CircleCIでCI/CDを始めよう！開発者向けチュートリアル

コードをプッシュすると、数分後には自動でテストが完了し、結果が手元に届く。問題があればAIがエラーの原因を分析し、修正方法を提案してくれる。複数のテストが並列で実行され、フィードバックは最速で得られる。

これが、CircleCIで実現できる開発サイクルです。

本チュートリアルでは、こうした自動化された開発環境を、ゼロから構築する手順をハンズオン形式で解説します。アカウント作成から始めて、GitHubリポジトリとの連携、パイプラインの構築、テスト結果の可視化、並列実行による高速化まで、実務で使える機能を段階的に体験できます。

準備するものは、GitHubアカウントとCircleCIアカウント（どちらも無料）のみです。デモ用のReactアプリケーションを用意していますので、CI/CDが初めての方でもすぐに始められます。

1. 事前準備

本チュートリアルをスムーズに進めるために、まずは以下の2点を準備しましょう。いずれも無料でアカウントを作成・利用でき、すぐに開発自動化の第一歩を踏み出せます。

- GitHub無料アカウント
- CircleCI無料アカウント:

1-1:GitHubアカウントのセットアップ

まずGitHubアカウントを用意しましょう。 https://github.com/signup から無料で作成できます。必要なものはメールアドレスとパスワードだけで、クレジットカードの登録なども不要です。詳細は GitHub のドキュメントをご確認ください。

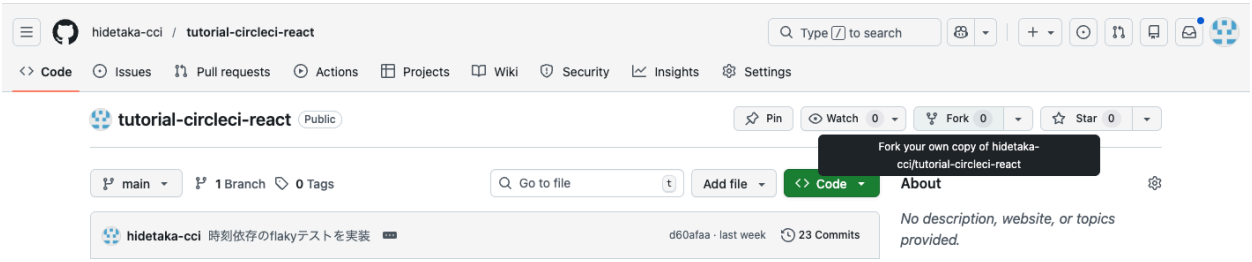
> GitHub でアカウントを作成する
> https://docs.github.com/ja/get-started/start-your-journey/creating-an-account-on-github

1-2: デモ用のプロジェクトをforkする

アカウントの準備ができましたので、CIを設定する対象となるプロジェクトを用意しましょう。以下のURLにシンプルな React アプリを用意しています。

https://github.com/hidetaka-cci/tutorial-circleci-react

リポジトリにアクセスし、Forkボタンをクリックしてください。「あなたのGitHubユーザー名/tutorial-circleci-react」でリポジトリが用意できたら、準備完了です。



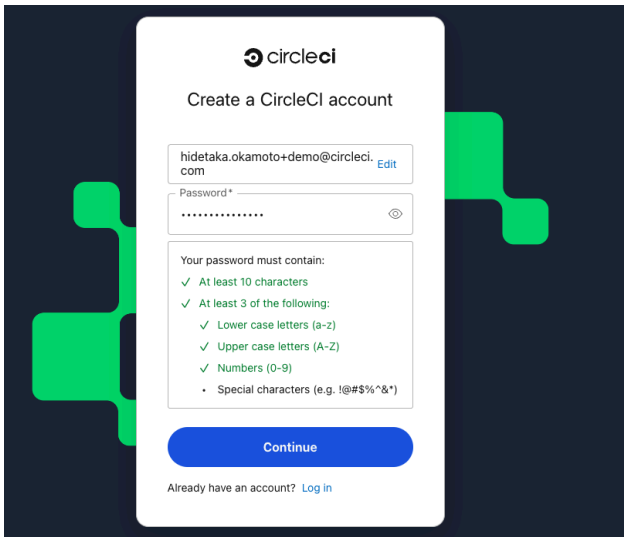
これで必要なアカウントやアプリケーションの準備ができました。次は最初のCIパイプラインの実行準備を行います。

1-3:CircleCIアカウントを作成する

続いてCircleCIアカウントを作成します。CircleCIアカウントを作成する際、作成いただいたGitHubアカウントを利用することで、パスワードの設定やアカウント作成後のGitリポジトリ接続設定などを省略できます。

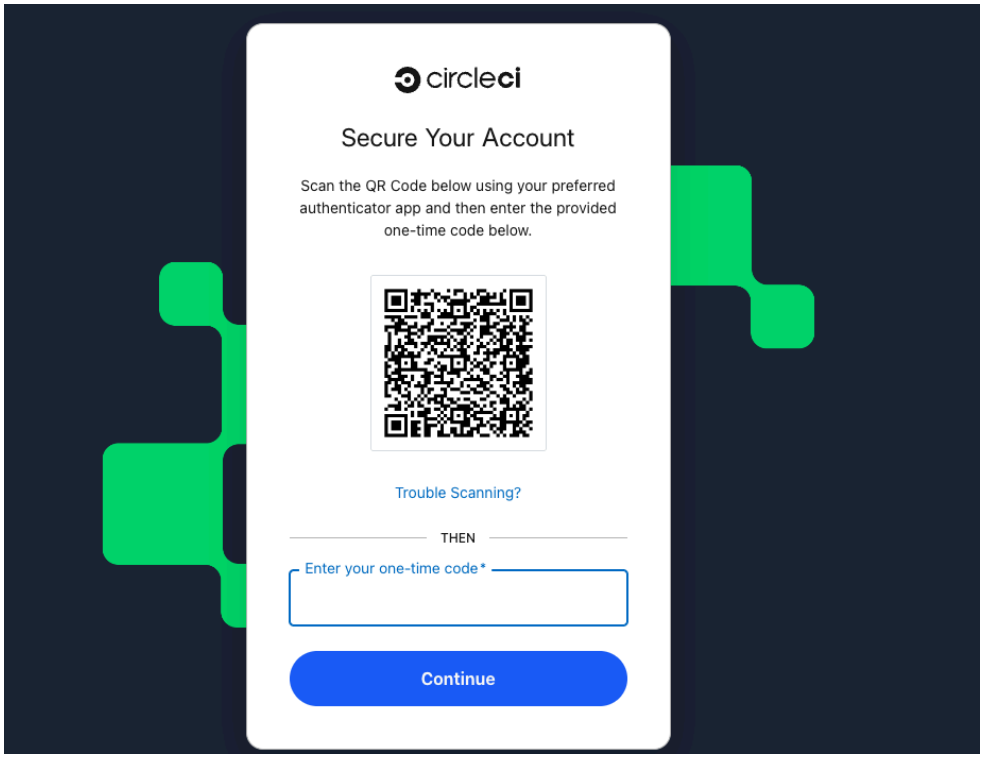
CircleCIの日本語トップページ（ https://circleci.com/ja/ ）にアクセスします。ページ上部の「無料で始める」ボタンをクリックすることでサインアップページに移動できます。

まずメールアドレスを入力しましょう。その後パスワードの入力を求められますので、入力してください。

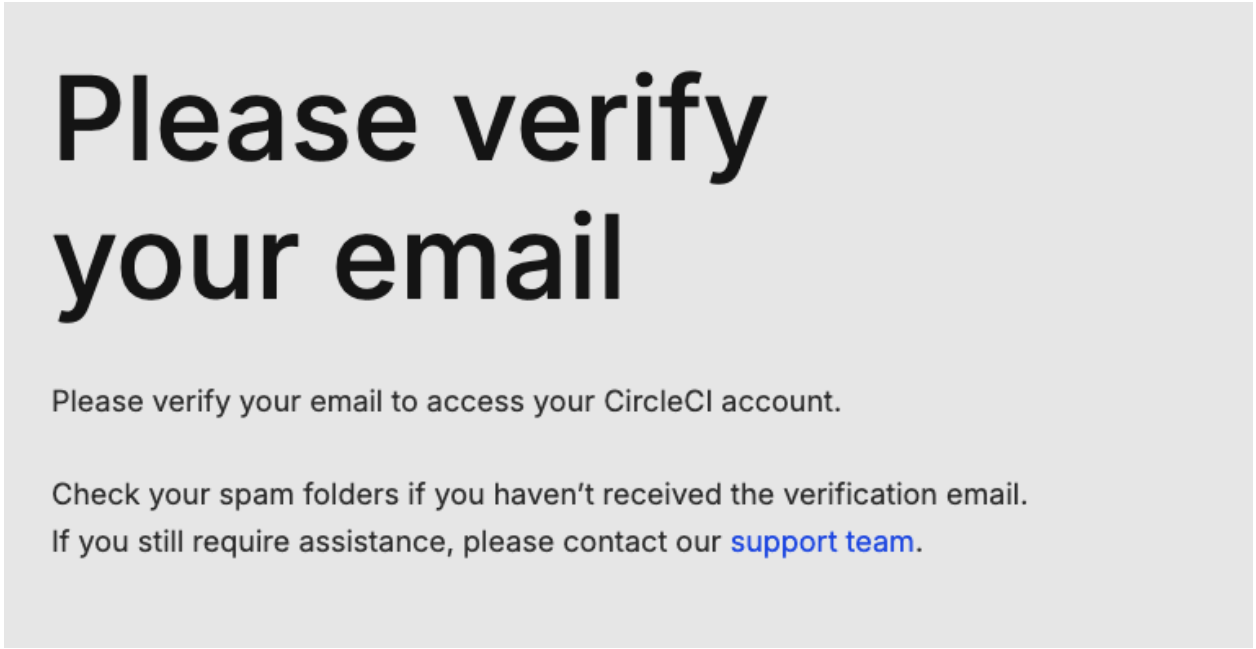


続いて2段階認証の設定を行います。Authy（https://www.authy.com/）などのアプリをインストールし、アプリからQRコードをスキャンしてください。アプリに2段階認証コードが表示されますので、入力しましょう。その後、リカバリーコードが表示されますので、パスワード管理アプリなどで保管しておき

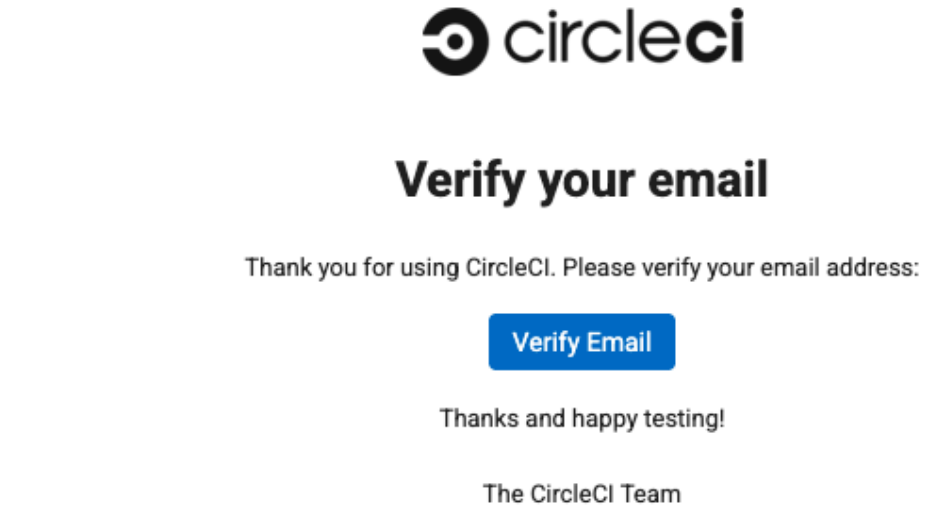
ましょう。



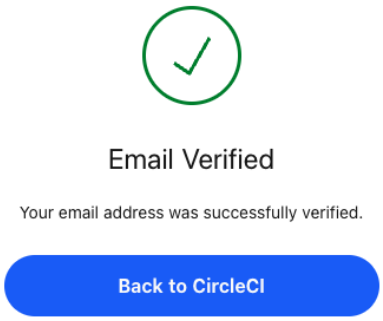
次に進むと、メールアドレスの確認ステップに移動したことが表示されます。



アカウント登録時に入力したメールアドレスに「welcome@circleci.com」からメールが届きます。「Verify Email」ボタンをクリックして、確認を完了しましょう。



