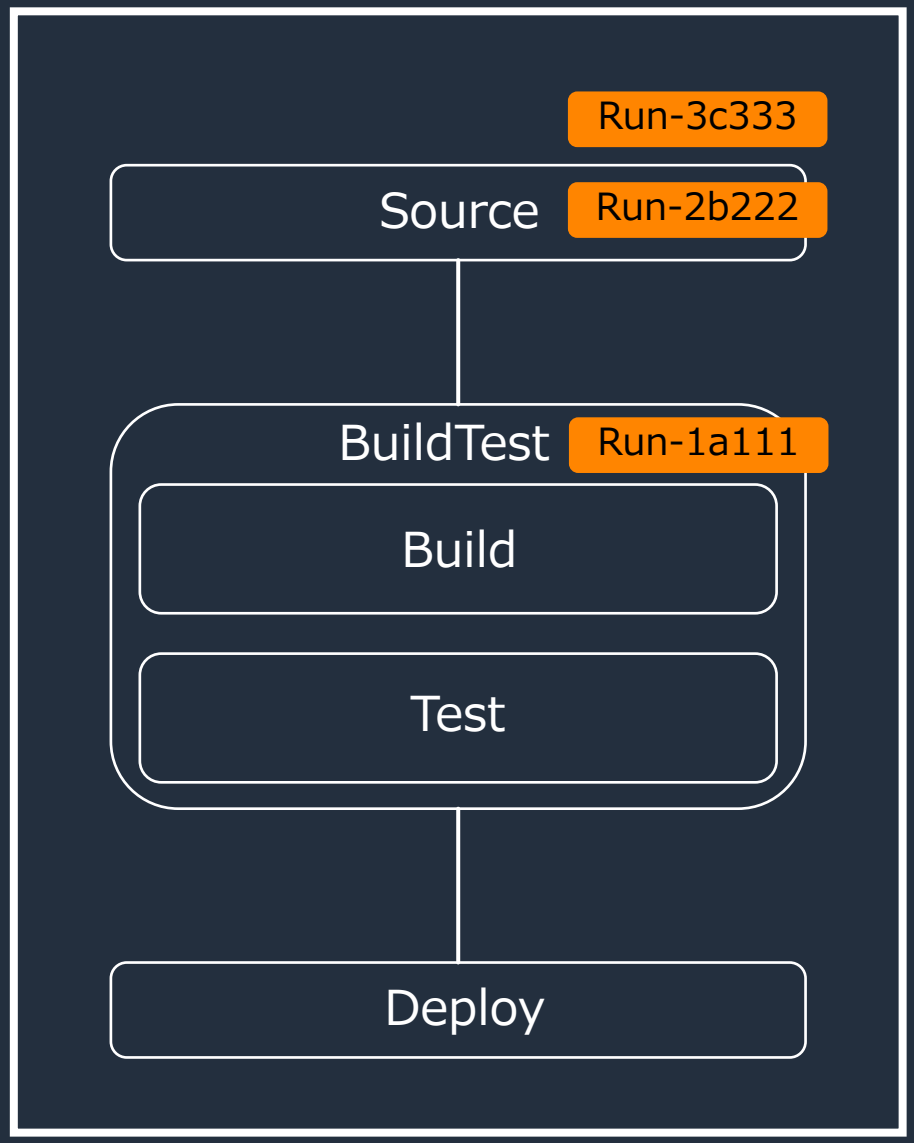
QUEUED の場合は順に実行される

RunMode の動作（QUEUED）



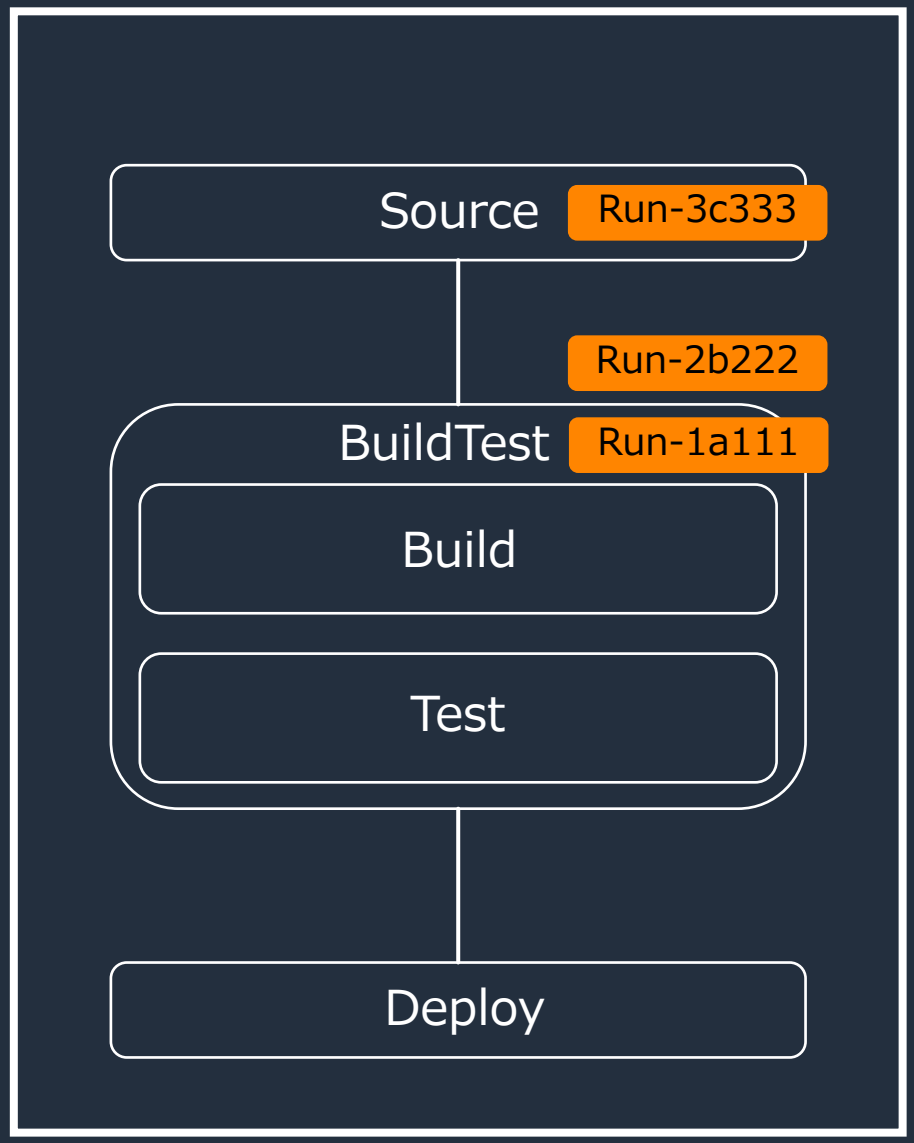
QUEUED の場合は順に実行される

RunMode の動作（QUEUED）



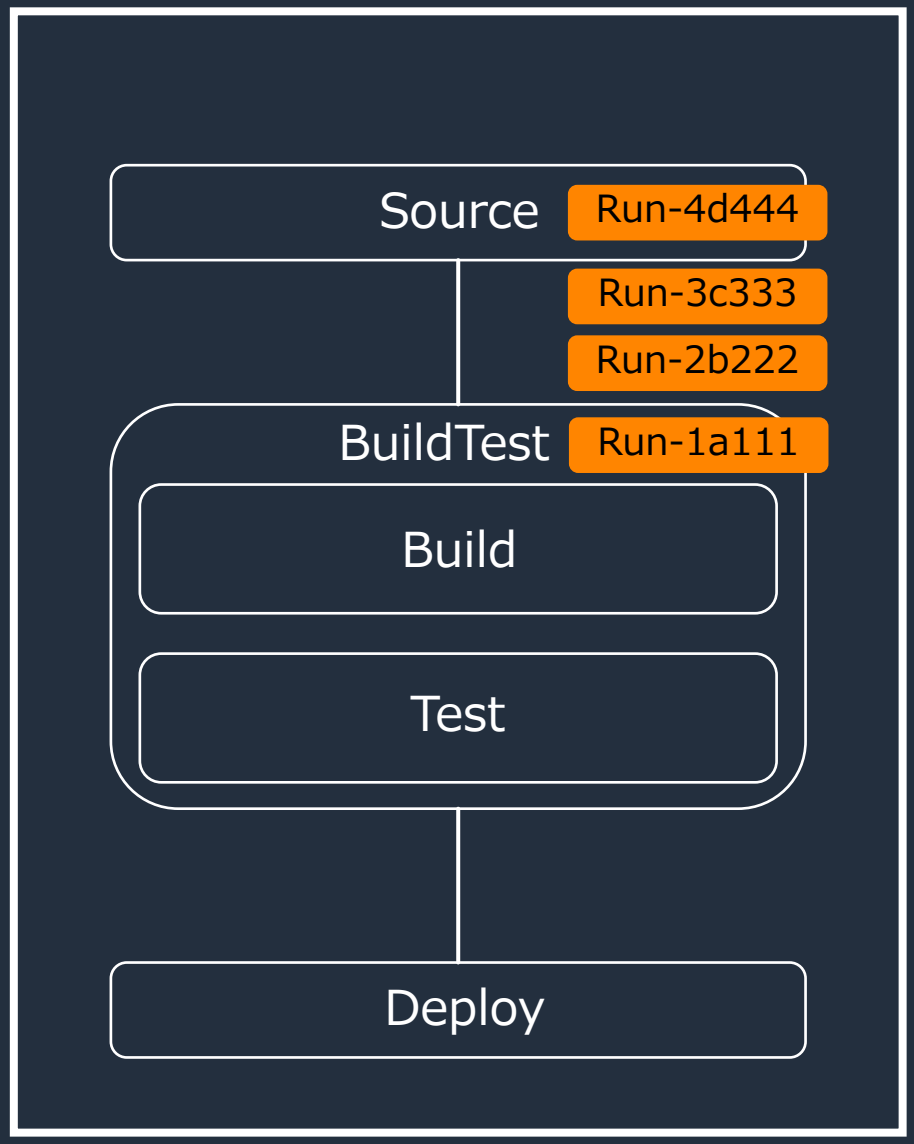
QUEUED の場合は順に実行される

RunMode の動作（QUEUED）



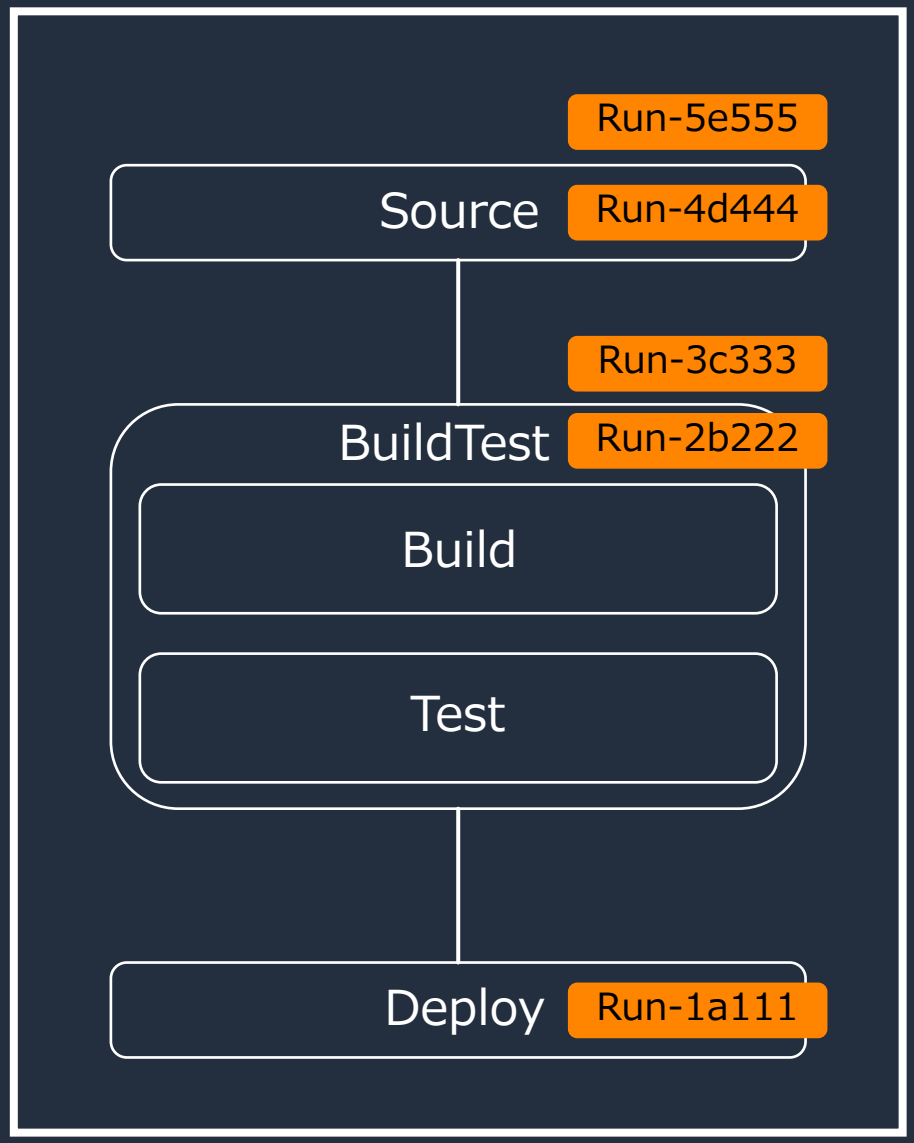
QUEUED の場合は順に実行される

RunMode の動作（QUEUED）



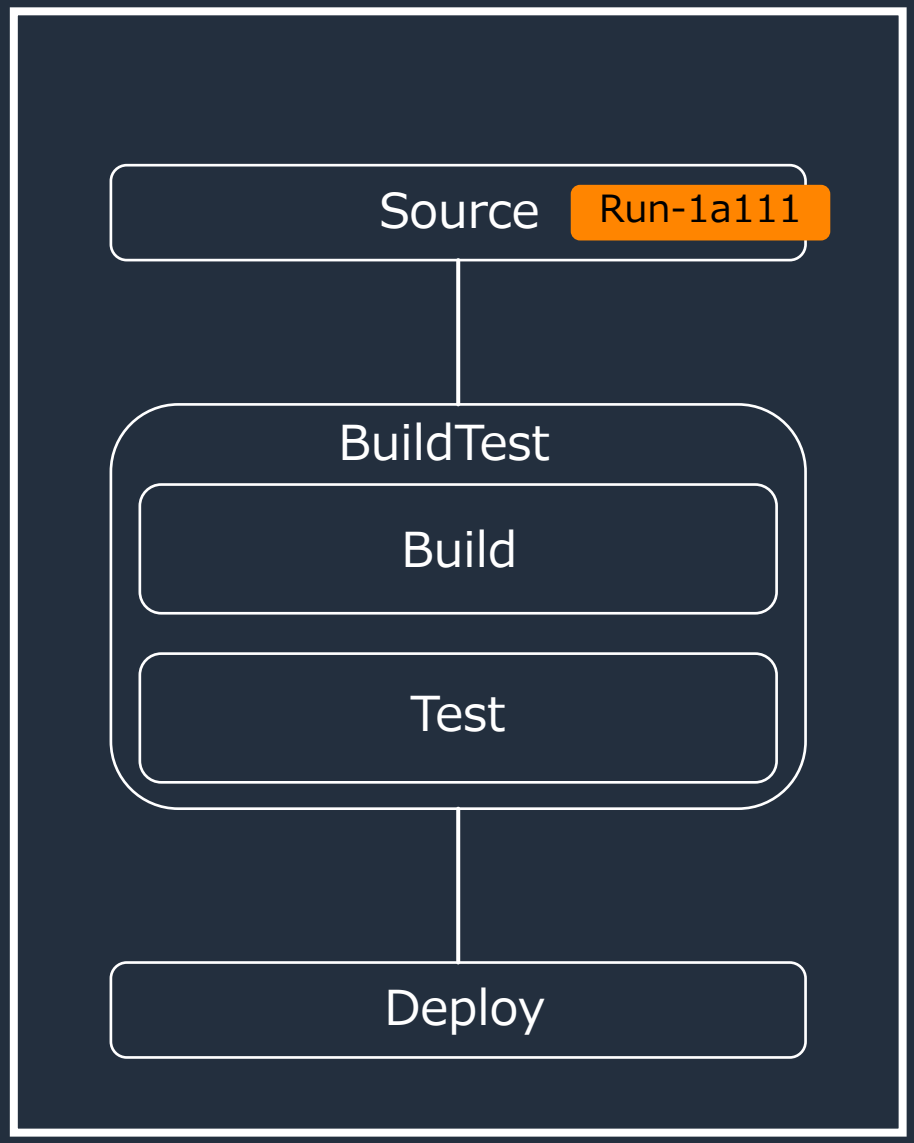
QUEUED の場合は順に実行される

RunMode の動作（QUEUED）