認証に成功すると、認証完了画面が表示されます。これでアカウント作成が完了しました。

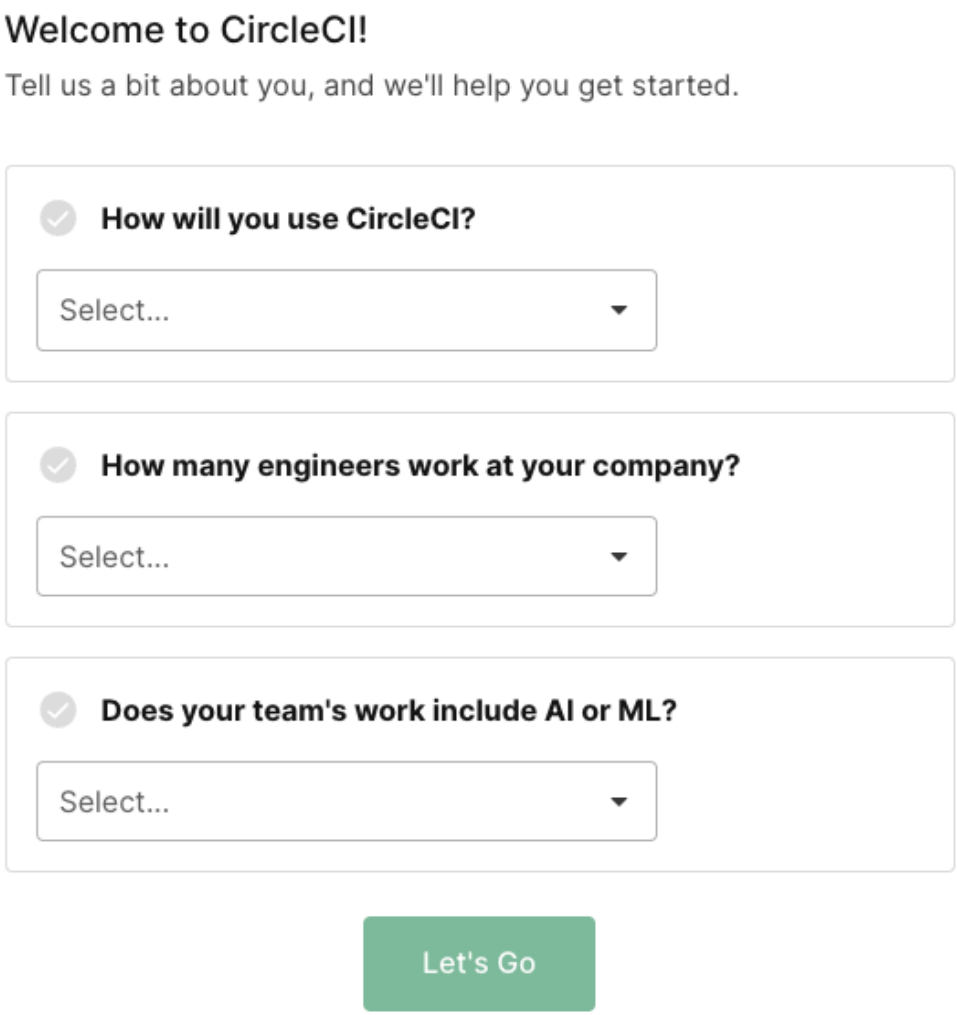


「Back to CircleCI」ボタンをクリックすると、ログイン画面に移動します。登録したメールアドレスとパスワード、そしてAuthfyなどで発行されたワンタイムパスワードを入力してログインしましょう。

ログインに成功すれば、アカウント作成は完了です。

1-4:CircleCIアカウントのセットアップ

アカウントを作成したので、セットアップを完了させましょう。まず3つのアンケートが表示されます。

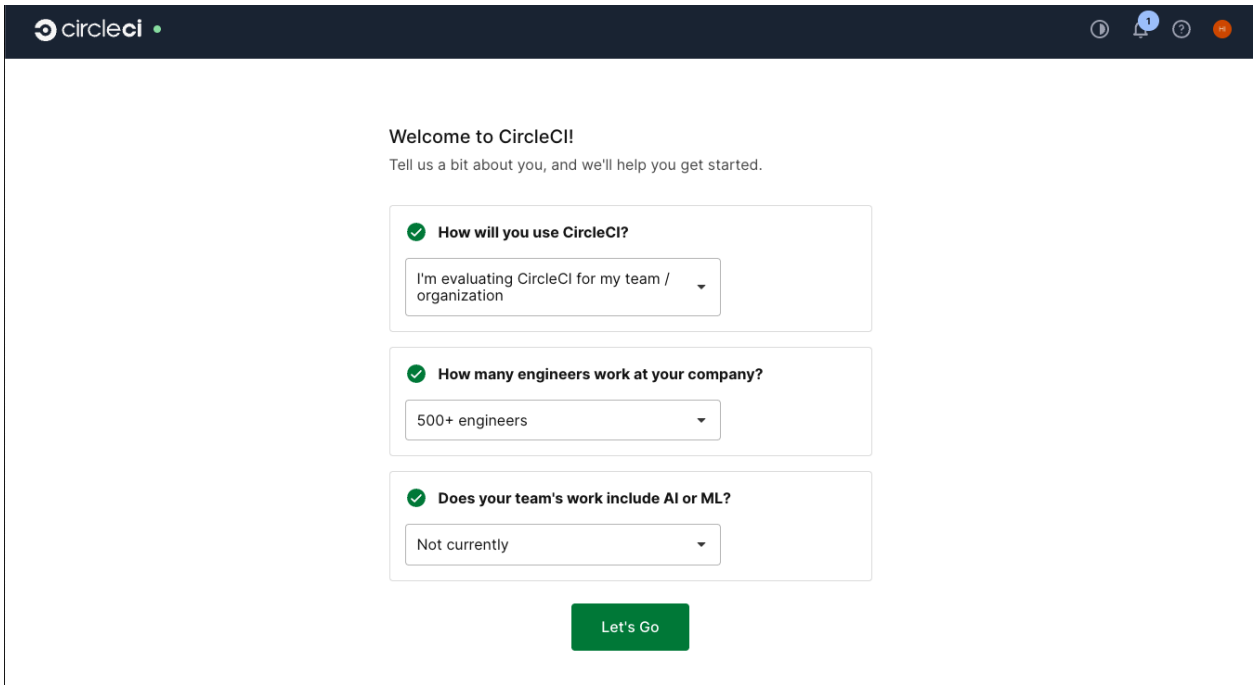


今回のチュートリアルでCircleCIをお試しの場合、以下のように入力しましょう。

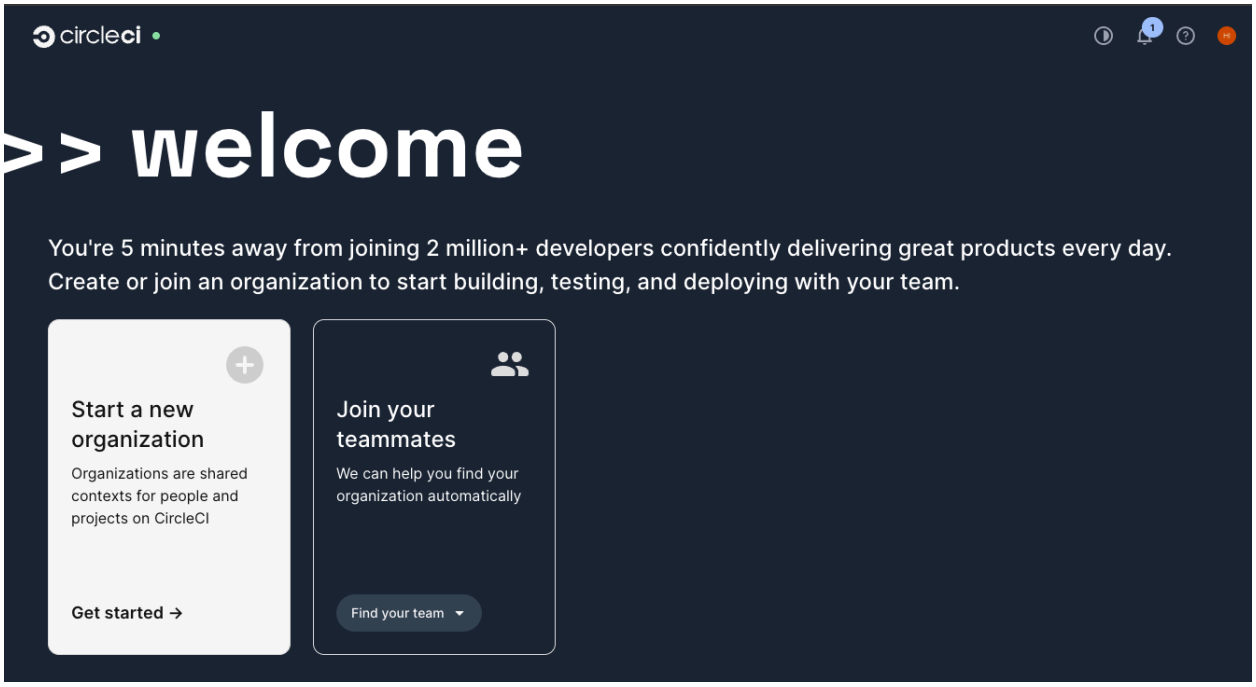
How will you use CircleCI?	I'm evaluating CircleCI for my team/ organization	利用するプロジェクトの種類について質問されています。チュートリアルでのテスト中ですので、「評価中です」と回答しましょう
How many engineers work at your company?	<あなたの会社・チームの開発者の人数>	開発チームの規模に関する質問です。個人的にお試しの場合は、「Just me」をお選びください。
Does your team's work include AI or ML?	<AIまたはMLモデルの開発に関わっていない場合は、Not	AIやMLのモデルに関する開発に関わっているかどうかの質

	currently>	問です。 AIエージェントやチャットアプリではなく、モデルに関する開発にフォーカスした質問です

入力が終わりましたら「Let's Go」をクリックしてください。

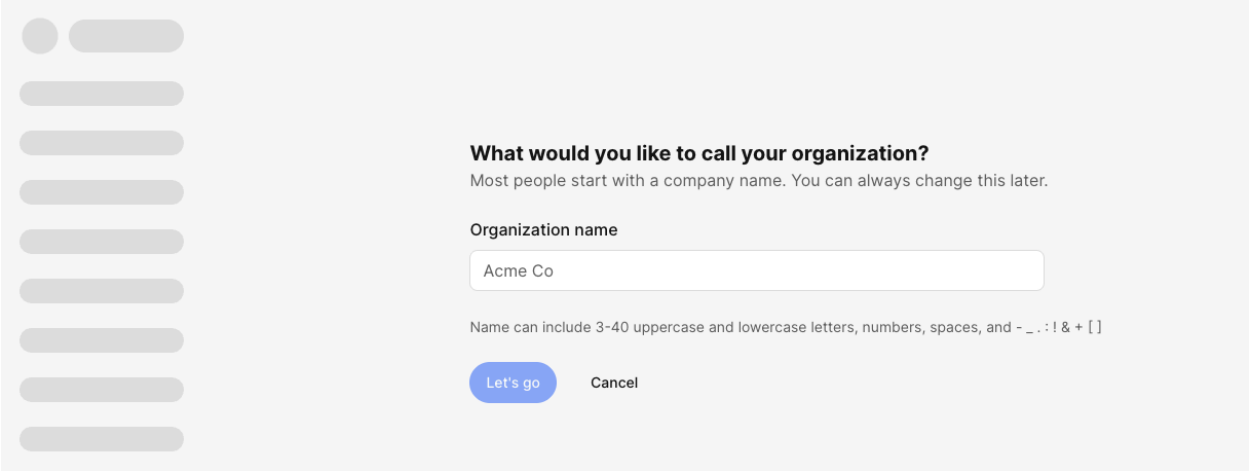


CircleCIのホーム画面が表示されます。ここではCI / CDパイプラインを管理している組織を選択します。

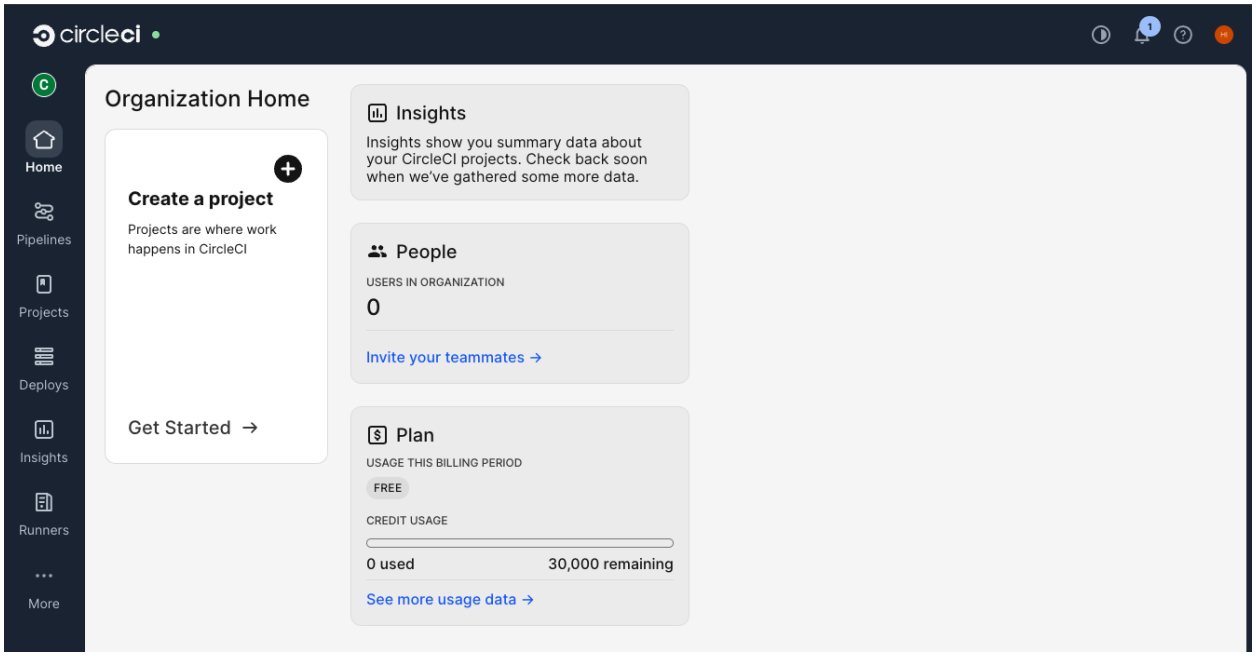


CircleCIを利用したCI / CDは、チームでの開発を想定しています。そのため、まずは新しく組織（Organization）を作成するか、既存のOrganizationに参加するかを選びます。今回は「Start a new organization」をクリックしましょう。