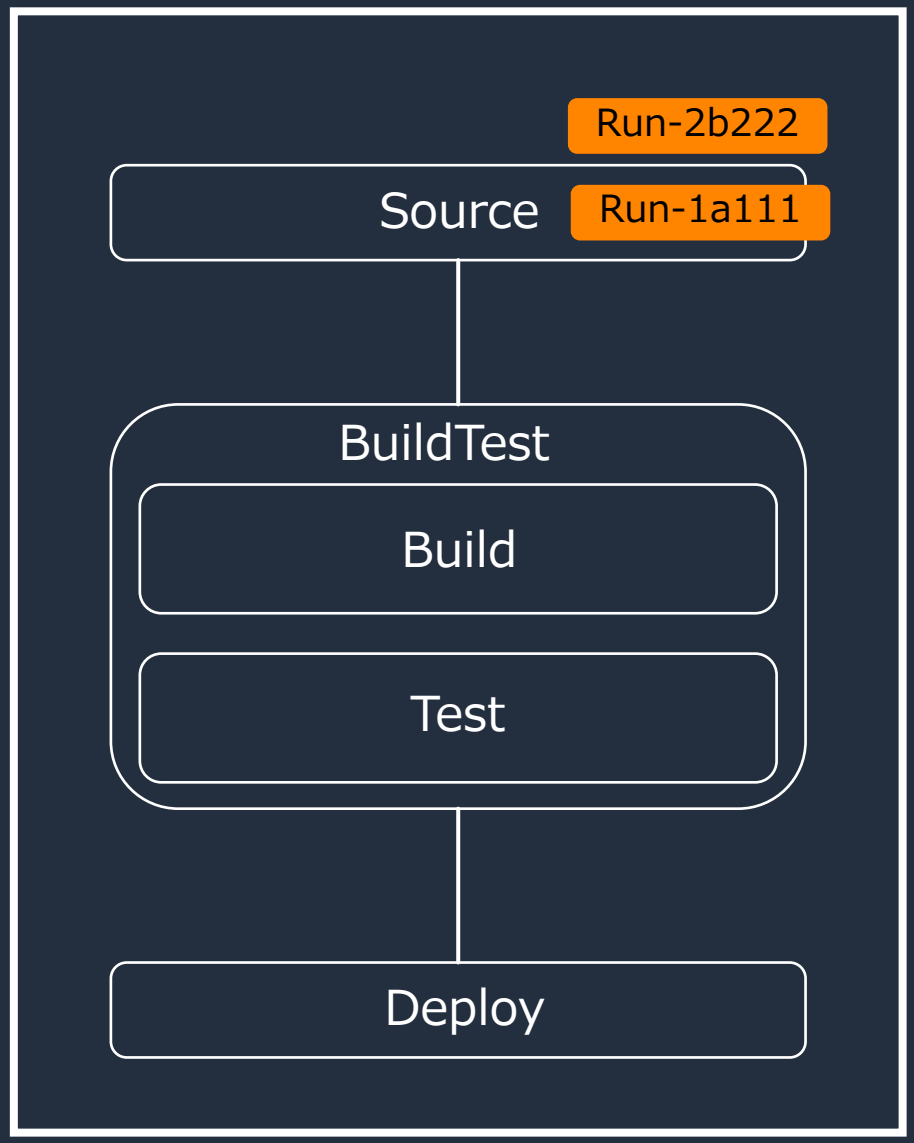
QUEUED の場合は順に実行される

RunMode の動作（SUPERSEDED）



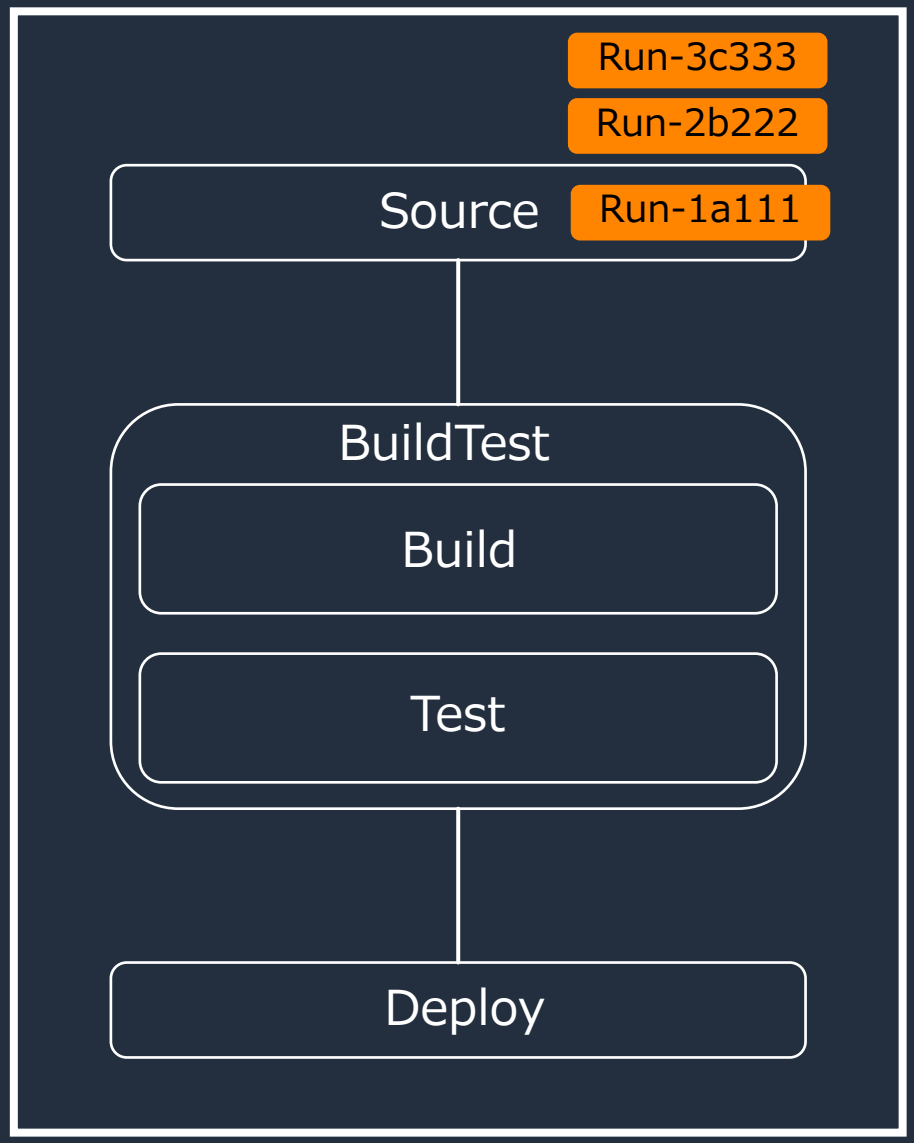
SUPERSEDED の場合、待っている処理は最新の Workflow のみが有効になる

RunMode の動作（SUPERSEDED）



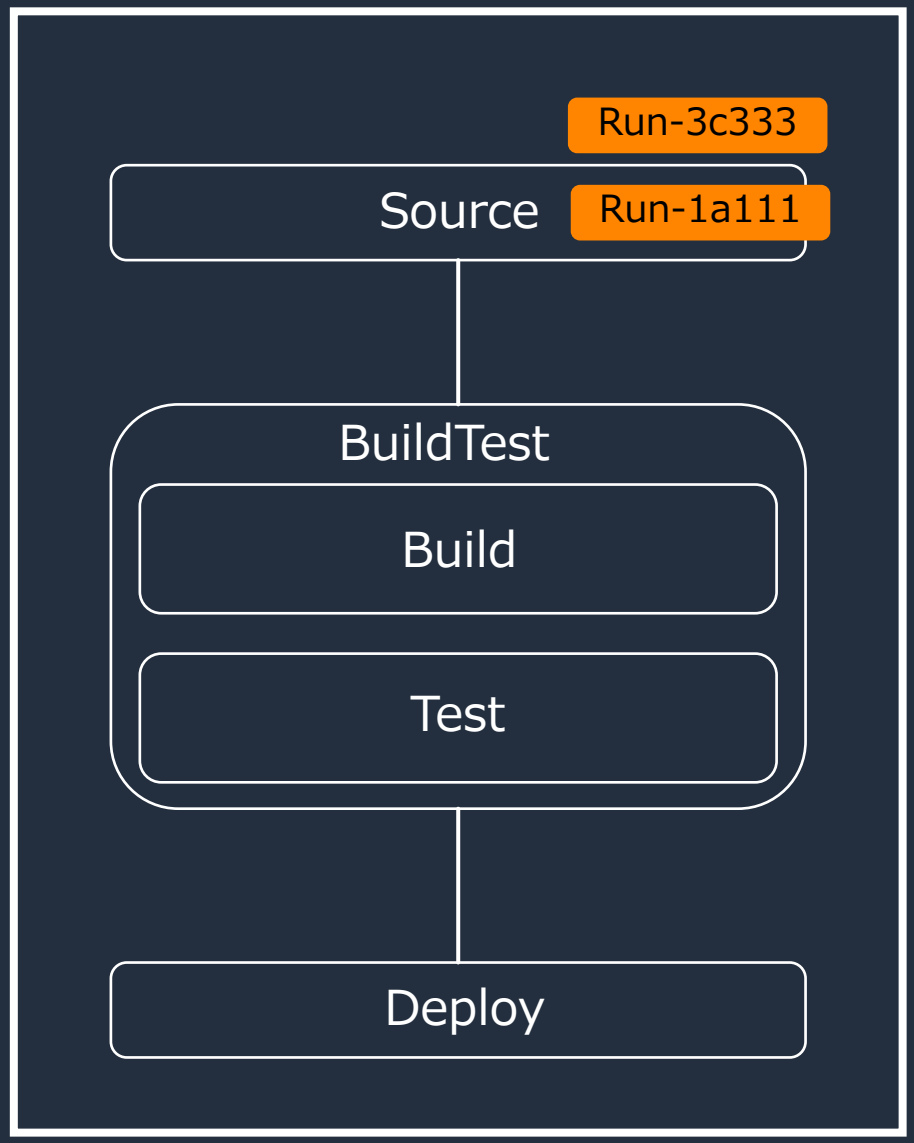
SUPERSEDED の場合、待っている処理は最新の Workflow のみが有効になる

RunMode の動作（SUPERSEDED）



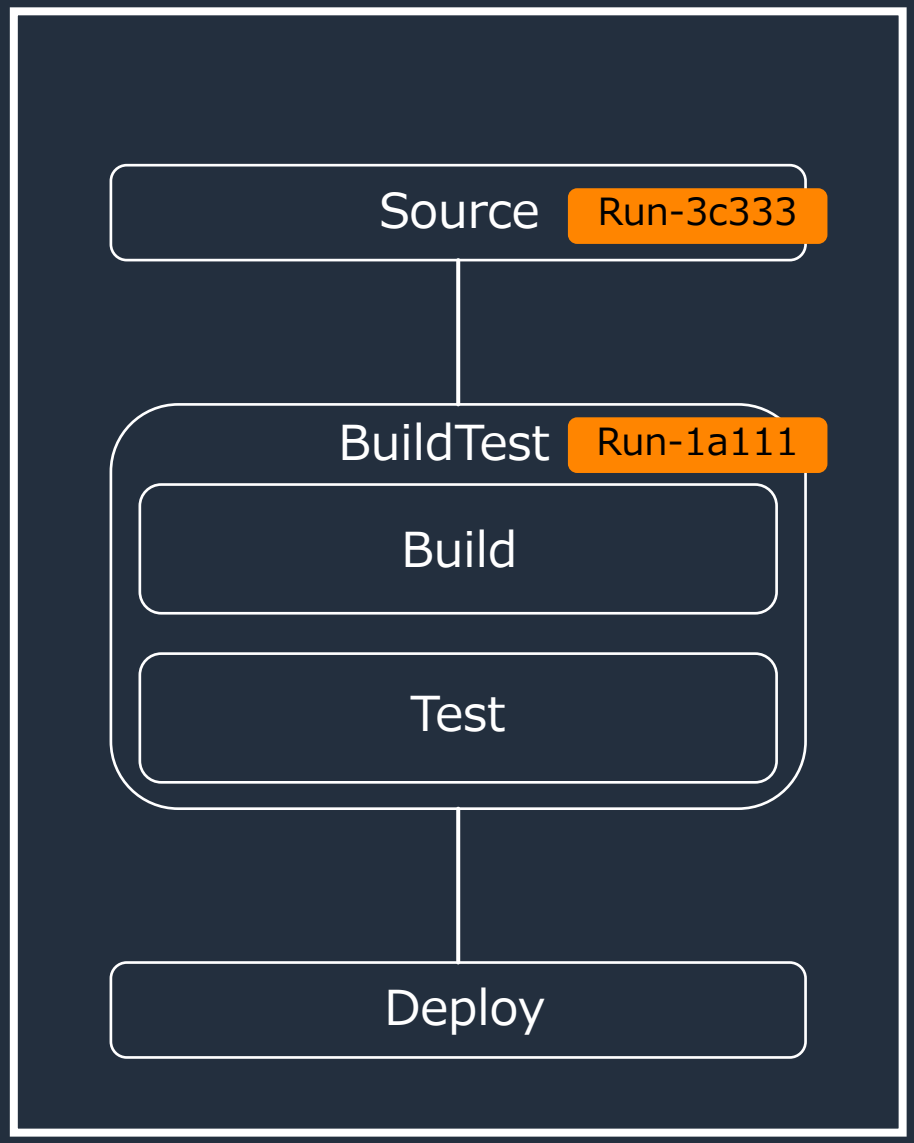
SUPERSEDED の場合、待っている処理は最新の Workflow のみが有効になる