新しくOrganizationを作成する画面が表示されます。ここでは会社名や開発組織名を入力します。GitHubをお使いの場合、GitHub側のOrganizationに対応する名前を入力することをお勧めします。Let's goボタンをクリックすると、Organizationが作成されます。



作成されたOrganizationのHome画面に移動します。ここでは組織にて管理されているCIパイプライン（プロジェクト）や利用状況・参加しているメンバーの数などを確認できます。



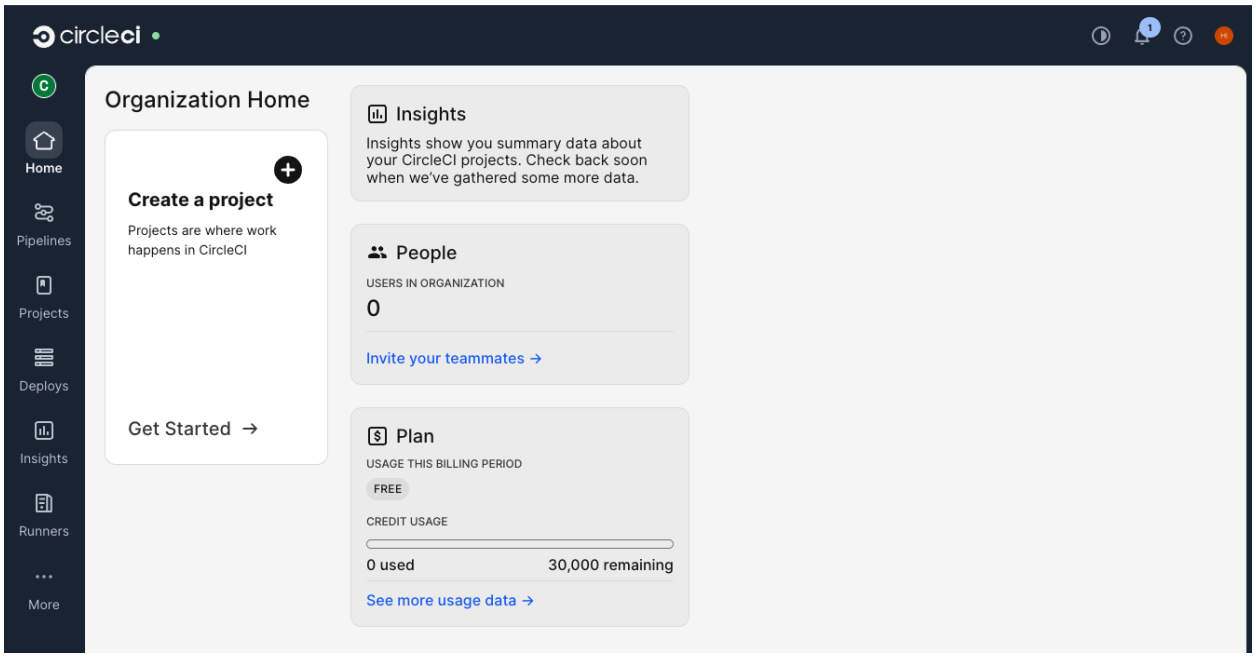
2. 最初のCircleCIプロジェクトを作成する

最初のゴールは、CircleCI上で「Hello World」のパイプラインを動かすことです。

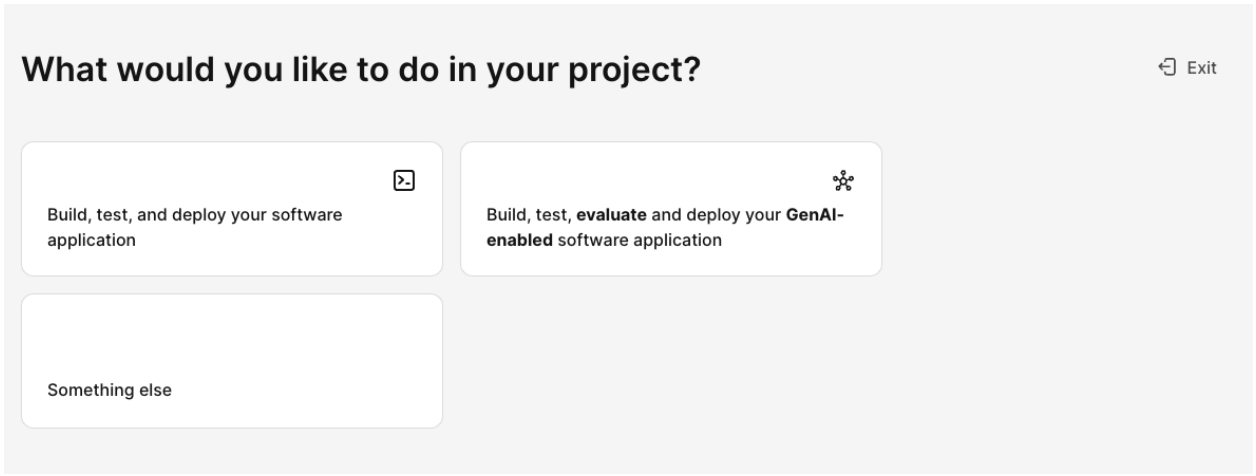
CircleCIにGitHubアカウントでログインし、ダッシュボードからプロジェクトを作成・接続します。最小限のYAMLファイルを作成し、パイプラインが無事実行されることを確認できれば、CircleCIの導入とリポジトリの接続は完了です。

2-1: プロジェクトの作成

CircleCIのホーム画面から、新しいプロジェクトを作成します。「Create a project」ボタンをクリックしてください。ボタンをクリックすると、プロジェクトのセットアップ画面に移動します。



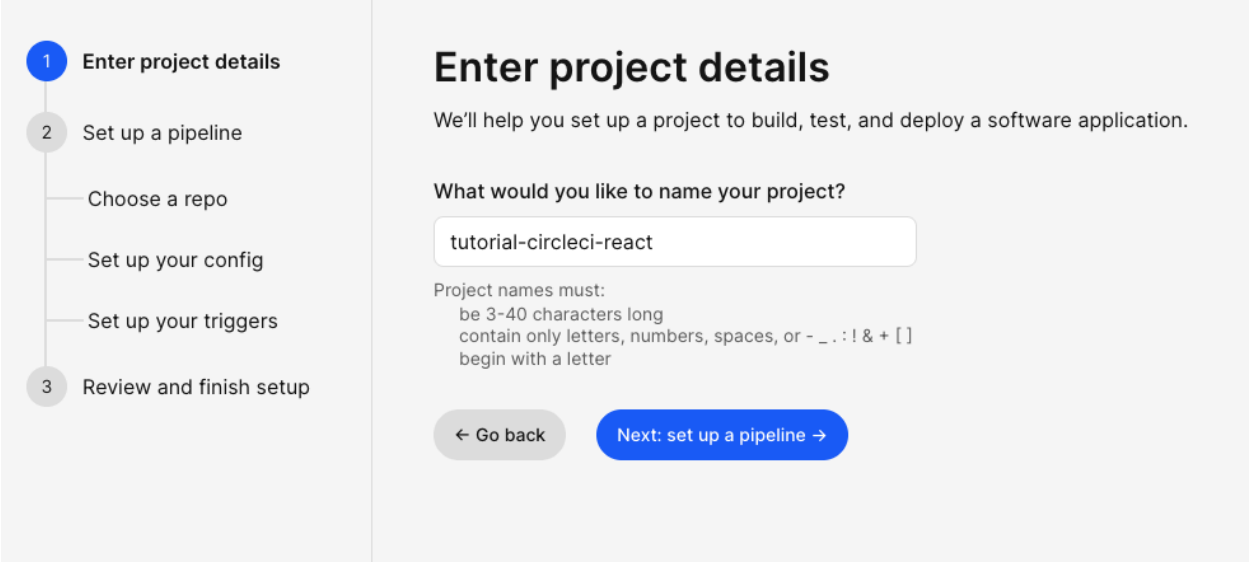
CircleCIの利用目的に関するアンケートが表示されます。今回はCI/CD構築のチュートリアルですので、「Build, test, and deploy your software application」を選択してください。



プロジェクト名の入力画面が表示されます。今回のデモアプリケーション名である「tutorial-circleci-react」を入力してください。

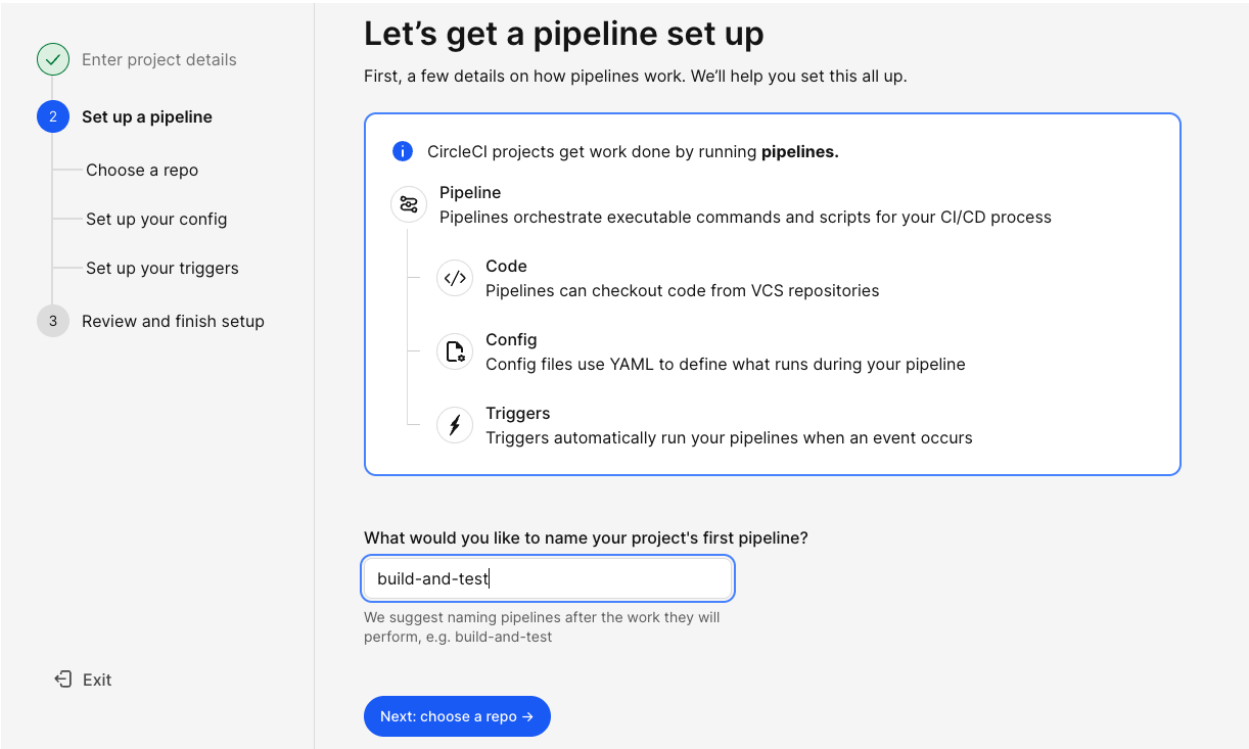
プロジェクト名は、CircleCI上でパイプラインを識別するために使用されます。GitHubリポジトリ名と一致させることで、管理が容易になります。

入力後、次へ進みます。



2-2: パイプラインの初期設定

パイプラインのセットアップ画面が表示されます。



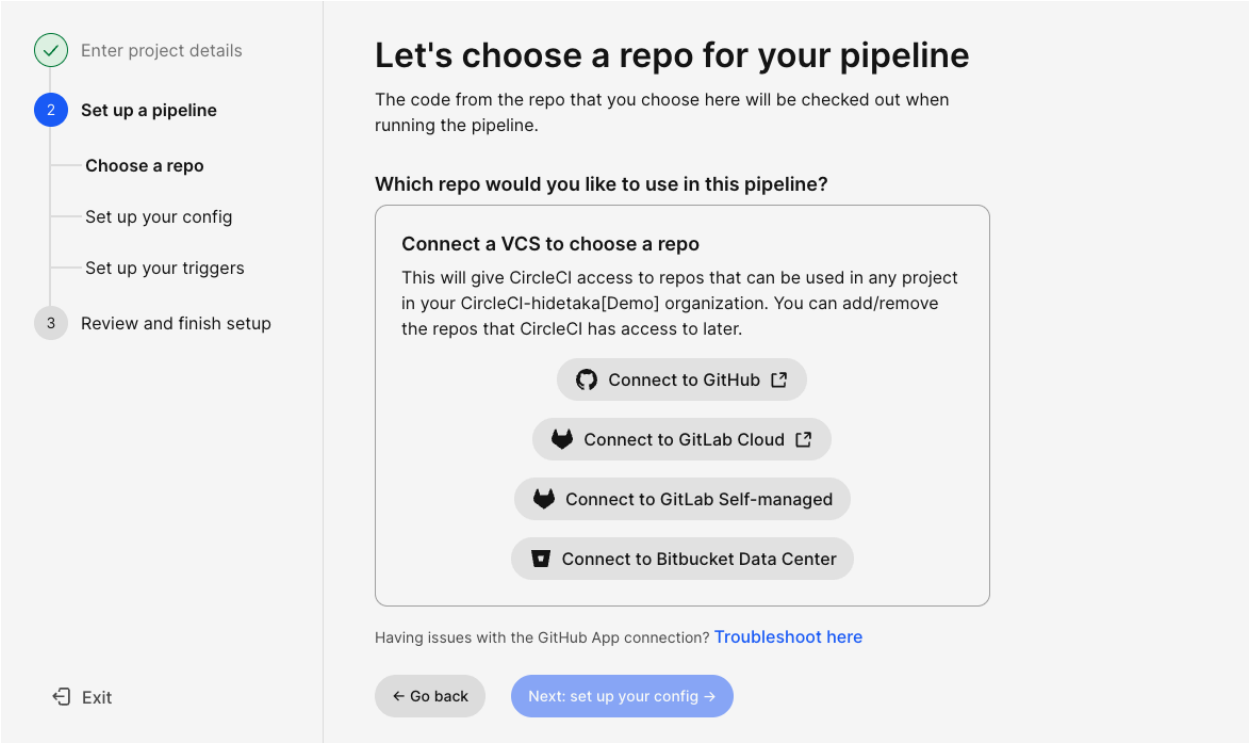
この画面では、CircleCIがどのようにパイプラインを構成するかの初期設定を行います。今回は デフォルトのまま「Next」ボタンをクリック してください。

次のステップで、実際に接続するGitリポジトリを選択します。

2-3: リポジトリの接続

CircleCIは、複数のバージョン管理システムと連携できます。今回はGitHubを利用しますが、GitLab、Bitbucket Data Centerなども選択可能です。

利用できるGitリポジトリのホスティングサービスがボタンで一覧表示されます。「GitHub」ボタンをクリックしてください。

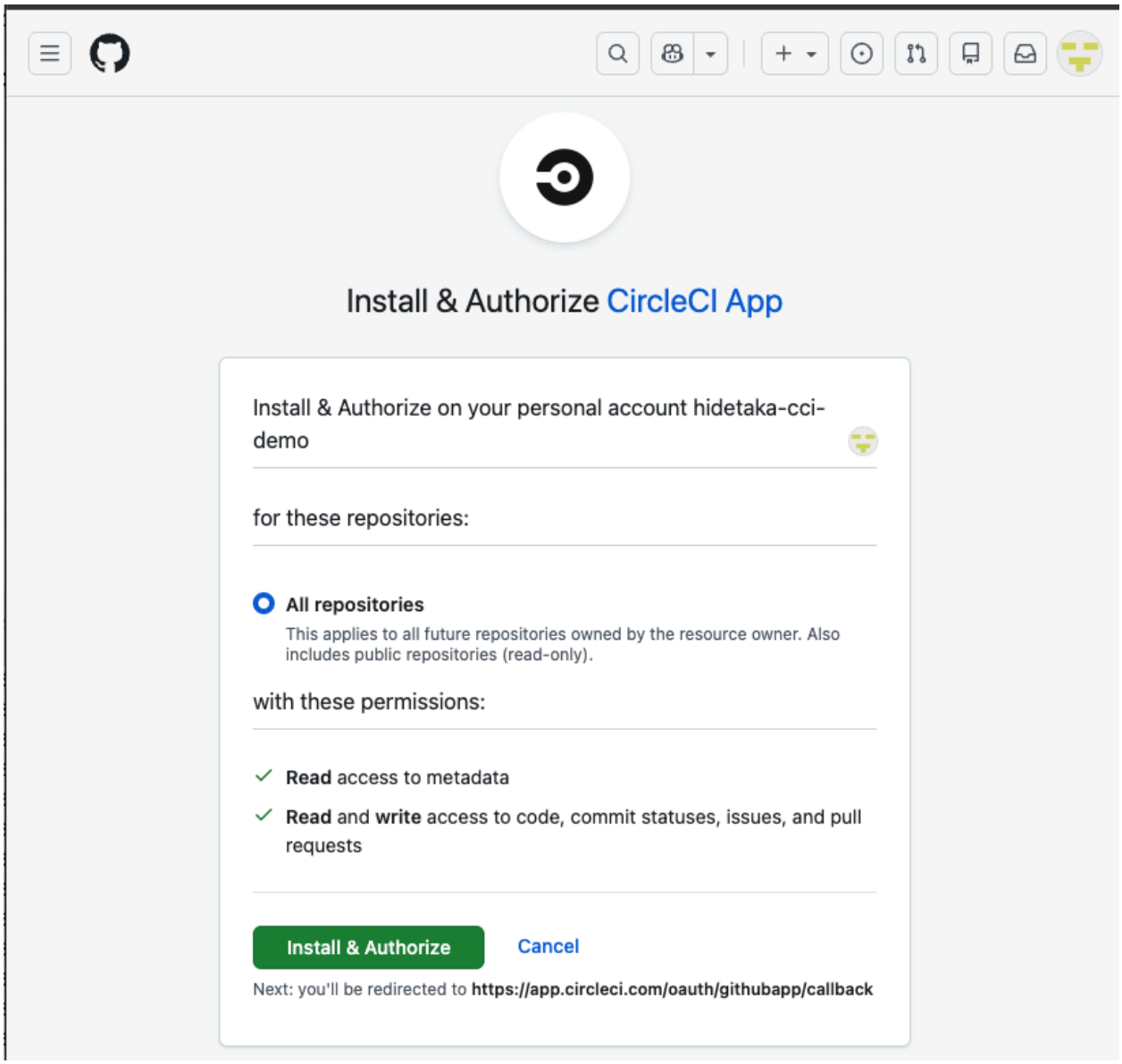


tips

CircleCIでは、複数のバージョン管理システムと連携できます。今回はGitHubを利用しますが、GitLab / Bitbucket Data Centerなどもお使いいただけます。また、SAMLを利用したシングルサインオンもサポートしていますので、組織的なユーザー・権限管理などを集約管理することも可能です。詳細は CircleCI のドキュメントをご確認ください。
https://circleci.com/docs/guides/permissions-authentication/set-up-sso/

GitHub Appのインストール

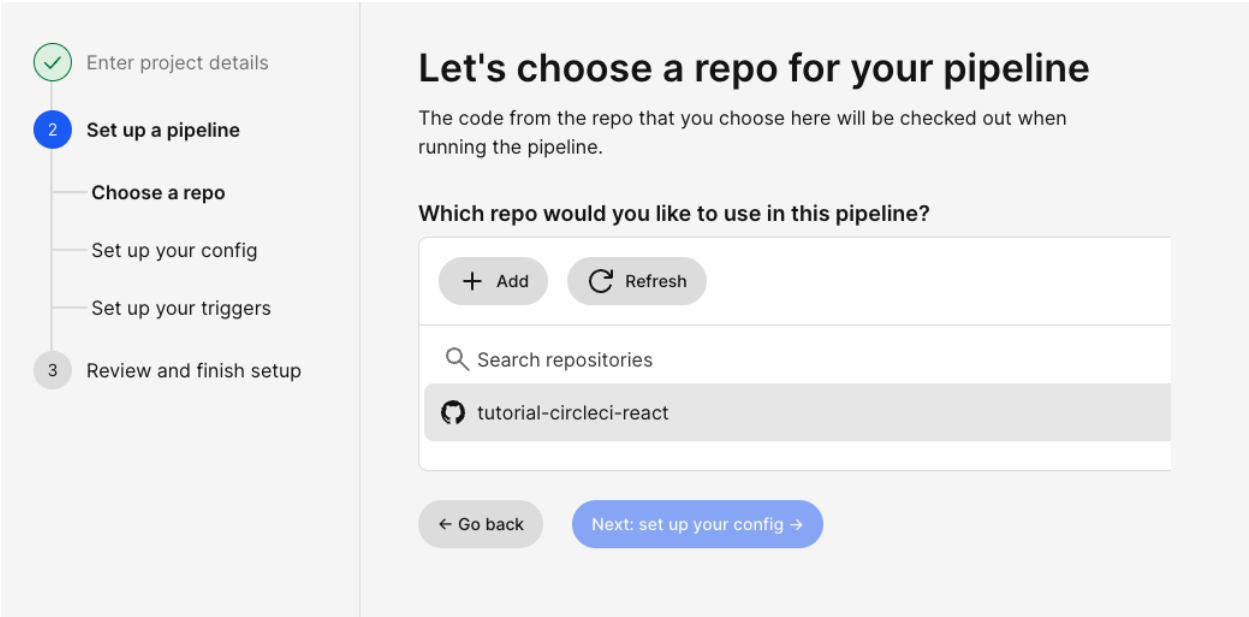
GitHubとの連携には、GitHub Appのインストールが必要です。画面の指示に従って、GitHub Appをインストールしてください。



このステップでは、CircleCIがGitHubリポジトリにアクセスするための権限を付与します。権限の内容を確認し、承認してください。

リポジトリの選択

インストールが完了すると、アクセス可能なリポジトリの一覧が表示されます。
Forkした「tutorial-circleci-react」リポジトリを選択してください。



トラブルシューティング：リポジトリが表示されない場合

リポジトリが一覧に表示されない場合は、以下を確認してください。
・「Refresh」ボタンをクリックして、リポジトリ一覧を再読み込みしてください
　GitHub Appの権限設定を確認してください
・GitHub側でCircleCIアプリに適切なリポジトリへのアクセス権が付与されているか
・リポジトリの所有権を確認してください
　自分のアカウントでForkしたリポジトリであるか

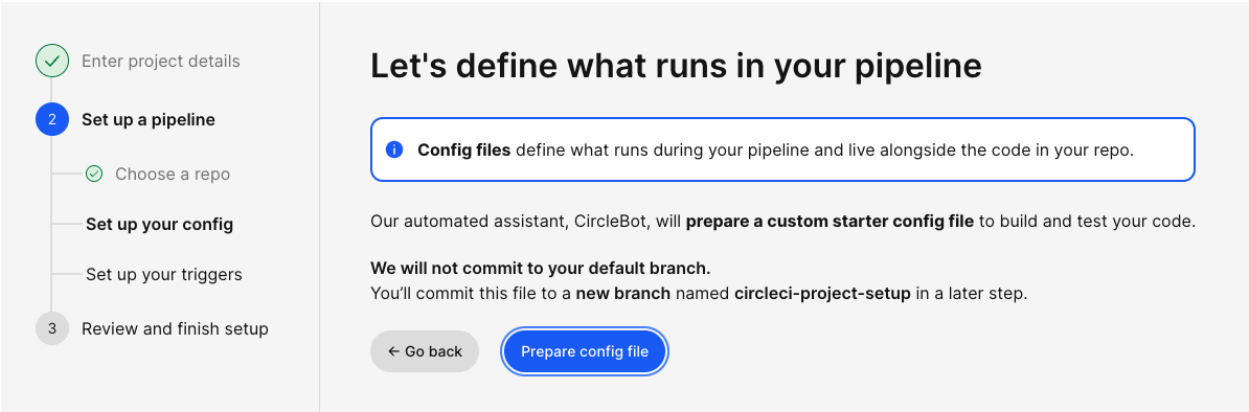
リポジトリを選択後、「next: set up your config」ボタンをクリック します。

2-4: 設定ファイルの生成

CircleCIでCI/CDパイプラインを実行するには、リポジトリのルートディレクトリに `.circleci/config.yml` という設定ファイルが必要です。このセクションでは、CircleCIのセットアップウィザードを使って、この設定ファイルを自動生成し、初回パイプラインを実行します。