RunMode の動作（SUPERSEDED）



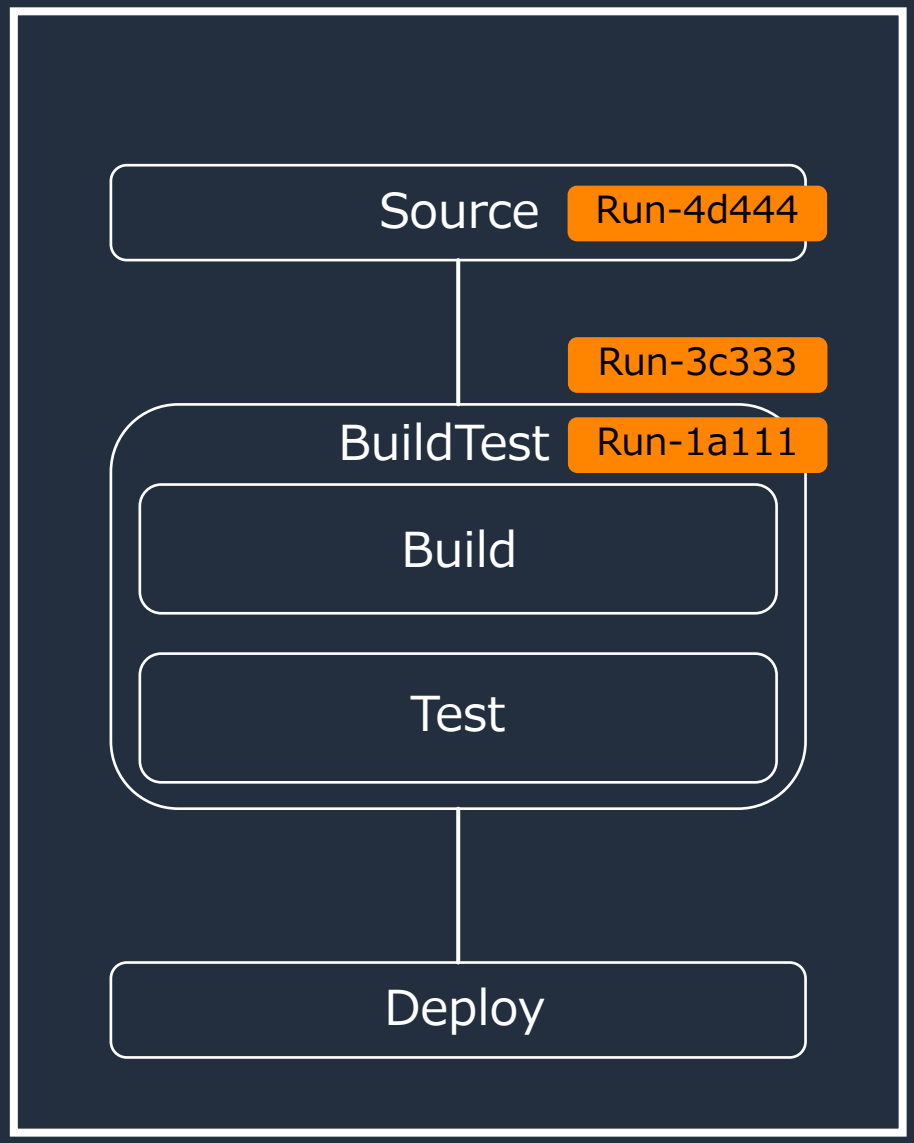
SUPERSEDED の場合、待っている処理は最新の Workflow のみが有効になる

RunMode の動作（SUPERSEDED）



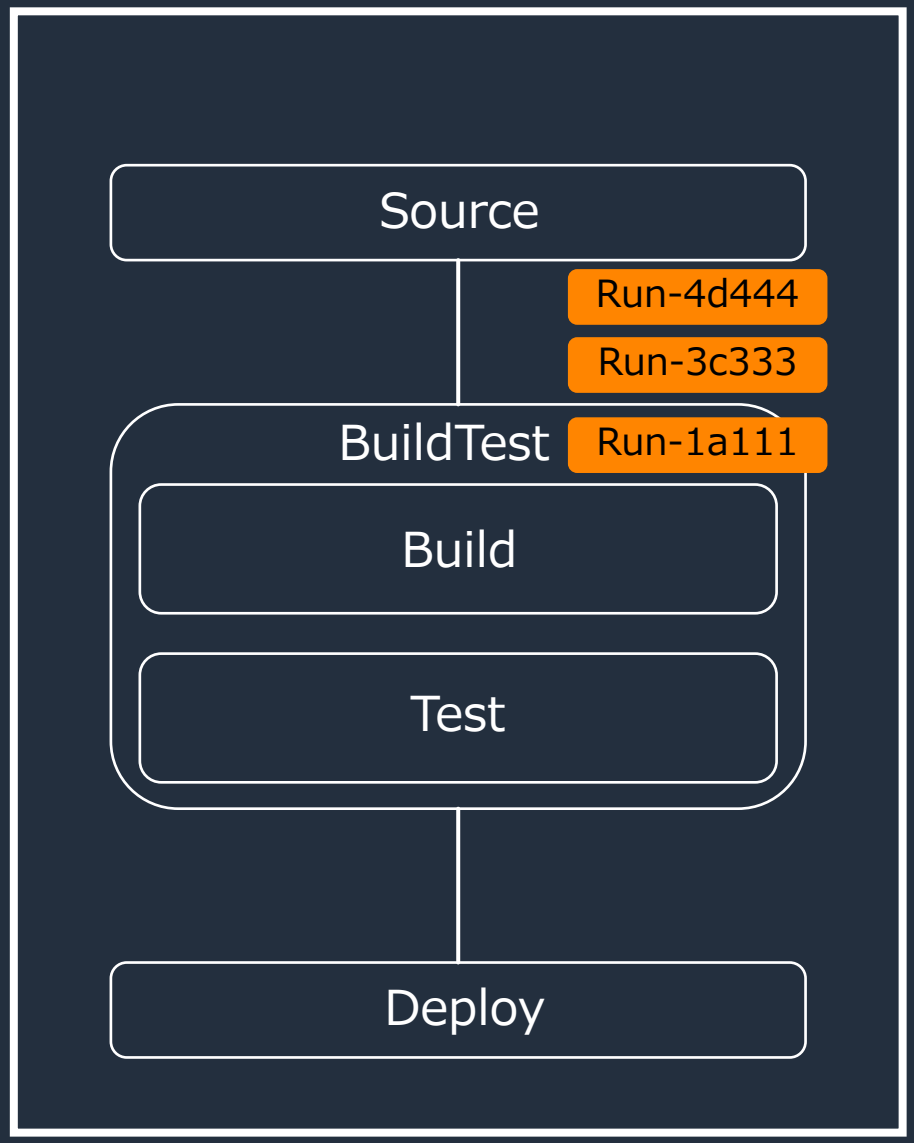
SUPERSEDED の場合、待っている処理は最新の Workflow のみが有効になる

RunMode の動作（SUPERSEDED）



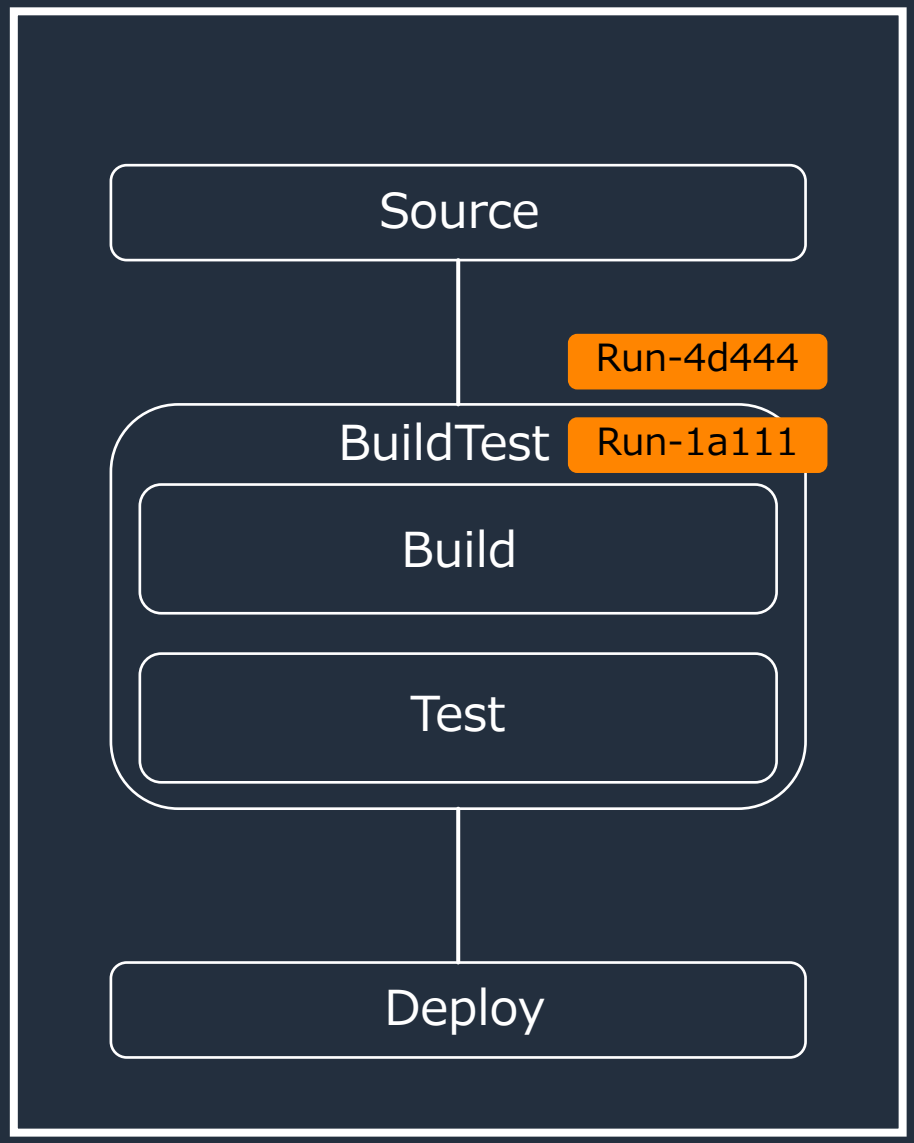
SUPERSEDED の場合、待っている処理は最新の Workflow のみが有効になる

RunMode の動作（SUPERSEDED）



SUPERSEDED の場合、待っている処理は最新の Workflow のみが有効になる

RunMode の動作（SUPERSEDED）



SUPERSEDED の場合、待っている処理は最新の Workflow のみが有効になる

Workflow の構成要素

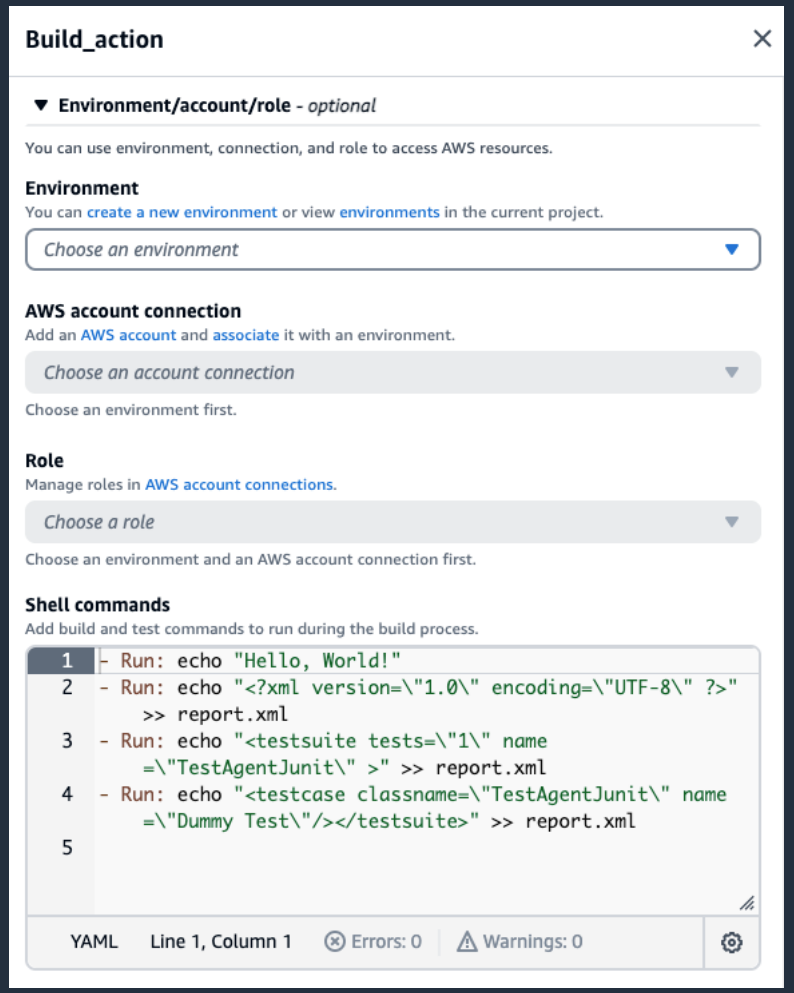
1. 主要な構成要素
2. 代表的な Actions
3. Actions の設定項目