設定ファイルの準備

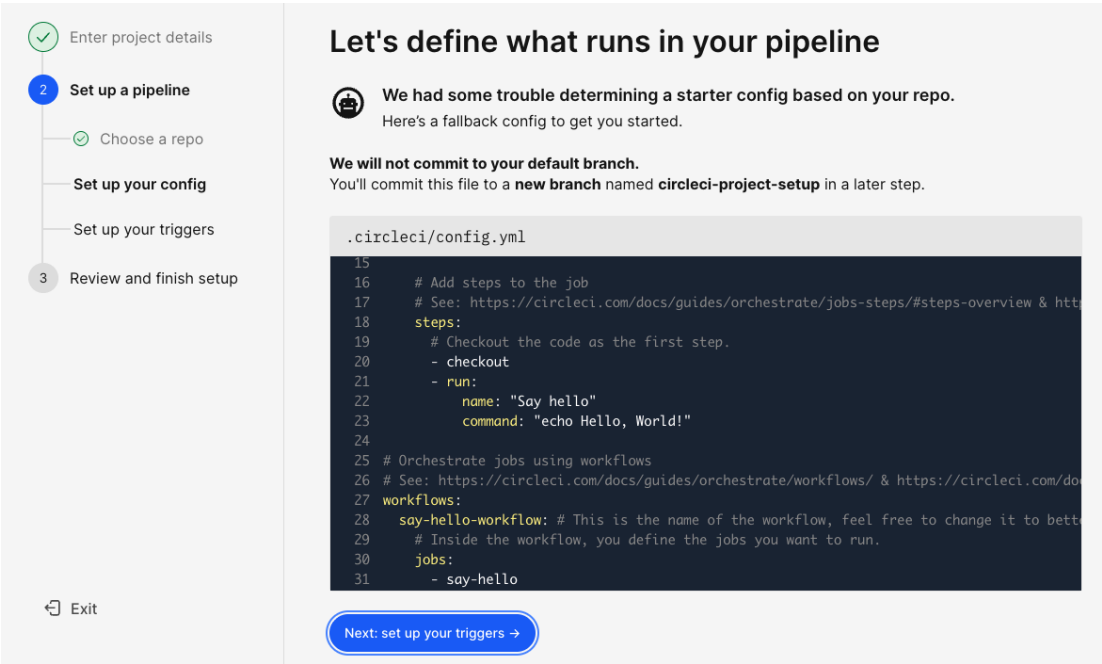
リポジトリ選択後、設定ファイルの準備画面が表示されます。ここではCircleCIの自動アシスタント「CircleBot」が、プロジェクトに適した初期設定ファイルを生成します。この設定ファイルはデフォルトブランチに直接コミットされることはありません。代わりに、circleci-project-setup という専用ブランチが作成され、そこに設定ファイルがコミットされます。



「Prepare config file」ボタンをクリックしてください。

CircleCIがリポジトリの内容を分析し、適切な設定ファイルを生成します。プロジェクトの構成によっては、プロジェクトに応じた設定か、分析に失敗した場合はHello Worldワークフローが設定されたファイルが生成されます。

画像は分析に失敗した場合に表示されるHello Worldワークフローが生成された場合の画面です。

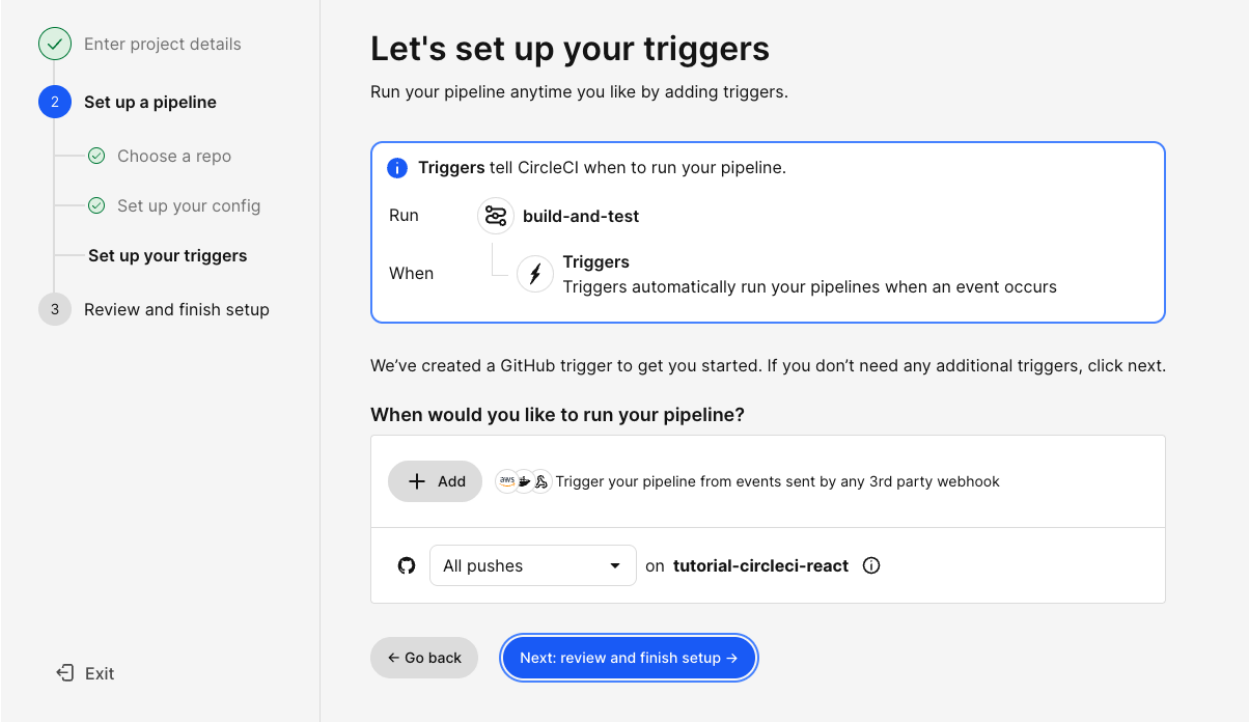


どちらのパターンが表示されても問題ありません。次のSection 3で実際のアプリケーションに合わせた設定に変更しますので、ここでは生成された設定をそのまま使用します。

「set up your triggers」ボタンをクリック して次のステップに進みましょう。

トリガーの設定

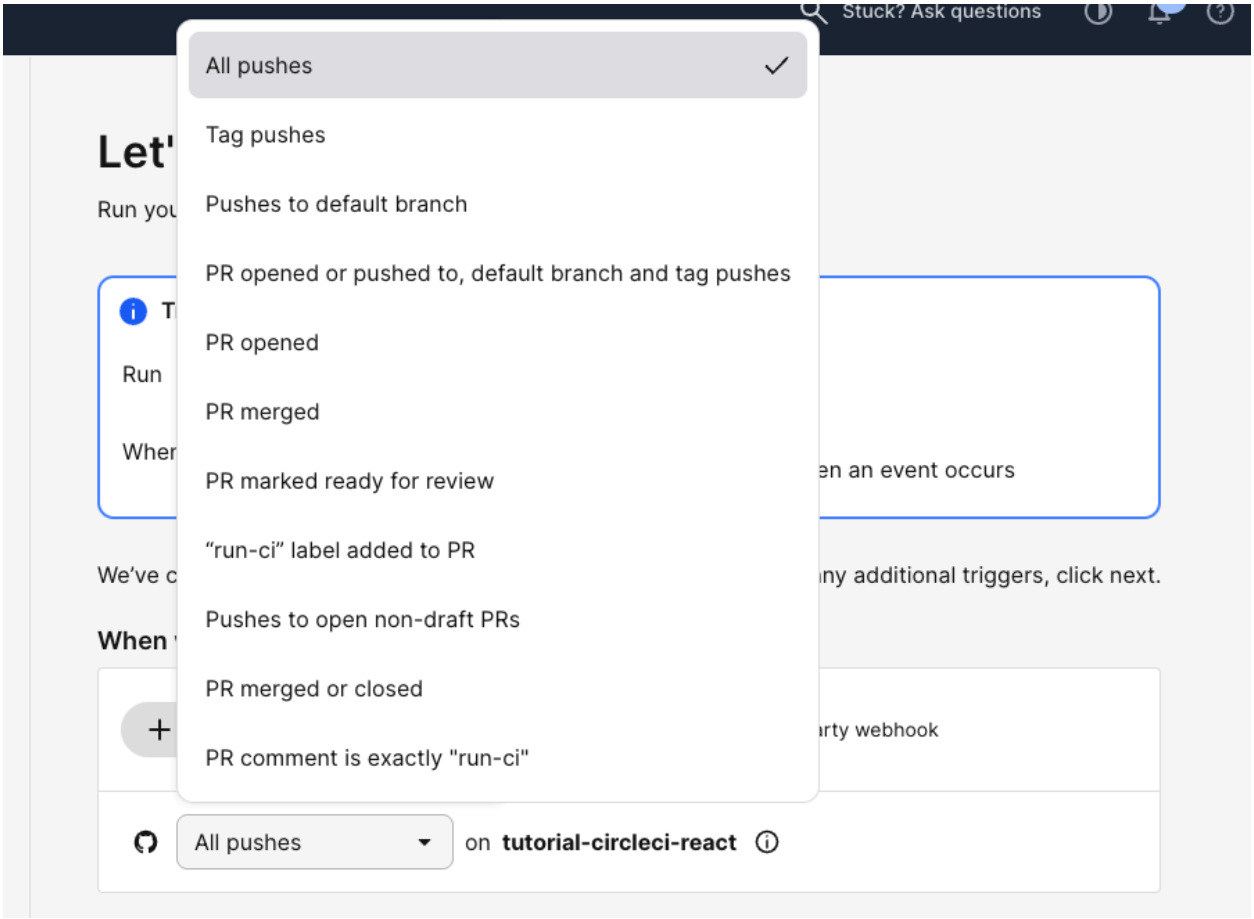
トリガー設定画面では、CI/CDパイプラインを実行する条件を指定できます。CircleCIではGitやGitHubの操作によるトリガーとWebhookベースのトリガーの2種類が指定できます。



「All pushes」のセレクトボックスをクリックすると、より詳細な条件を設定できます。例えば以下のような条件でCIやCDパイプラインをトリガーしたい場合に活用しましょう。

- タグのプッシュ時に実行
- 特定のブランチのみ実行
- Pull Request作成時に実行

セットアップ画面では1つのみ設定ができます。しかし後から設定画面で複数登録することができます。実際の運用ではトリガー設定を組み合わせて必要な時だけチェックやデプロイが行えるようにしましょう。



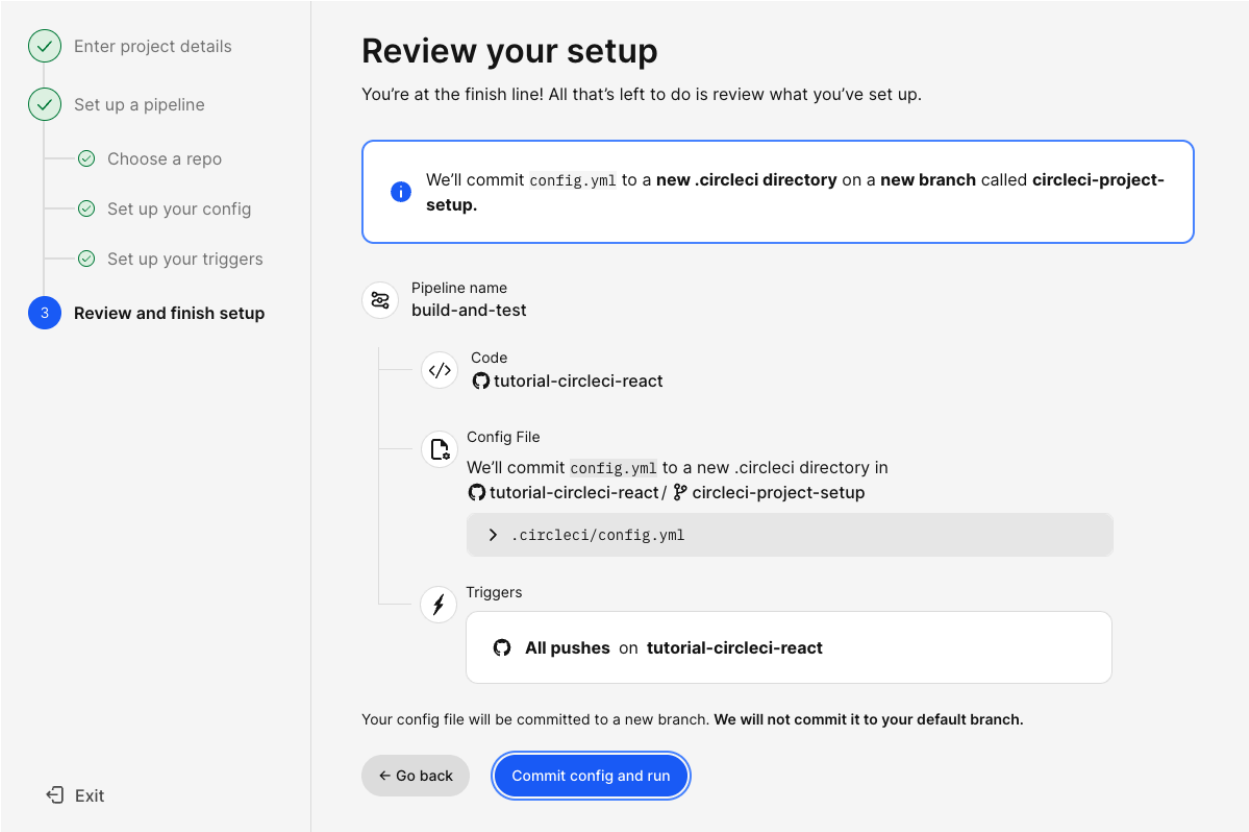
この設定は後から変更できます。今回のチュートリアルでは、デフォルトの「All pushes」のまま変更せずに進めます。