代表的な action の紹介 – Build -

- ソースコードのコンパイル、単体テストの実行、コードカバレッジのチェック、デプロイするために必要なファイル群の出力のために使う

- Steps セクションで以下の形式で記述したコマンドが実行される
  - -Run: *command*

- Output セクションの定義にしたがって、テストレポートも作成できる



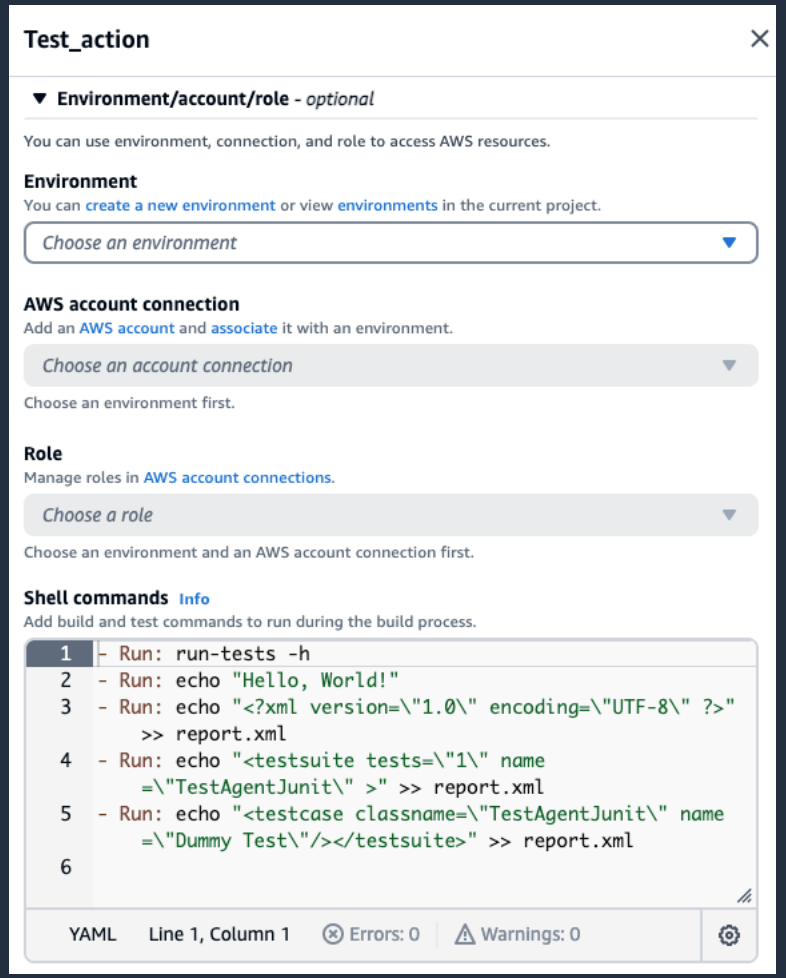
代表的な action の紹介 – Test -

- インテグレーションテストや
  システムテストの実行に使う

- Steps セクションで以下の形式で記述した
  コマンドが実行される
  - -Run: *command*

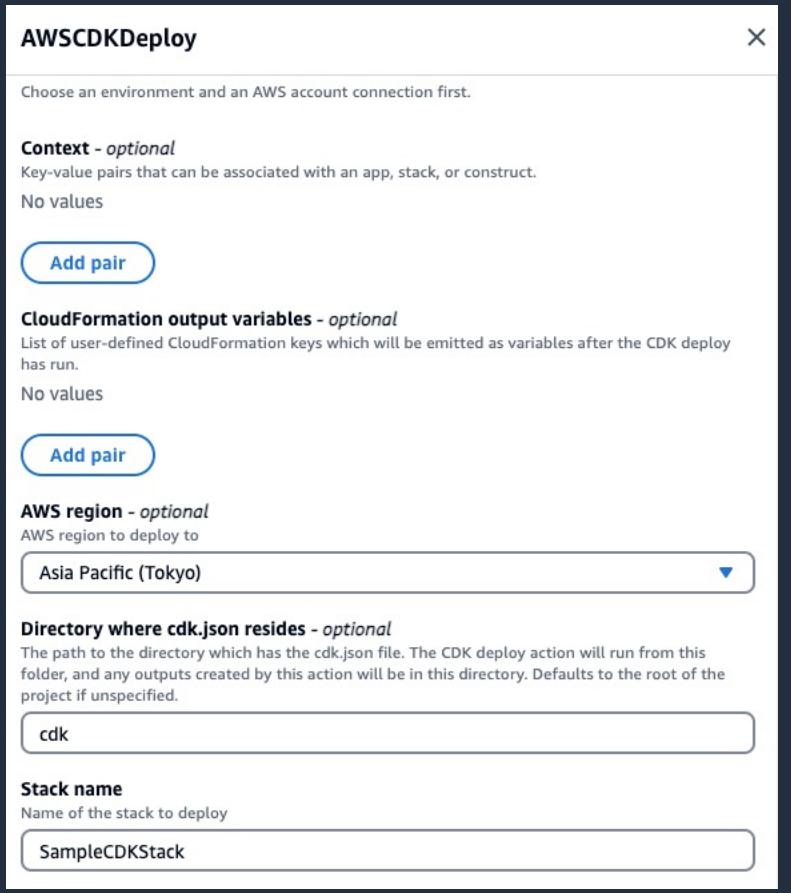
- Output セクションの定義にしたがって、
  テストレポートも作成できる

- 拡張テスト機能（universal-test-runner）が
  利用可能



代表的な action の紹介 – AWS CDK deploy -

- CDK プロジェクトを利用して対象の AWS 環境にスタックをデプロイする
- CDK のすべての開発言語に対応
- AWS アカウント、リージョン、スタック名を指定することで、対象の環境に特定のスタックをデプロイできる
- CDK プロジェクトの場所はデフォルトではリポジトリのルートになるが、プロジェクトのパスを指定できるため、別階層に配置することも可能



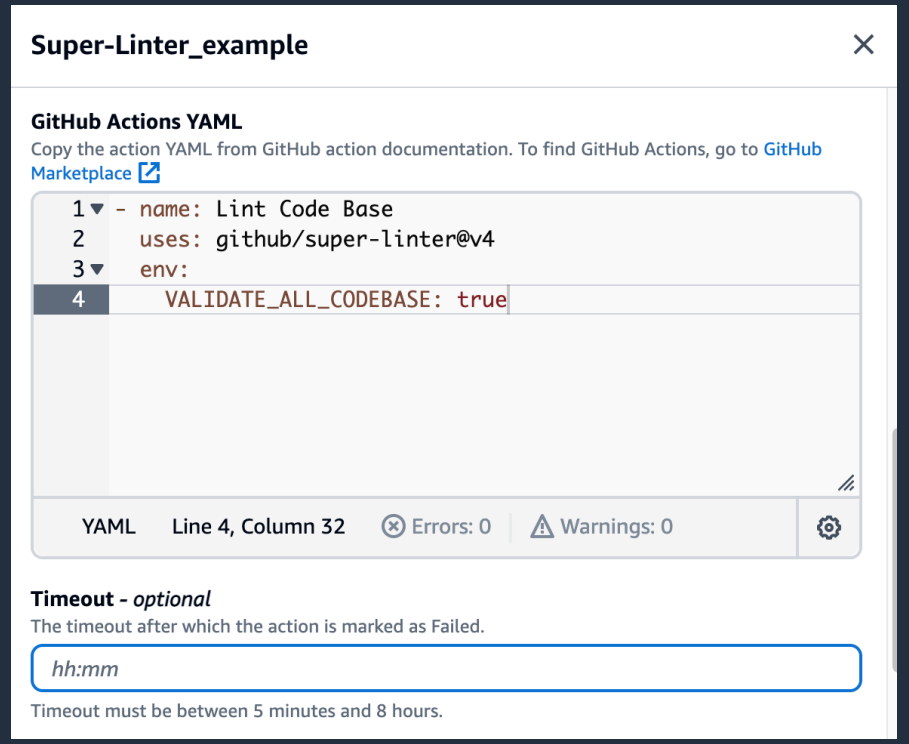
代表的な action の紹介 – GitHub Actions -

- GitHub action を実行する CodeCatalyst action

- Steps セクションに GitHub Marketplace のアクション詳細ページに表示されている GitHub Action のコードを指定する

例） Super Linter の設定

```
Actions:
  GitHubAction:
    Identifier: aws/github-actions-runner@v1
    Configuration:
      Steps:
        - name: Lint Code Base
          uses: github/super-linter@v4
          env:
            VALIDATE_ALL_CODEBASE: "true"
            DEFAULT_BRANCH: main
```



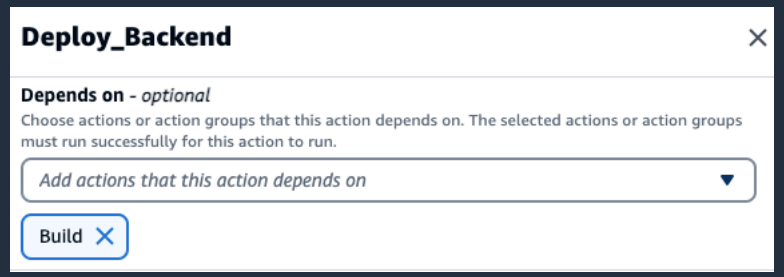
Workflow の構成要素

1. 主要な構成要素
2. 代表的な Actions
3. Actions の設定項目

Actions の設定項目

- Build action・Test action と各種 Deploy 系の action で設定可能な項目が一部異なっている
- YAML ファイルの仕様はユーザーガイドの Workflow definition reference にて確認できる
- ここでは代表的な以下の要素について紹介する
  - DependsOn
  - Input
  - Compute
  - Environment
  - Output
  - Configuration
  - Cache

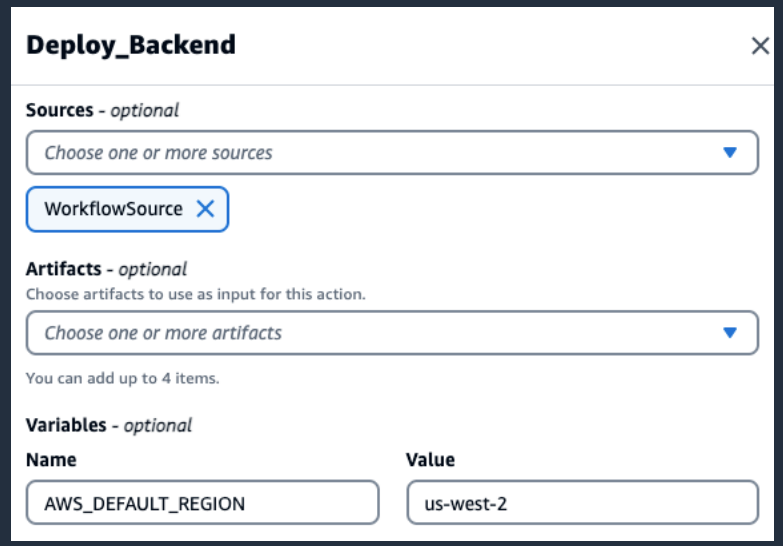
DependsOn




```
Deploy_Backend:
  Identifier: aws/cdk-deploy@v1
  DependsOn:
    - Build
```

- action の前後関係を設定する項目
- 前提となる action を設定することで、Workflow 内の action 実行順序が決定される
- 複数の action を設定することができる

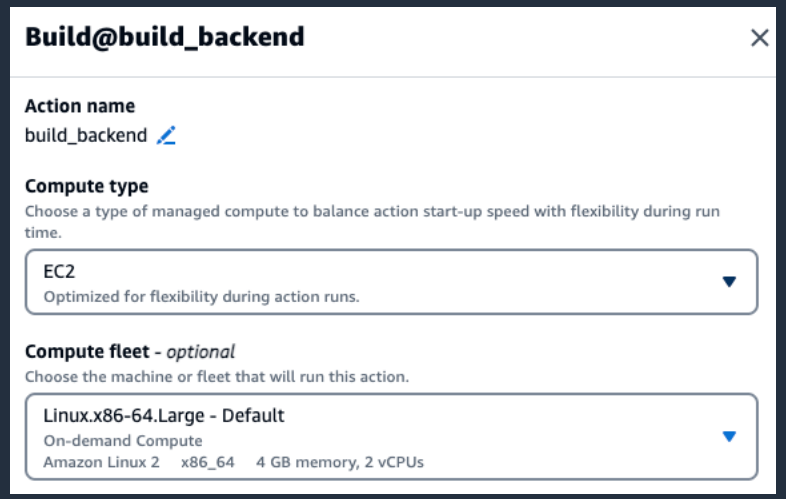
Inputs




```
Inputs:
  Sources:
    - WorkflowSource
  Artifacts:
    - backend
  Variables:
    - Name: AWS_DEFAULT_REGION
      Value: us-west-2
```

- action に対する入力データ
- 以下の設定ができる
  - Sources
    - 参照したい Source repositories
  - Artifacts
    - 参照したい別の action の出力結果
  - Variables
    - action 内で利用できる変数の名前と値

Compute




```
build_backend:
  Identifier: aws/build@v1
  Compute:
    Type: EC2
    Fleet: Linux.x86-64.Large
```

- Action 内の設定項目で、 Action の実行環境（CPU、メモリ、OS）のこと
- Compute には以下の 2種類がある
  - Amazon EC2
  - AWS Lambda
- オンデマンドとプロビジョンドが選べる
  - オンデマンド＝Workflow 実行のたびに都度作成される
  - プロビジョンド＝プロジェクト専用の環境が常にアイドル状態で待機し、即座に処理を実行できる
- 選択できる実行環境のスペックも選択可能

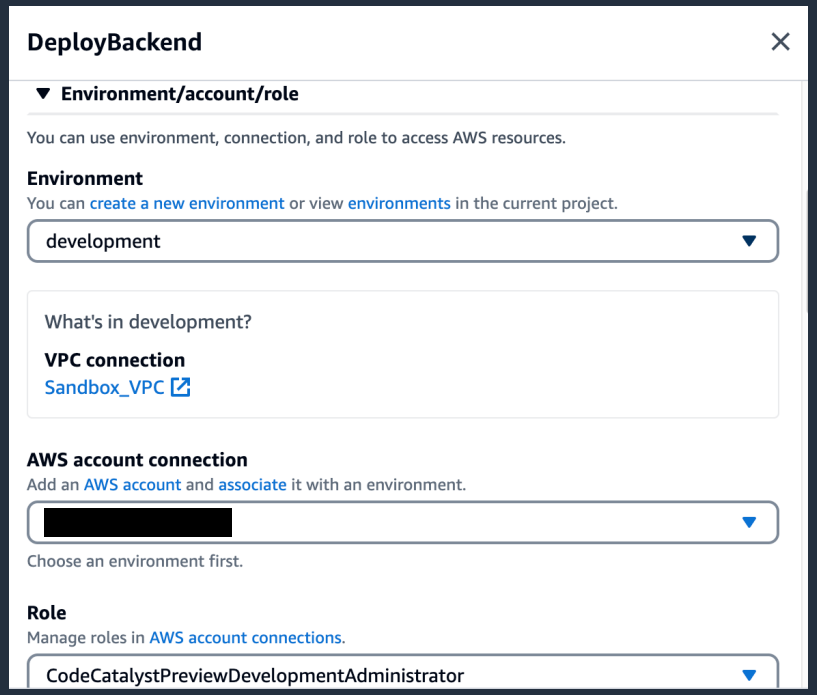
参考：オンデマンドフリートで選択できる環境

Name	Operating system	Architecture	vCPUs	Memory (GiB)	Disk space	Supported compute types
Linux.Arm64.Large	Amazon Linux 2	Arm64	2	4	64 GB	Amazon EC2
					10 GB	Lambda
Linux.Arm64.XLarge	Amazon Linux 2	Arm64	4	8	128 GB	Amazon EC2
					10 GB	Lambda
Linux.Arm64.2XLarge	Amazon Linux 2	Arm64	8	16	128 GB	Amazon EC2
Linux.x86-64.Large	Amazon Linux 2	x86-64	2	4	64 GB	Amazon EC2
					10 GB	Lambda
Linux.x86-64.XLarge	Amazon Linux 2	x86-64	4	8	128 GB	Amazon EC2
					10 GB	Lambda
Linux.x86-64.2XLarge	Amazon Linux 2	x86-64	8	16	128 GB	Amazon EC2

※ fleet を選択しなかった場合は Linux.x86-64.Large が採用される

https://docs.aws.amazon.com/codecatalyst/latest/userguide/compute.html#compute.on-demand

Environment

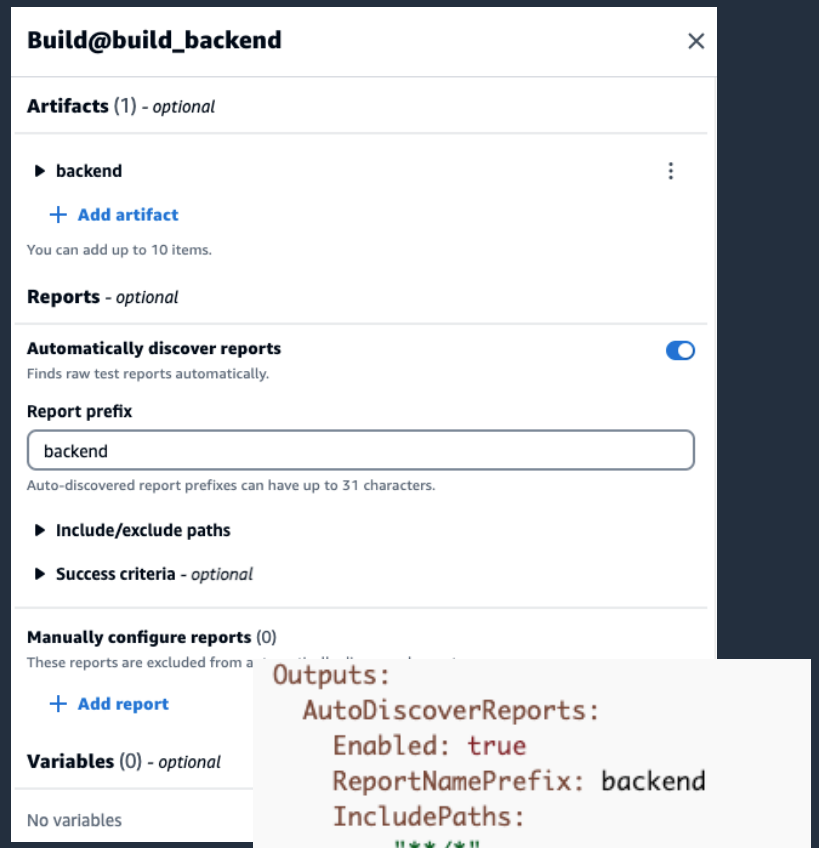



```
Environment:
  Connections:
    - Role: CodeCatalystWorkflowDevelopmentRole
      Name: [redacted]
  Name: development
```

- action 内の設定項目で、 デプロイ先となる AWS 環境のこと
- Environment の登録画面にて設定を作成し、Workflow で選択する
  - AWS アカウント ID と IAM Role を関連付けて選択することで、デプロイ先と権限が決まる
  - Environment に VPC 接続が設定されている場合は VPC にも接続される

※ Environment については Identity, permissions, and access 編に詳細な記載があるので、詳しく知りたい場合はそちらを参照

Outputs




```
Outputs:
  AutoDiscoverReports:
    Enabled: true
    ReportNamePrefix: backend
    IncludePaths:
      - "**/*"
    SuccessCriteria:
      PassRate: 100
      BranchCoverage: 50
      LineCoverage: 70
      Vulnerabilities:
        Severity: CRITICAL
  Artifacts:
    - Name: backend
      Files:
        - "**/*"
```

- 当該 action で残したいの実行結果の設定
  - 後続の action に引き渡したいフォルダー・ファイル・変数
  - テストレポート
- 以下の設定ができる
  - Artifacts
    - 後続の action で利用したいフォルダー・ファイル
  - Variables
    - 後続の action で利用できる変数の名前と値
  - AutoDiscoverReports／Reports
    - CodeCatalyst コンソールで表示させたいテスト・コードカバレッジ・ソフトウェア構成解析（SCA）のレポート

サポートされるレポートの種類

- テスト
  - Cucumber JSON (.json)
  - JUnit XML (.xml)
  - NUnit XML (.xml)
  - NUnit3 XML (.xml)
  - TestNG XML (.xml)
  - Visual Studio TRX (.trx, .xml)
- コードカバレッジ
  - JaCoCo XML (.xml)
  - SimpleCov JSON (.json)
  - Clover XML (version 3, .xml)
  - Cobertura XML (.xml)
  - LCOV (.info)
- ソフトウェア構成解析（SCA）
  - SARIF (.sarif, .json)
- 静的解析
  - PyLint (.py)
  - ESLint (.js, .jsx, .ts, .tsx)
  - SARIF (.sarif, .json)

https://docs.aws.amazon.com/codecatalyst/latest/userguide/test-workflow-actions.html

Configuration


```
Configuration:
  Steps:
    - Run: pip3 install -r requirements-dev.txt
    - Run: python3 -m coverage run --branch --omit "*/tests/**,*/opt/**" -m pytest
        -k
        unit --junitxml=unitTests.xml
    - Run: python3 -m coverage xml
  Container:
    Registry: ECR
    Image: 111122223333.dkr.ecr.us-east-2.amazonaws.com/image-name
```

- 当該 action のプロパティ設定
- 設定内容は action ごとに異なる

Cache

レイテンシーを減らすために実行結果をキャッシュし、再利用する

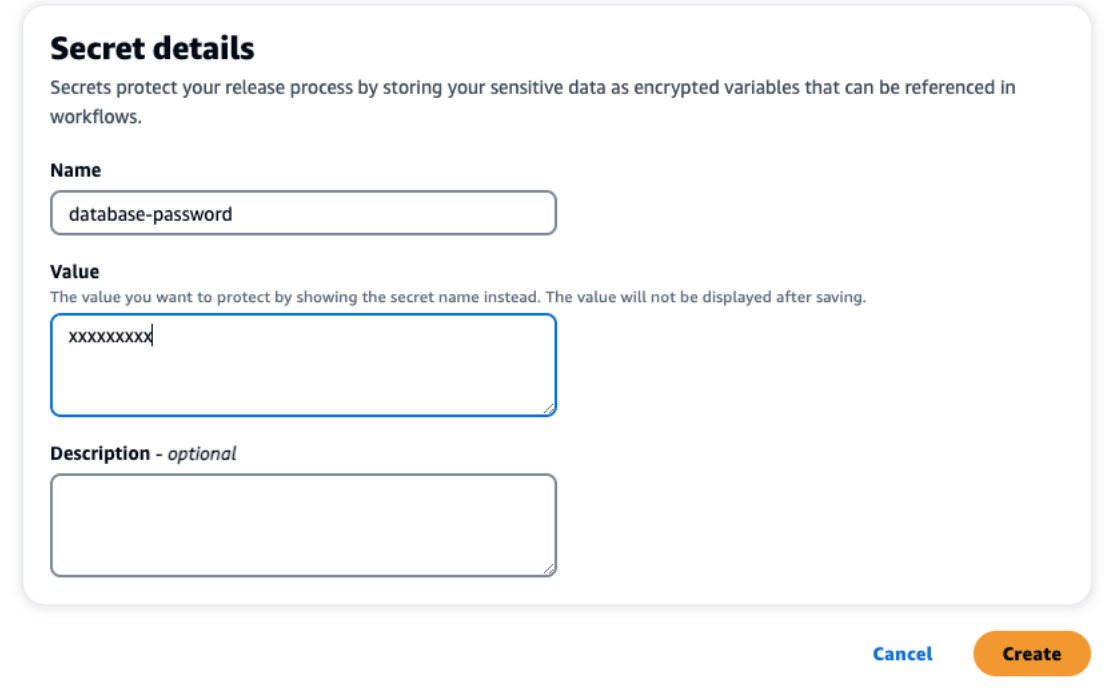
```
Actions:
  BuildMyNpmApp:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: npm install
        - Run: npm run test
    Caching:
      FileCaching:
        cacheKey1:
          Path: file1.txt
          RestoreKeys:
            - restoreKey1
        cacheKey2:
          Path: /root/repository
          RestoreKeys:
            - restoreKey2
            - restoreKey3
```

- action 終了時に Path に指定したファイル・ディレクトリをキャッシュ
- ワークフローとブランチごとにキャッシュは分離される
- キャッシュはデフォルトで最後に使用されてから 14日後に有効期限切れとなる
- build action と test action のみで使用できる
- Compute Type は EC2 のみが利用可能

https://docs.aws.amazon.com/codecatalyst/latest/userguide/workflows-caching.html

Secrets

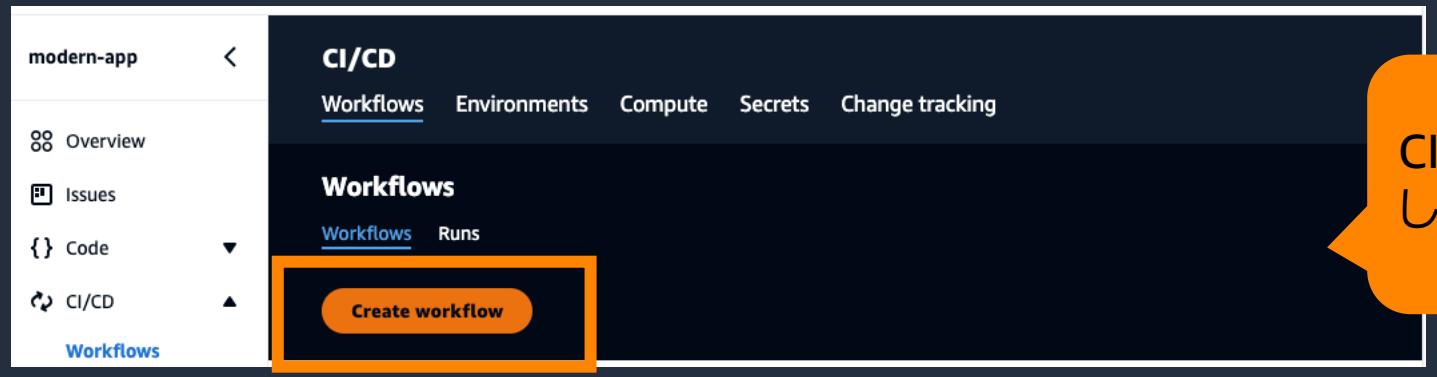
認証情報のような機密データを Workflow 内で使用する場合に、値を保護することができる



- Secret の識別子は ${Secrets.*name*} という形式になる
  - 左の例だと ${Secrets.database-password}
- YAML 内にて、具体値の代わりに Secrets の識別子を記述することで、機密データを Workflow 内で使用できる
- 登録した値をコンソールで確認することはできないことに注意