「next: review and finish setup」ボタンをクリックしてください。

最終確認とコミット

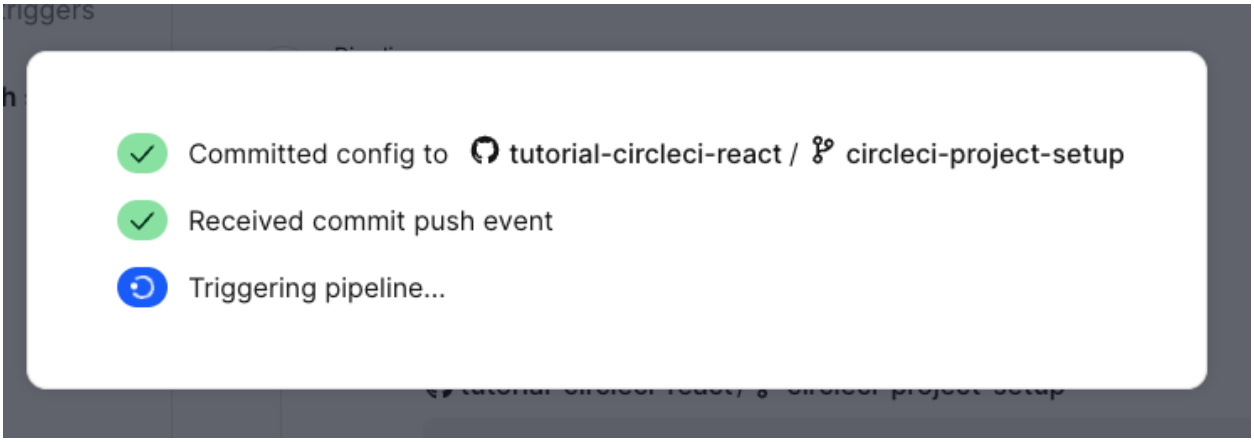
最終確認画面が表示されます。以下のような情報が表示されますので、設定に間違いがないかチェックしましょう。

- 作成される設定ファイル: `.circleci/config.yml` の内容
- コミット先: `circleci-project-setup` ブランチ
- トリガー設定: 選択したトリガー条件



内容を確認し、問題がなければ「commit config and run」ボタンをクリック します。

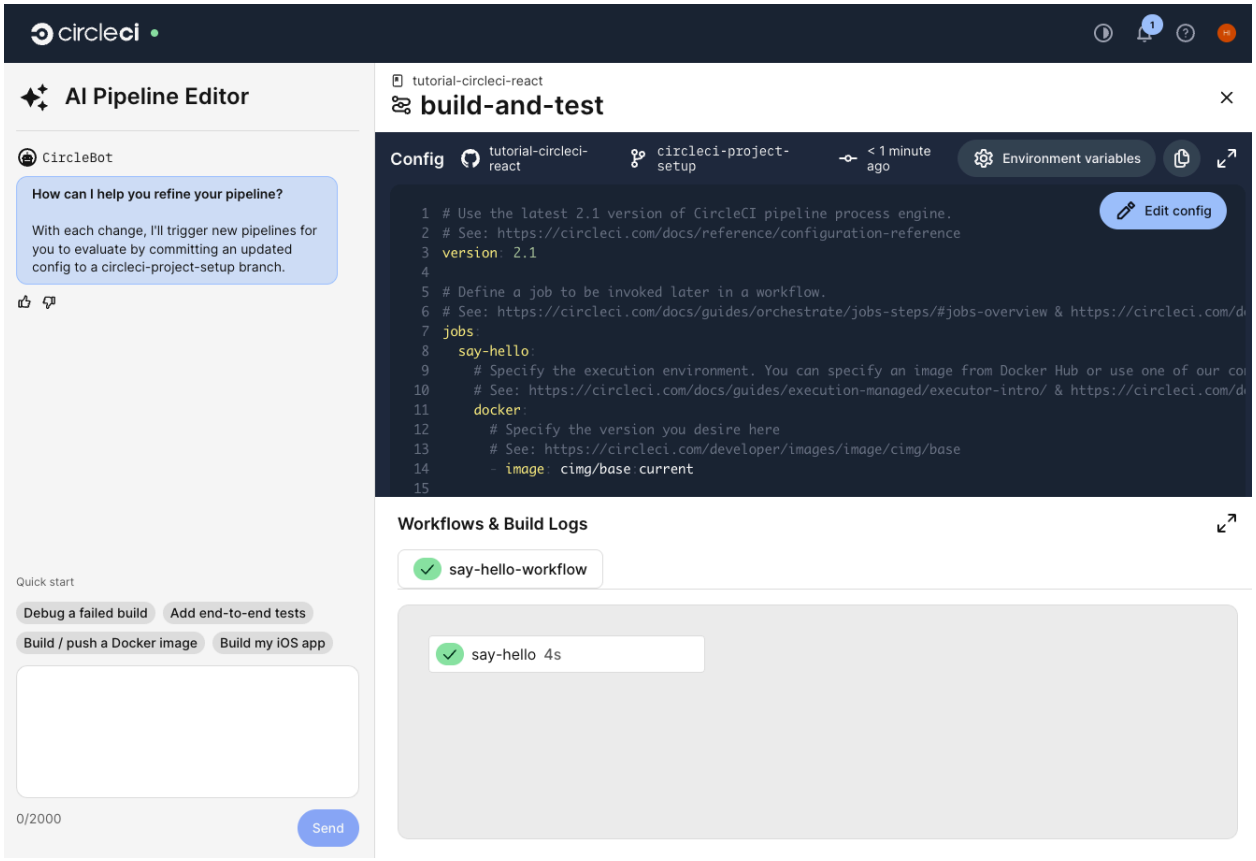
ボタンをクリックすると、新しいブランチにYAMLファイルが生成され、初回のパイプラインが実行されます。



処理が開始されると、画面にプログレスインジケーターが表示されます。数秒から数十秒で完了しますので、そのままお待ちください。

2-5: 初回パイプラインの確認

設定ファイルのコミットが完了すると、AI Pipeline Editorの画面に移動します。

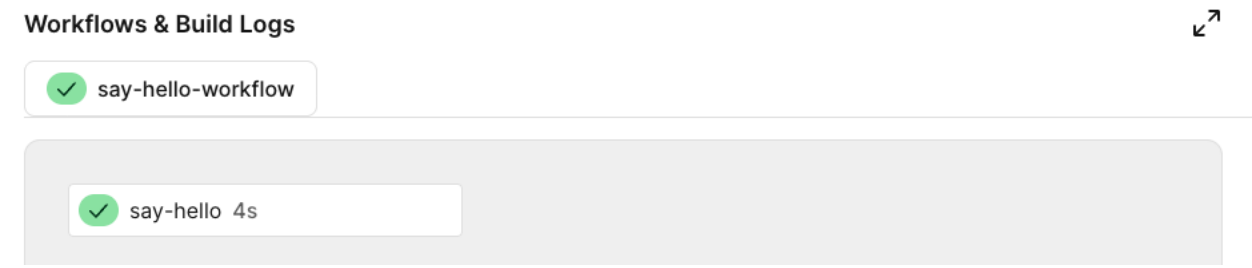


この画面では、以下の情報が表示されます。

- ワークフローの実行ログ: パイプラインの実行状況
- 現在のYAMLファイル: 作成された設定ファイルの内容
- AIチャット: AI Pipeline Editor機能（後のセクションで使用します）

ジョブの実行結果を確認する

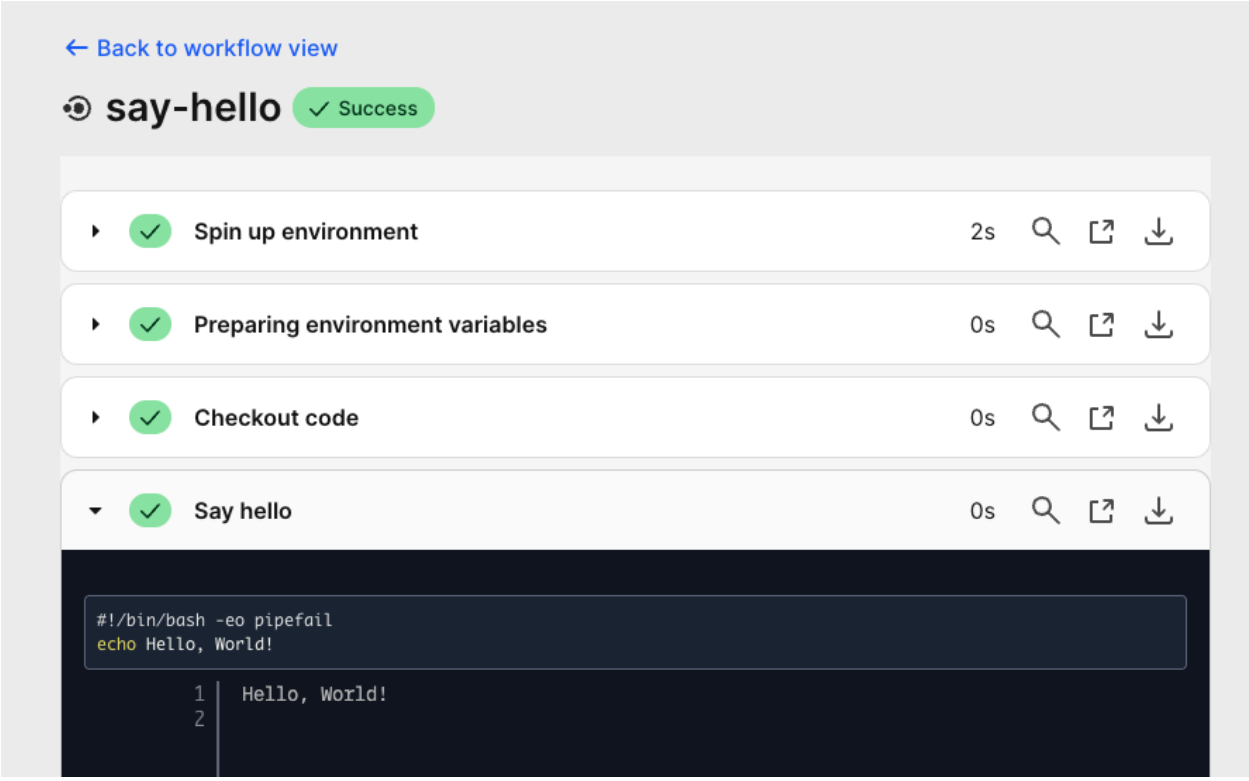
初期設定の場合、`say-hello` という名前のジョブが実行されています。ジョブ名をクリック すると、ジョブの詳細画面が表示されます。



ジョブの詳細画面では、実行されたステップごとに以下の情報が表示されます。

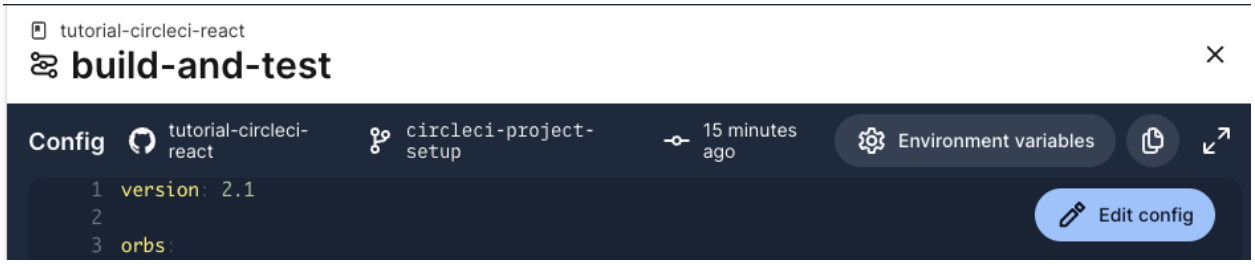
- ステップ名: 実行された処理の名前
- 実行結果: Success（成功）またはFailed（失敗）
- 出力ログ: 各ステップの実行結果

各ステップの出力は折りたたまれています。ステップ名をクリック すると、詳細なログを確認できます。



ジョブの確認が完了したら、右上の×ボタンをクリック してエディター画面を閉じます。

ページを離れることへの確認メッセージが表示される場合がありますが、「続行」を選択してください。エディターは後から再度開くことができますので、安心して閉じていただけます。