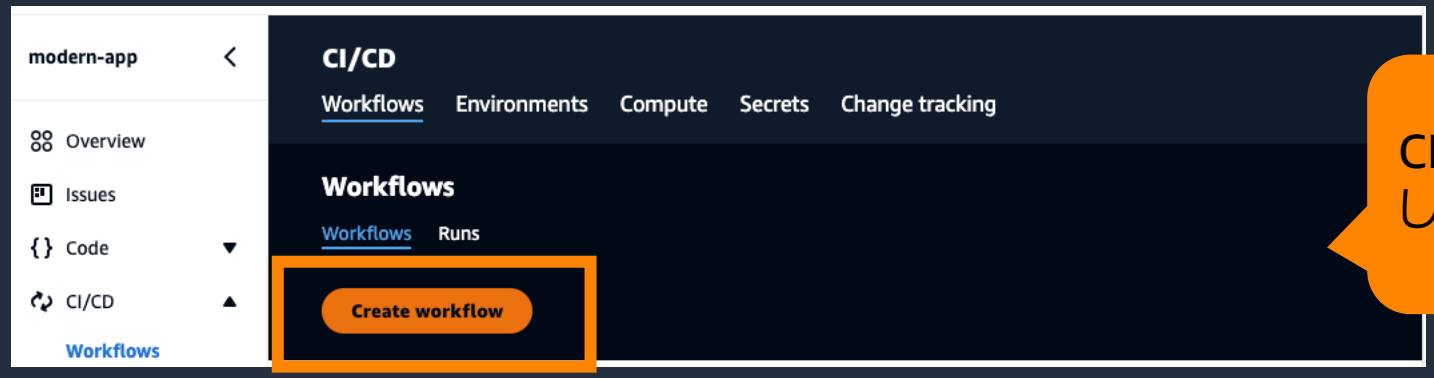
Workflow の作成方法

Workflow の追加



① 左のメニューから CI/CD → Workfow を選択し、Create workflow ボタンを押す

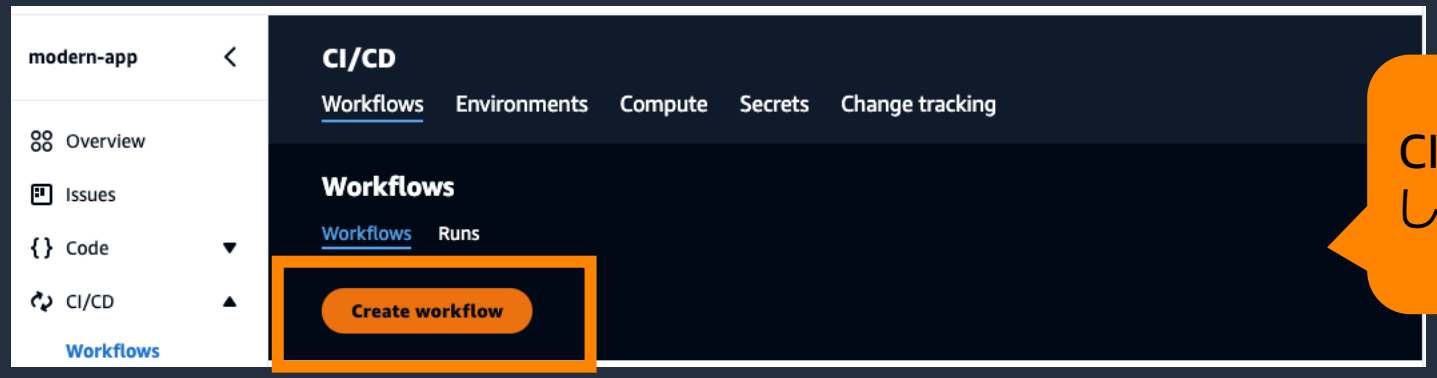
Workflow の追加



① 左のメニューから CI/CD → Workfow を選択し、Create workflow ボタンを押す

② リポジトリとブランチを選択

Workflow の追加



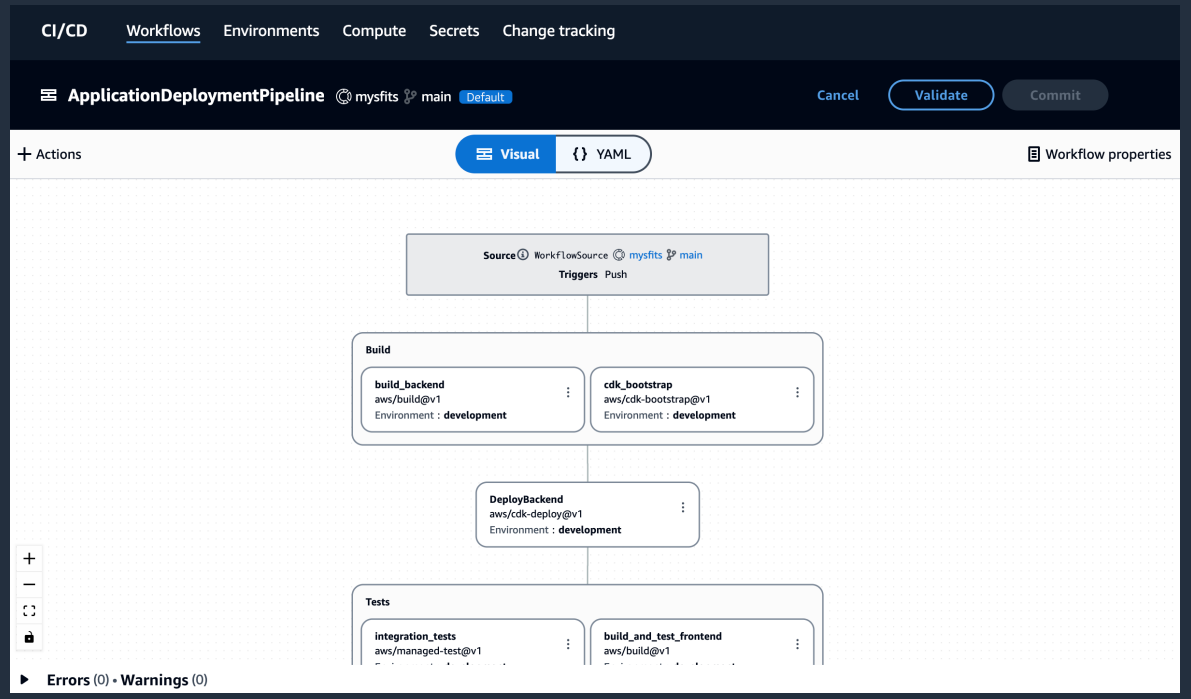
① 左のメニューから CI/CD → Workfow を選択し、Create workflow ボタンを押す

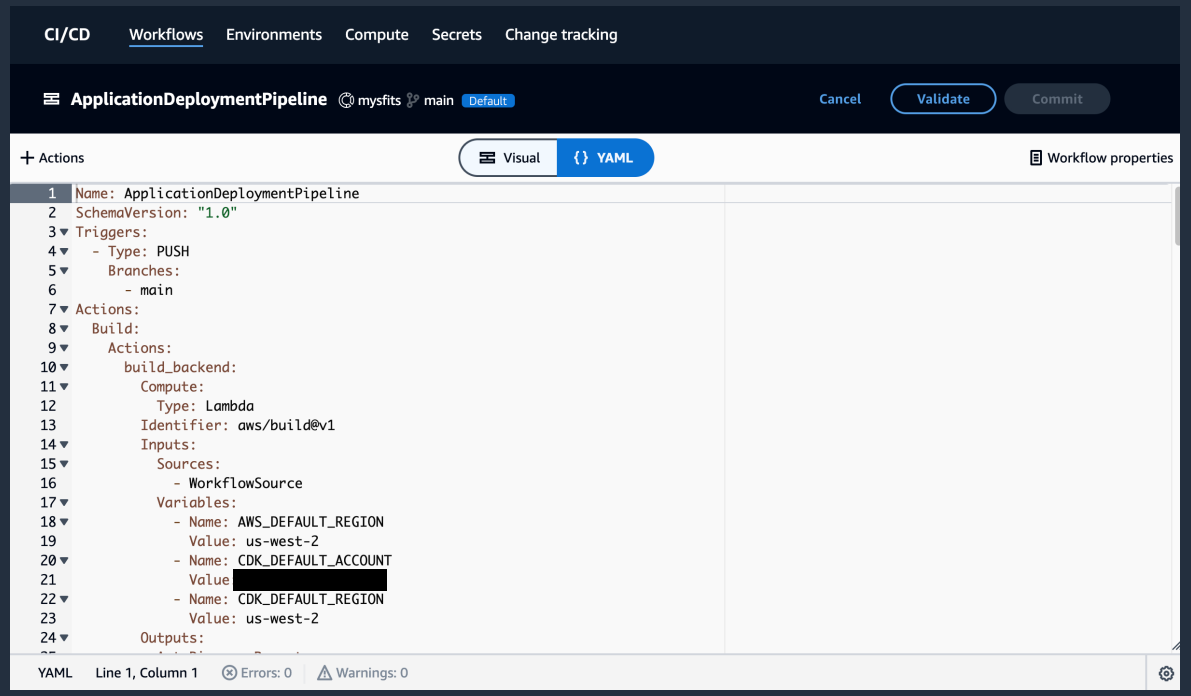
② リポジトリとブランチを選択

③ 空の Workflow が表示される

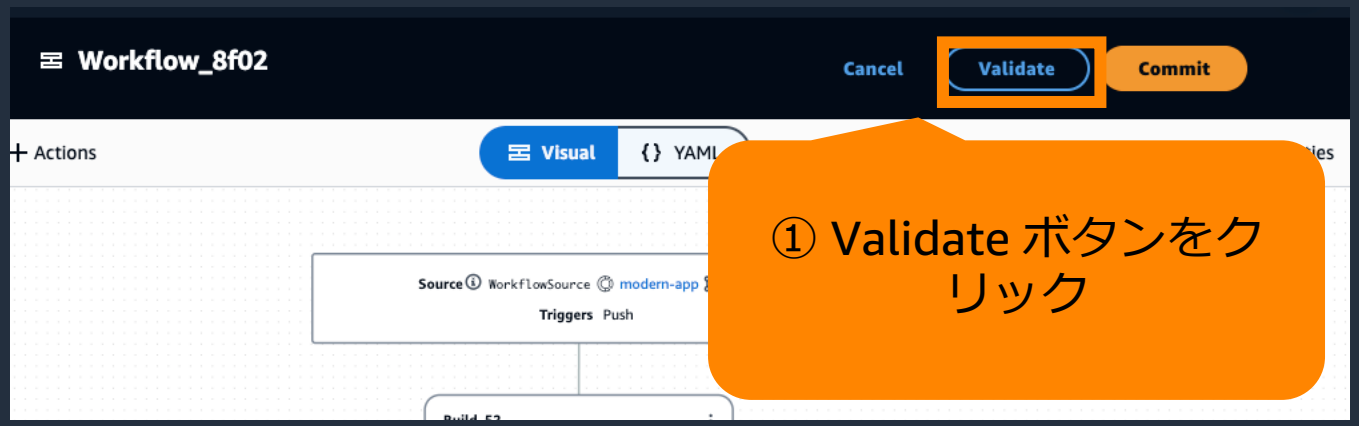
Workflow 定義の編集

- GUI（Visual タブ）と CUI（YAML タブ）のどちらでも編集可能
- GUI で作成した後で CUI で編集することや、その逆もできる

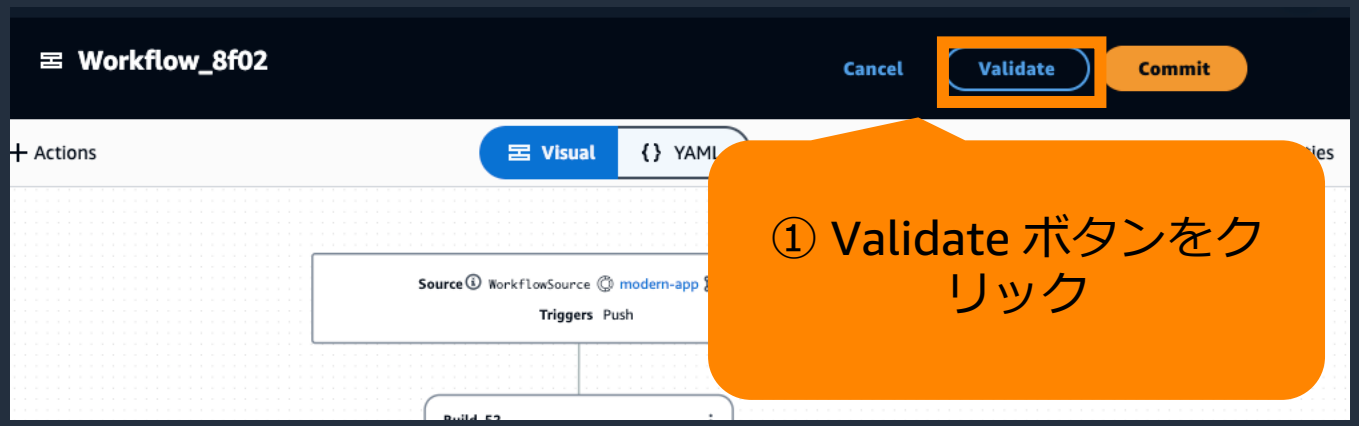




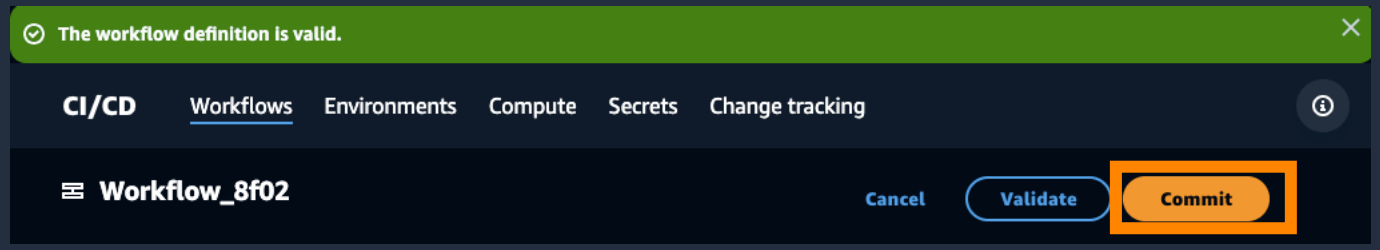
Workflow の保存



Workflow の保存

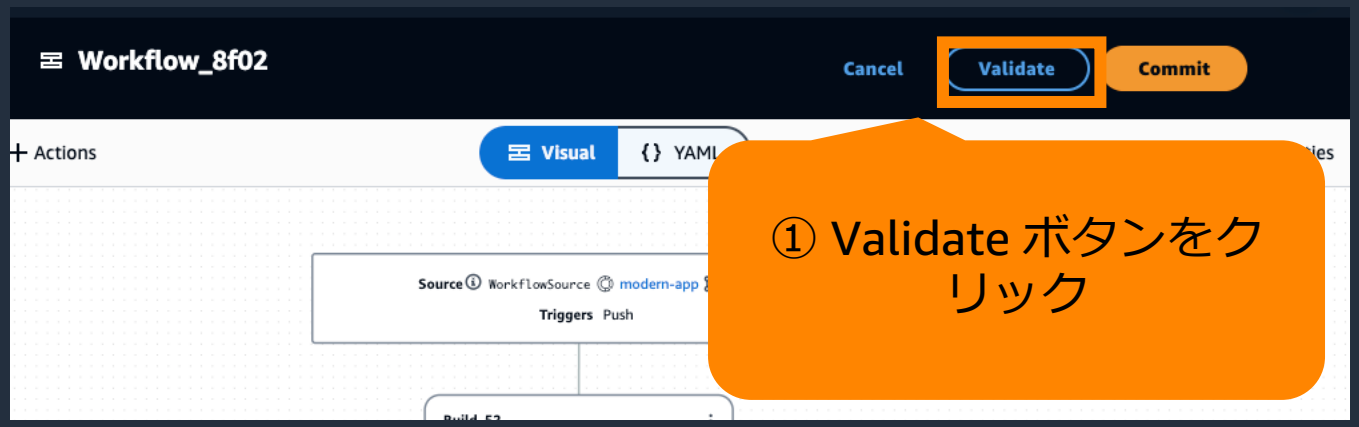


① Validate ボタンをクリック

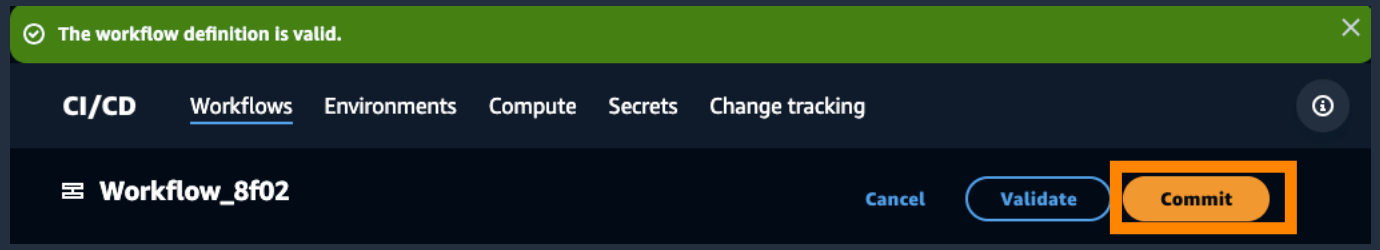


② Commit ボタンをクリック

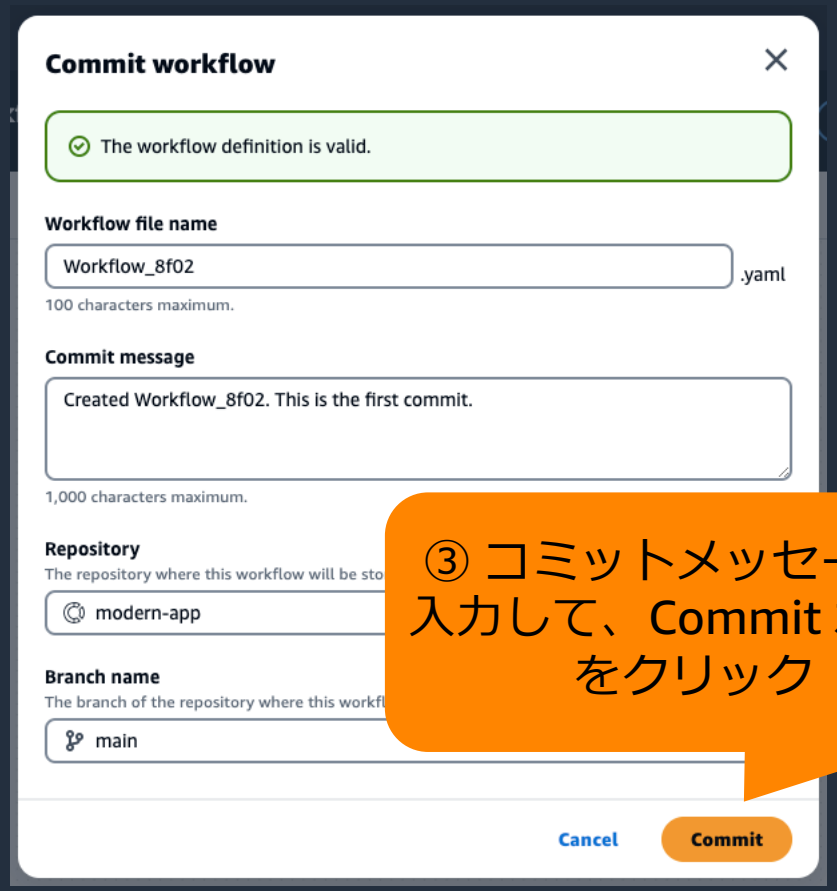
Workflow の保存



① Validate ボタンをクリック



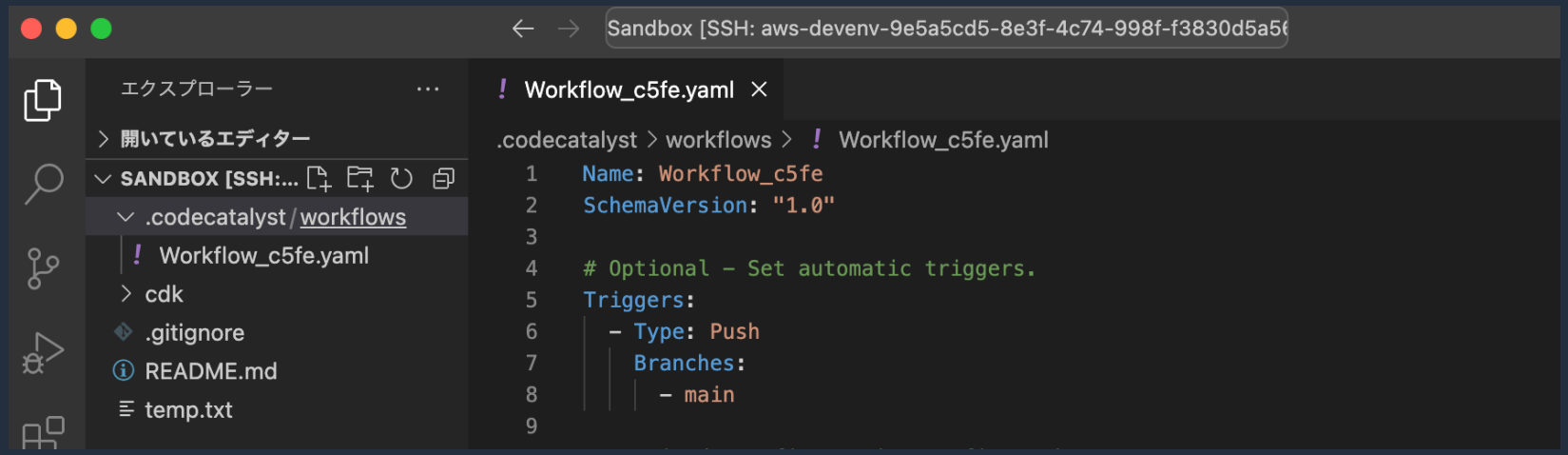
② Commit ボタンをクリック



③ コミットメッセージを入力して、Commit ボタンをクリック

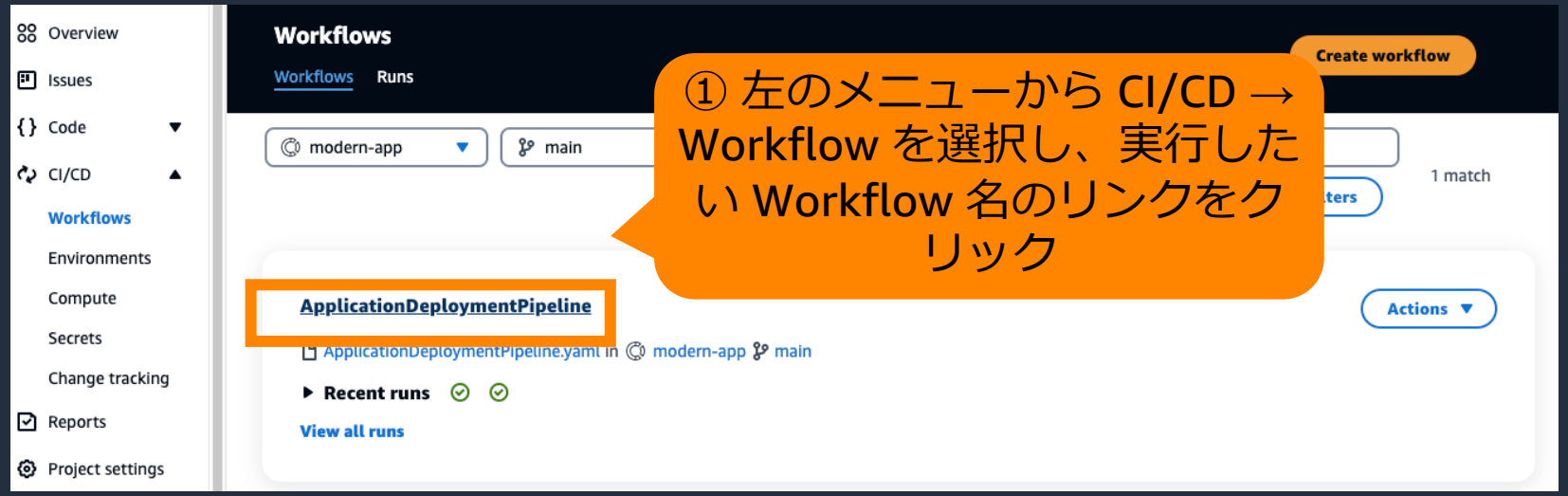
Workflow ファイルを直接編集

- .codecatalyst/workflows/ 配下に Workflow の YAML ファイルが配置される
- 直接編集して、リポジトリにアップロードすることで Workflow の更新が可能
- ファイルを追加することで、Workflow の追加も可能
- アプリケーション・インフラリソース・パイプライン定義すべてが
  一つのリポジトリで管理でき、リポジトリへのアップロードでデプロイできる



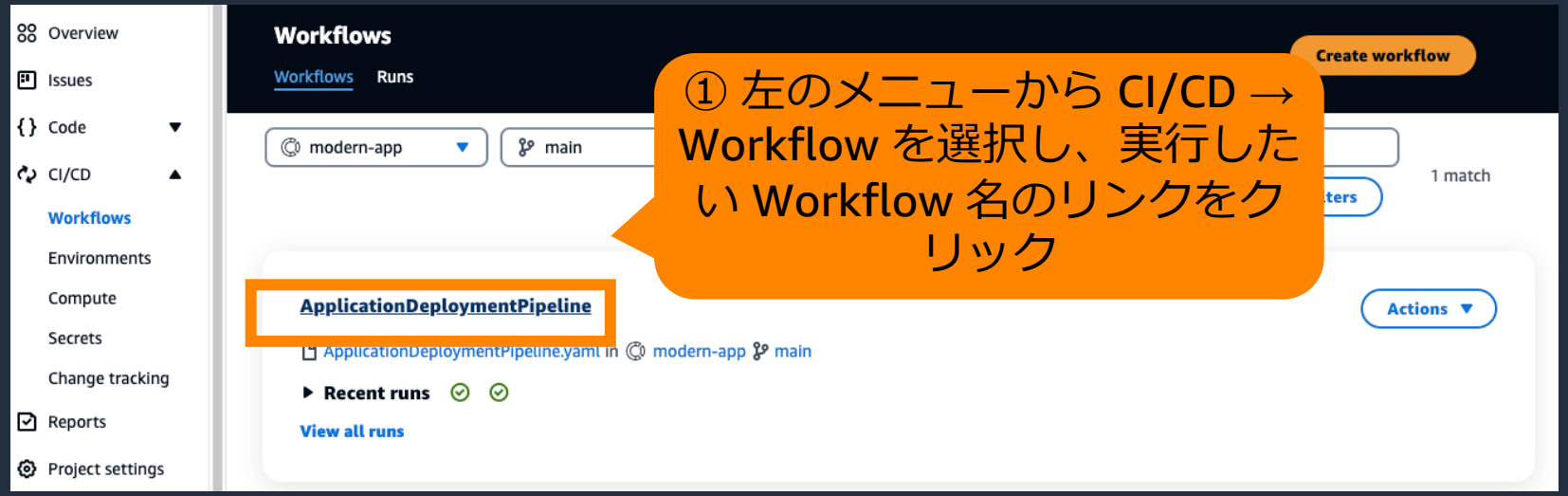
Workflow の実行・管理方法

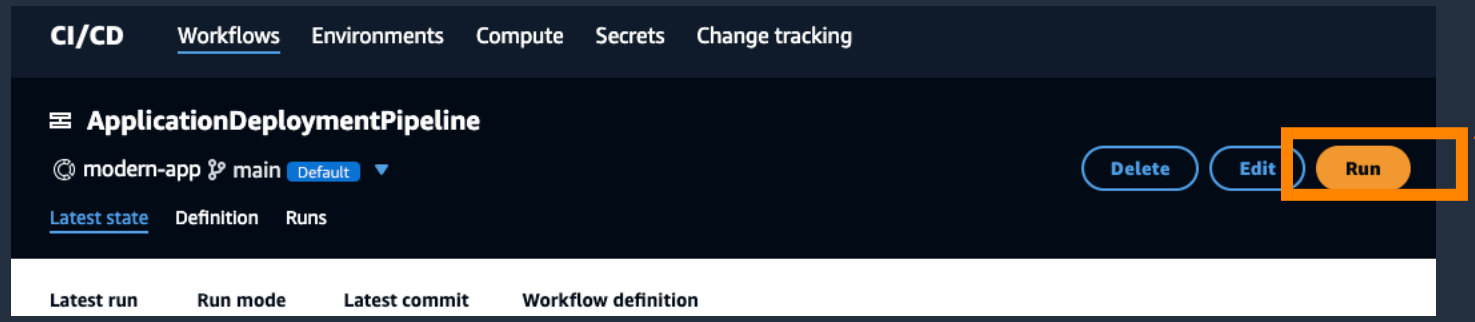
Workflow の手動実行



※ ここでは手動実行の方法を紹介しているが、Workflow 作成時に指定した Triggers の条件（リポジトリへの Push など）や、スケジュール実行も可能

Workflow の手動実行



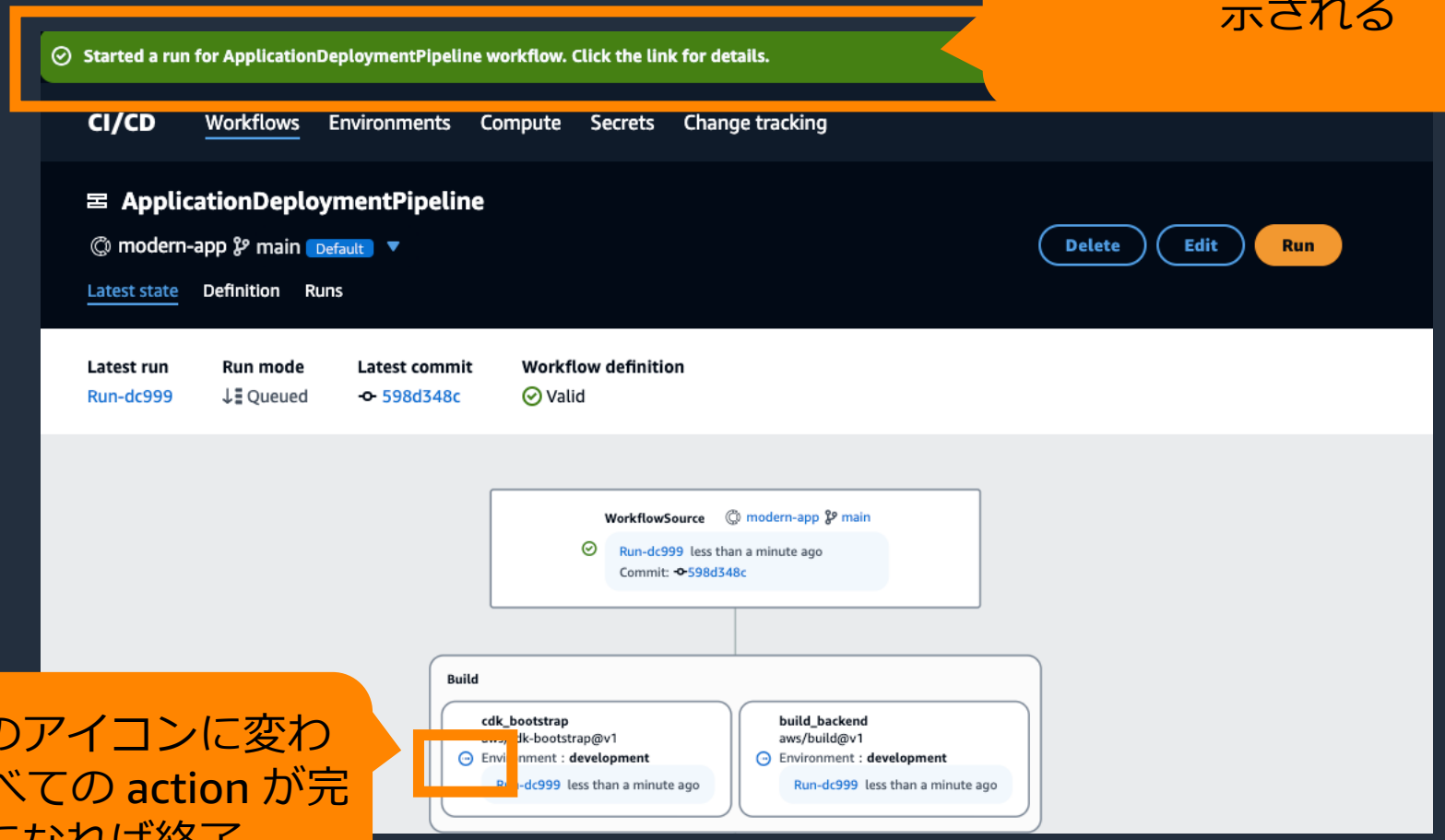


② Run ボタンをクリック

※ ここでは手動実行の方法を紹介しているが、Workflow 作成時に指定した Triggers の条件（リポジトリへの Push など）や、スケジュール実行も可能