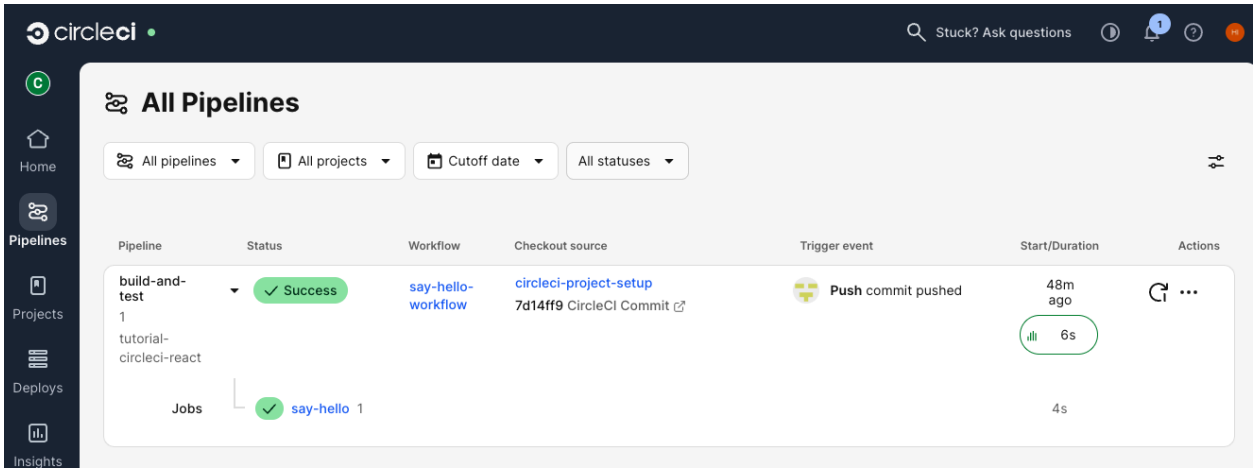


Pipelinesページの確認

エディター画面を閉じると、Pipelinesページに移動します。

このページでは、CircleCI上で実行されるすべてのCI/CDワークフロー（パイプライン）が新着順に表示されます。各パイプラインには以下の情報が表示されます。

- ステータス: Success（成功）、Failed（失敗）、Running（実行中）
- ブランチ名: パイプラインが実行されたブランチ
- コミットメッセージ: トリガーとなったコミットのメッセージ
- 実行時間: パイプラインの実行にかかった時間



「Success」ボタンをクリック すると、実行したパイプラインの詳細ページに移動できます。

このセクションで学べたこと

このセクションでは、CircleCIの導入における最初の一連の流れを体験しました。

- プロジェクト作成からパイプライン実行までの流れ: GitHubリポジトリの接続、設定ファイルの自動生成、初回パイプラインの実行
- CircleCIの基本構成要素: プロジェクト、パイプライン、ジョブ、ワークフローの関係性

- CircleCIのUI: Pipelinesページ、ジョブ詳細ページ、AI Pipeline Editorなど主要画面の使い方

次のSection 3では、この設定ファイルをカスタマイズして、実際のCI/CDパイプラインを構築します。

3. CI体験 (Goal: テストを実行し、結果を見れるようにする)

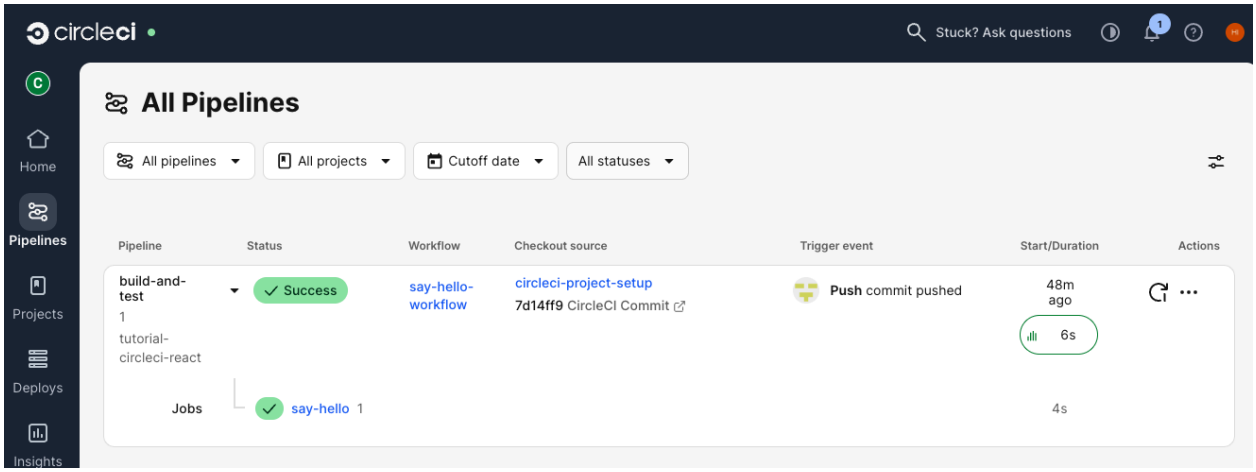
接続が完了したら、次にアプリケーションの品質を担保するCI（継続的インテグレーション）の仕組みを構築します。このステップでは、ビルド、テスト、リントの各タスクを個別のジョブとして定義します。

コードに変更があるたびにこれらのジョブが自動で実行され、すぐに開発者にフィードバックが届く開発サイクルに必須の自動化を実現します。

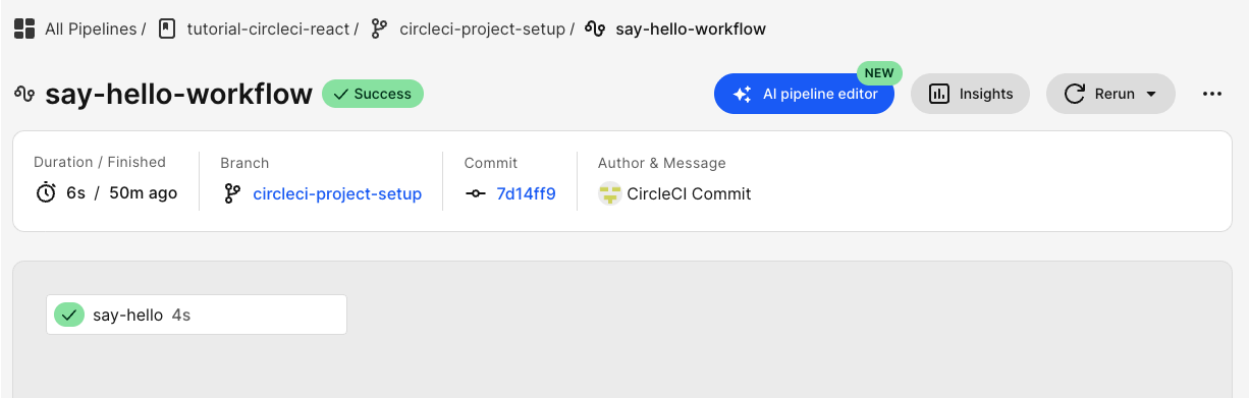
3-1: パイプライン編集ページに移動する

パイプラインの定義はYAMLファイルで行います。CircleCIのダッシュボード上から編集できますので、編集ページに移動しましょう。

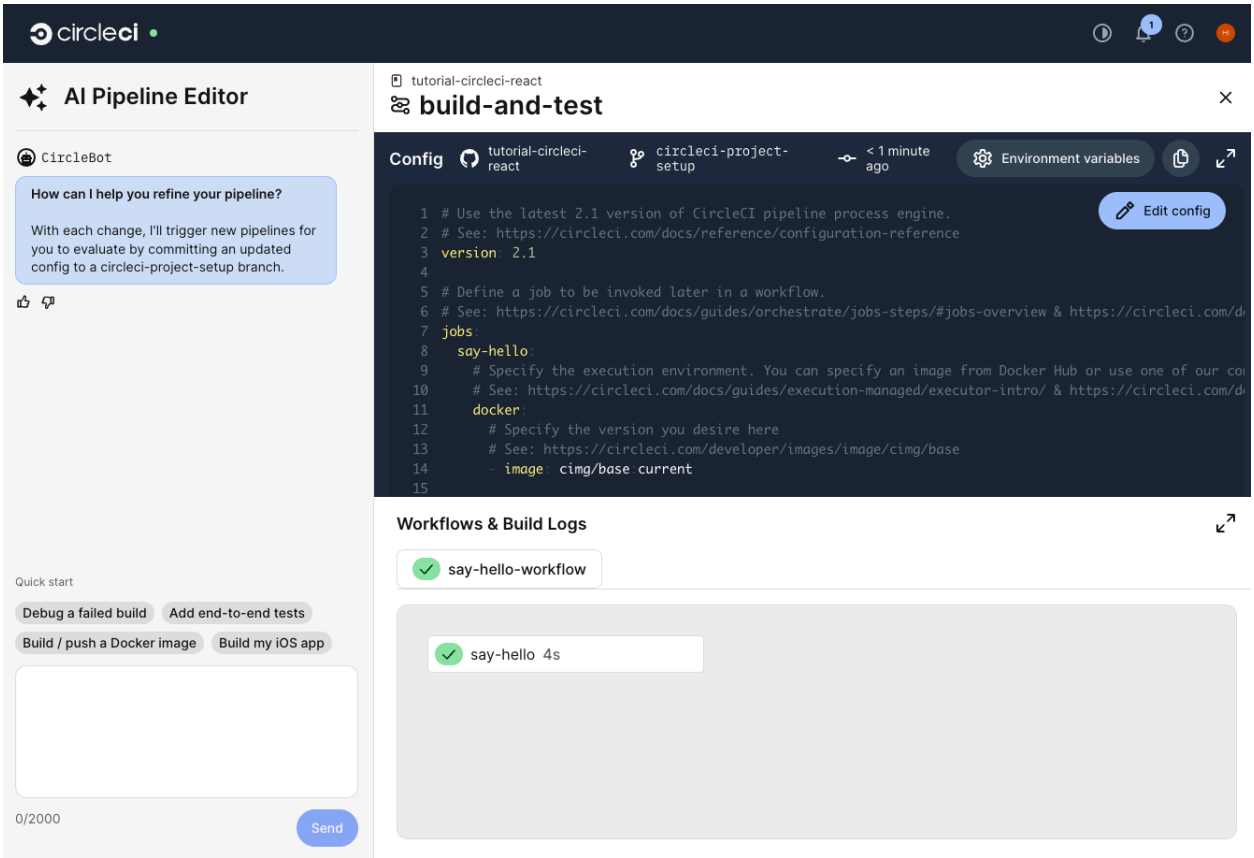
Pipelinesページから、「Success」ボタンをクリックしてください。パイプラインの詳細ページに移動します。



パイプライン詳細ページが表示されたら、「AI Pipeline Editor」ボタンをクリック してください。



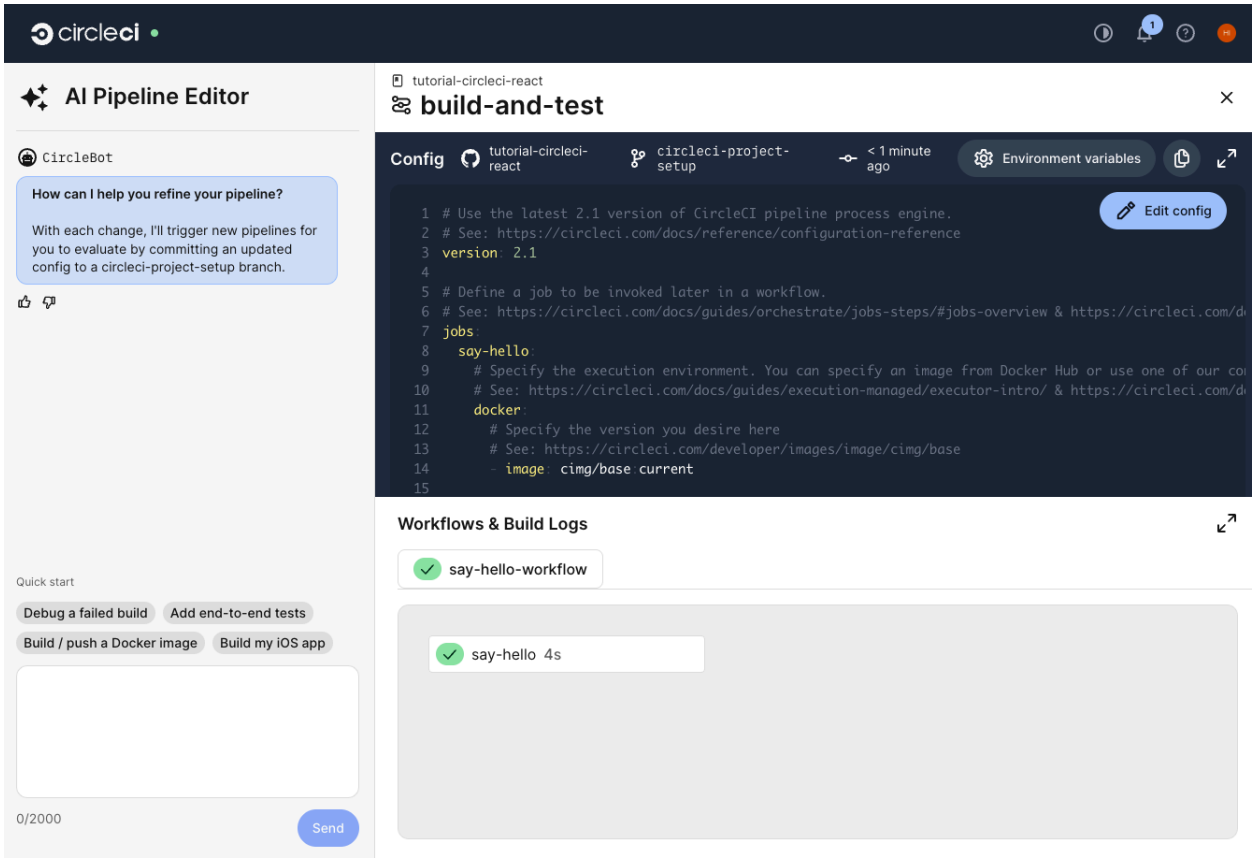
AI Pipeline Editorが表示されます。



この画面では、生成AI（AI Pipeline Editor）を利用した編集と、「Edit config」を利用した手動の編集の2パターンで設定ファイルを編集できます。