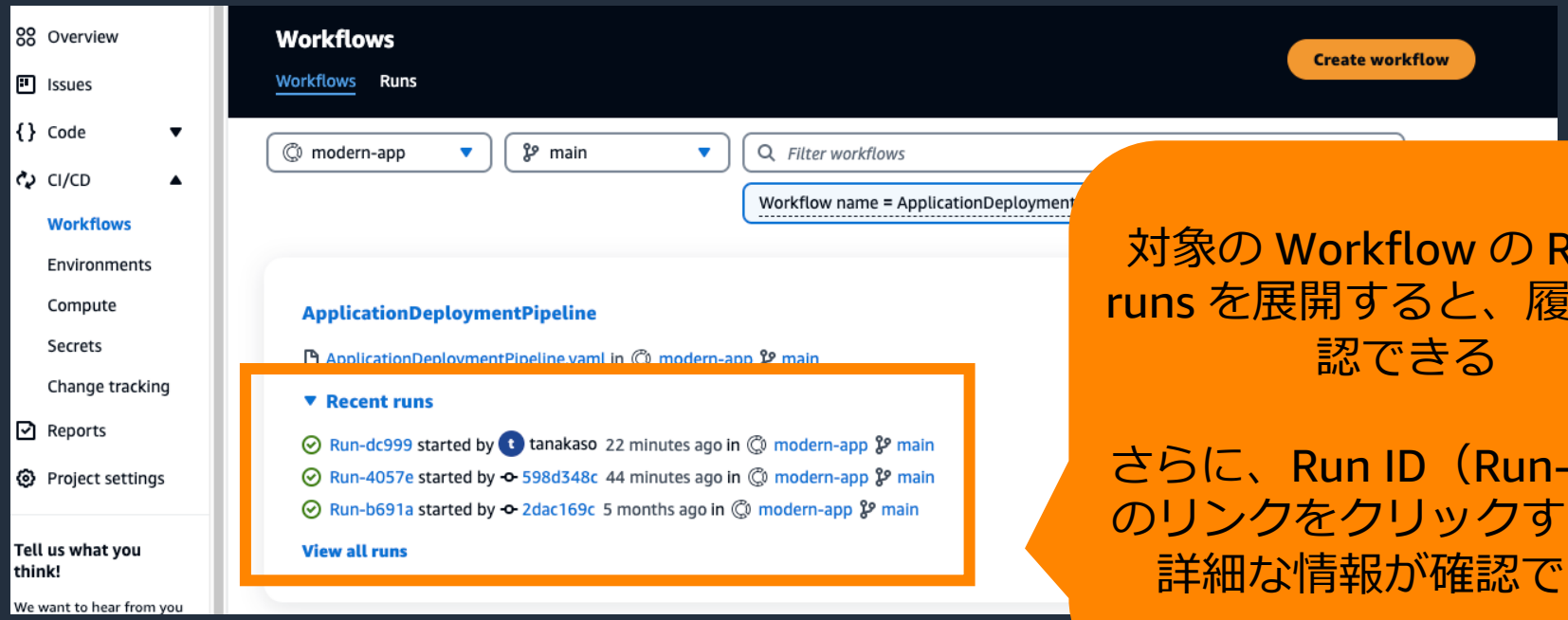
Workflow の手動実行

Workflow 開始メッセージが表示される



実行中のアイコンに変わる → すべての action が完了になれば終了

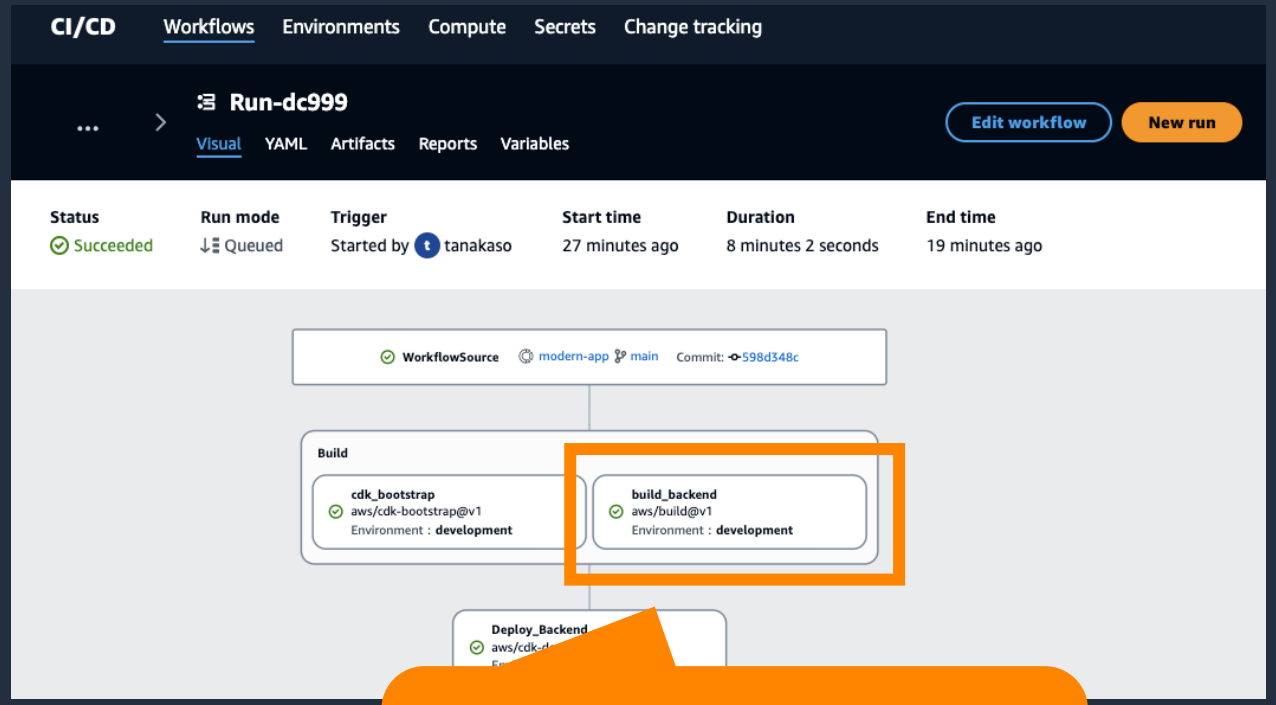
Workflow の実行結果の確認



対象の Workflow の Recent runs を展開すると、履歴が確認できる

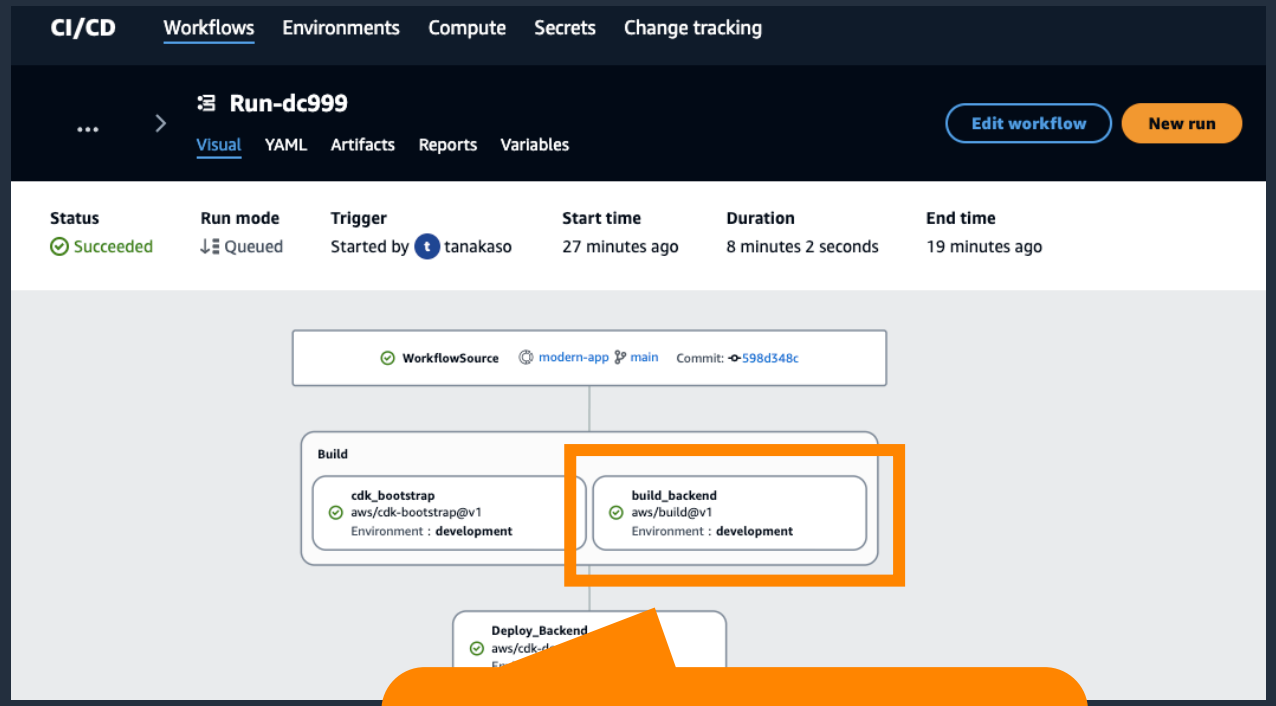
さらに、Run ID（Run-xxxx）のリンクをクリックすると、詳細な情報が確認できる

Workflow の実行結果の確認



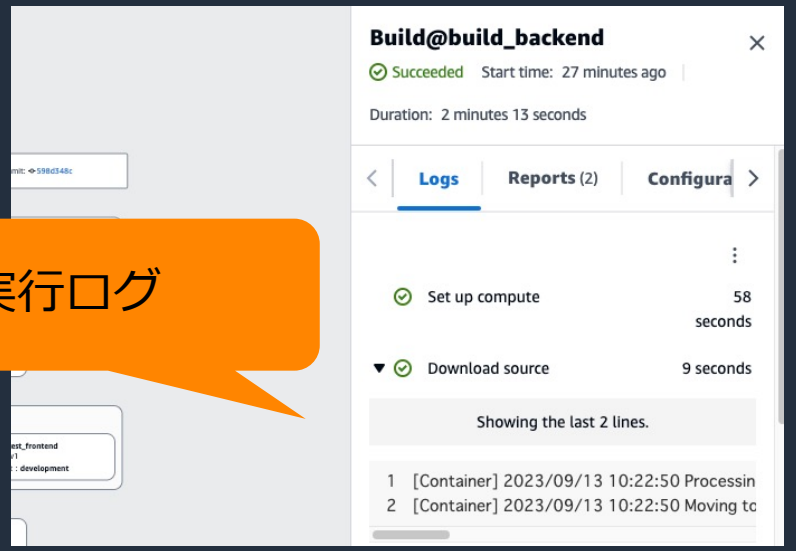
詳細を確認したい action をクリックすると、右側に実行結果ビューが表示される

Workflow の実行結果の確認

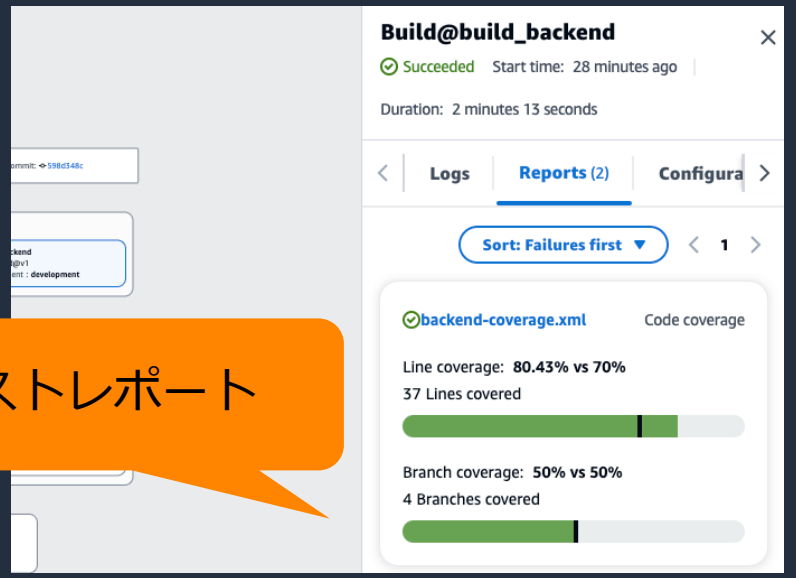


詳細を確認したい action をクリックすると、右側に実行結果ビューが表示される

実行ログ



テストレポート



Actions のベストプラクティス

機密情報の取り扱い

- 機密情報を YAML に埋め込まない
- クレデンシャルやキー、トークンを YAML に埋め込むのではなく、CodeCatalyst のシークレットを使うことを推奨
- シークレットを使えば、機密情報を YAML 内に保存したり参照したりするのが簡単になる

GitHub Actions の取り扱い

- GitHub Actions のソースコードは利用者の責任範囲となる
- 実行する GitHub Actions の信頼性とセキュリティに問題がないことを確認しておく
  - GitHub Actions は、シークレット、リポジトリトークン、ソースコード、アカウントリンク、コンピュート時間へのアクセスが許可されている

Workflow のクォータ

Workflow のクォータ（1/2）

項目	上限
Space あたりの Workflow 数	800
Workflow 定義ファイルのサイズ	256KB
一つのイベントから実行される Workflow の数	50
一つのイベントから実行されるファイルの数	4,000
Space あたりの active な fleet 数	10
fleet ごとの active な compute インスタンス数	20
action ごとの input artifact の数	10
action ごとの output artifact の数	10
一つの action の output variable の合計サイズ	120KB
output variable の最大長	500 以上（action による） ※ 制限値以上の値は切り捨てられることもある
Workflow が実行されて生成された artifact の保存期間	30日
action ごとのレポートの数	50

Workflow のクォータ（2/2）

項目	上限
テストレポートあたりのテストケース数	20,000
カバレッジレポートあたりのファイル数	20,000
レポートごとのソフトウェア構成解析の検出数	20,000
静的解析レポートごとのファイル数	20,000
Space あたりの Workflow の同時実行数	100
Workflow あたりの同時実行 action 数	50
Space あたりの同時実行 action 数	200
Workflow の実行時間	build・test action は 8時間 その他の action は 1時間
Space あたりの secrets の数	500,000

※ クォータの全項目リストは以下のドキュメントを参照
https://docs.aws.amazon.com/codecatalyst/latest/userguide/workflows-quotas.html

まとめ

まとめ

- CodeCatalyst の Workflow では CI/CD パイプラインと、そこで実行されるビルド・テストの設定も一つの定義で一元管理
- GUI でも CUI でも編集可能
- CodeCatalyst で用意された action 以外に GitHub Actions など 3rd Party 製の action も使える
- 実行履歴、個々の action のログ、テストレポートも統合管理

Thank you!