3-2: AI Pipeline EditorでCIパイプラインを手軽に構築しよう

AI Pipeline Editorを使うことで、CircleCIのパイプライン定義ファイル（YAML形式）をチャットベースで作成・編集できます。CircleCIが生成AIモデルにパイプラインを定義するために必要な情報を提供していますので、ベストプラクティスに近い設定ファイルを簡単に生成できます。

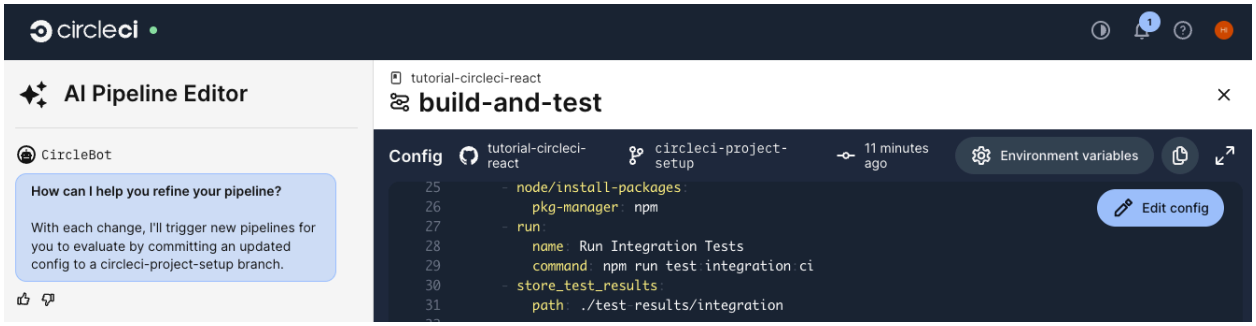


AI Pipeline Editorでの編集

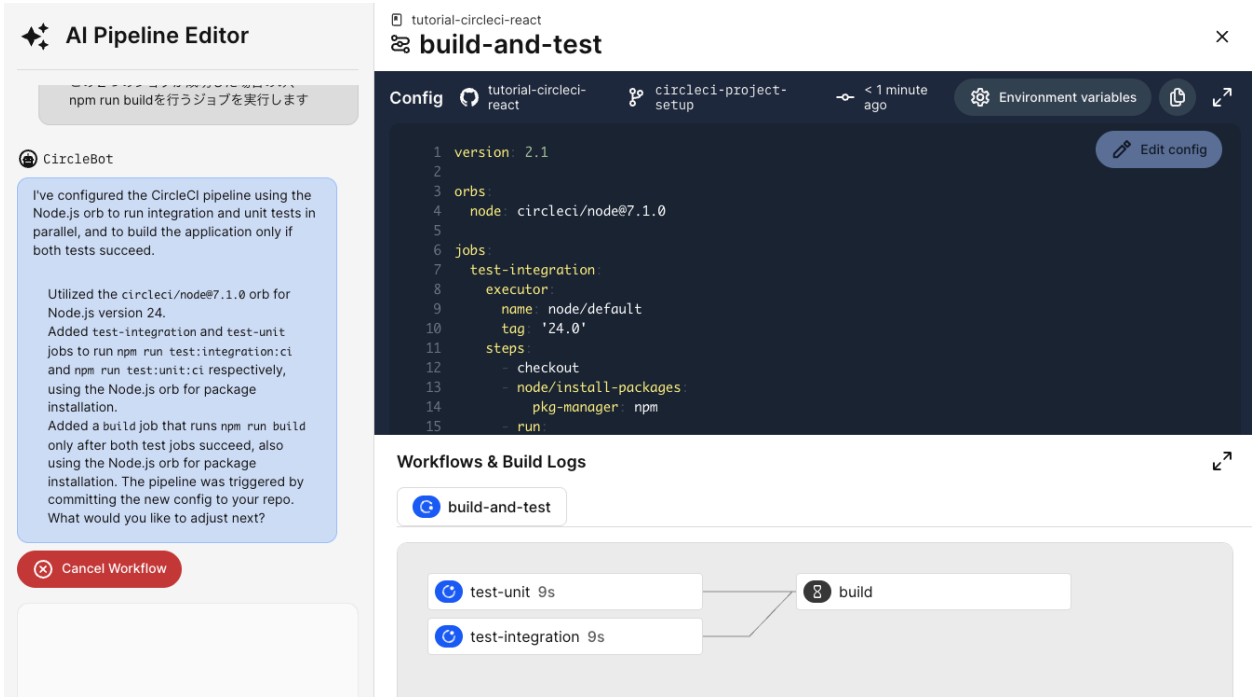
ここでこのプロジェクトに追加したいCI / CD タスクをチャットに指示してみましょう。

None

```
- これはReact / Viteのアプリです
- Node.jsはバージョン24を利用します
- CircleCIのnodejs orbを利用してください
- 全てのステップではnodejs orbを利用したインストールを実施すること
- CIテスト用のコマンド、「test:integration:ciとtest:unit:ci」がpackage.jsonのscripts
に用意されています。
- この２つをそれぞれジョブとして並列実行してください。
- この２つのジョブが成功した場合のみ、npm run buildを行うジョブを実行します
```

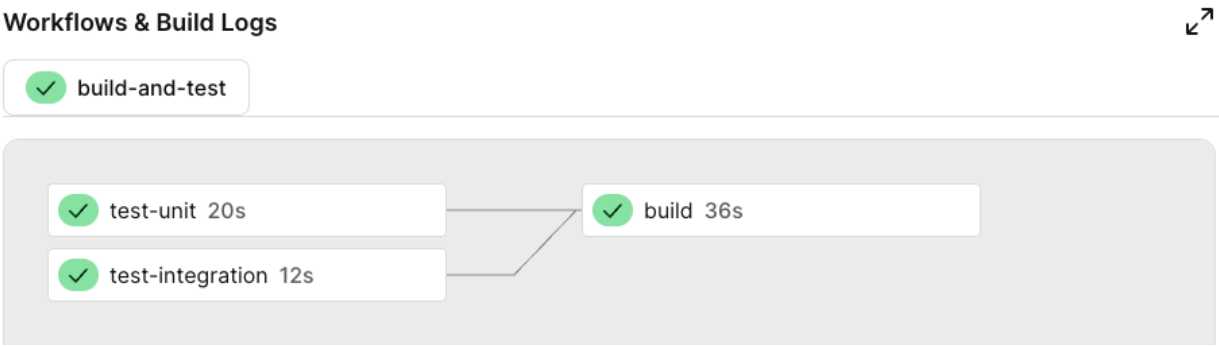


チャットを送信すると、十数秒程度でCIの設定ファイルが更新されます。YAMLファイルの更新に伴いgit pushが実行されているため、トリガー設定に従ってCIが実行されます。



パイプラインの実行を確認する

変更後のパイプラインが成功したことを確認しましょう。



このようにCircleCIの設定に特化したAIエージェントを活用することで、簡単に複数のステップかつ並列実行や順番の制御などを含めたCIパイプラインを構築できます。

AI Piepline Editorを利用する際の注意

エディターページに表示されているconfigのYAMLファイル以外は参照しません。そのため「Reactアプリのテストを追加して」や「RailsのE2Eテストを実施したい」のようなアプリケーションコードを理解する必要のある指示はうまく動作しません。「これらのコマンドを、次のような条件で以下の環境下にて実行しなさい」のように、具体的なコマンド・タスクベースで指示する必要があります。
また、この機能は今回のチュートリアルで紹介した　GitHub Appを利用した連携で動作するプロジェクトでのみご利用いただけます。

詳細はドキュメントをご確認ください。
https://support.circleci.com/hc/en-us/articles/34448217589147--Limited-Availability-Introducing-the-AI-Assisted-Pipeline-Editor

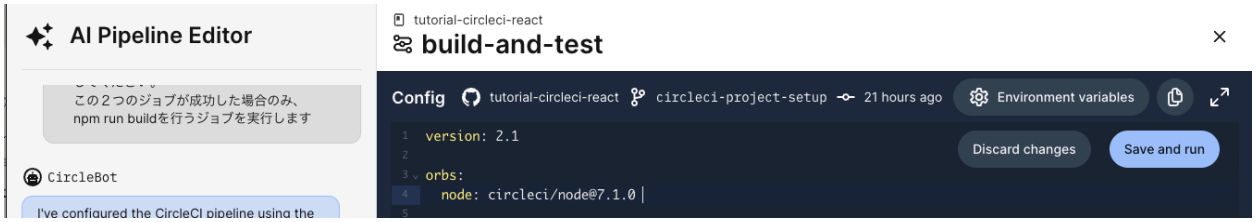
3-3: YAMLファイルを手動で編集する

次のステップでは、YAMLファイルを実際に編集しながら、CIパイプラインをよりスケーラブルにカスタマイズします。

まずは「Edit Config」ボタンをクリックしましょう。クリックすると、YAMLファイルを編集できるようになります。

編集を中止・終了する場合は、「Discard changes」ボタンをクリックしましょう。編集していた内容が

消えてしまいますので、もし何か変更をしている場合は「Save and run」を押すようにしてください。



YAMLファイルの内容を統一する

ここからはYAMLファイルを少しずつカスタマイズします。まずは全員が同じ設定ファイルから作業できるように、ファイルを完全に上書きしましょう。エディタに表示されているコードを全て削除してください。

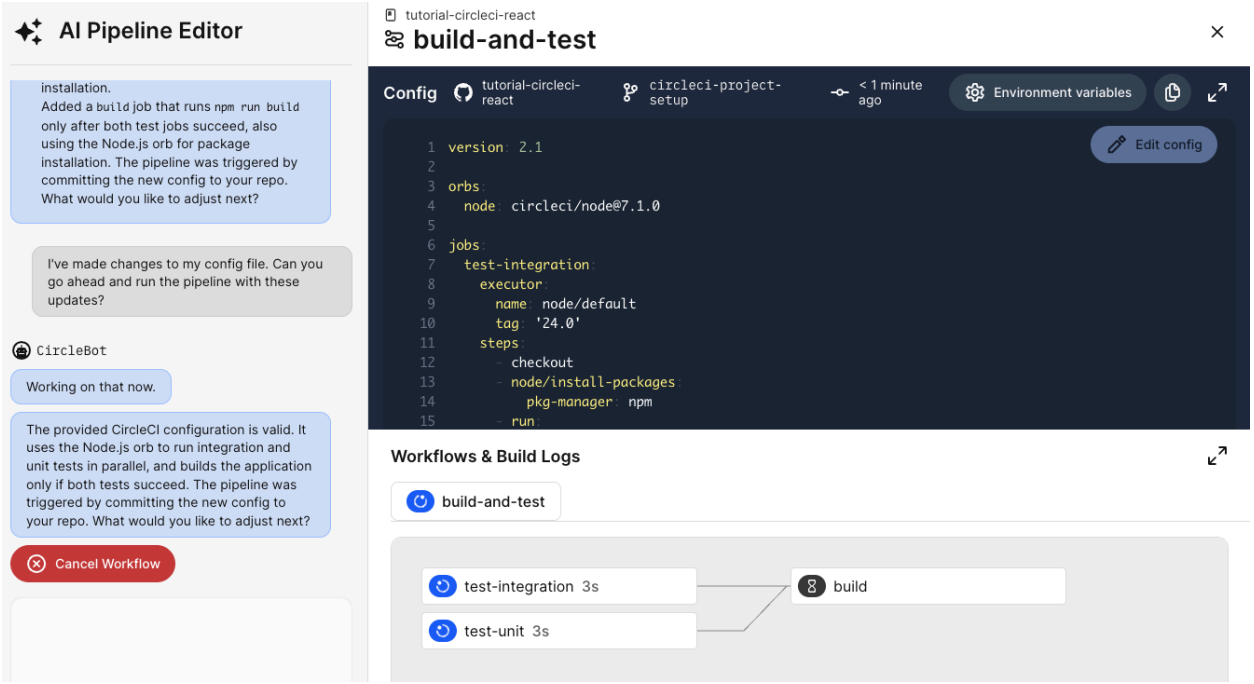
その後、以下のYAMLをコピーアンドペーストで貼り付けましょう。

```python
version: 2.1

orbs:
  node: circleci/node@7.1.0

jobs:
  test-unit:
    executor:
      name: node/default
      tag: '24.0'
    steps:
      - checkout
      - node/install-packages:
          pkg-manager: npm
      - run:
          name: Run Unit Tests
          command: npm run test:unit:ci
```

```
  test-integration:
    executor:
      name: node/default
      tag: '24.0'
    steps:
      - checkout
      - node/install-packages:
          pkg-manager: npm
      - run:
          name: Run Integration Tests
          command: npm run test:integration:ci

  build:
    executor:
      name: node/default
      tag: '24.0'
    steps:
      - checkout
      - node/install-packages:
          pkg-manager: npm
      - run:
          name: Build Application
          command: npm run build

  lint:
    executor: node/default
    steps:
      - checkout
      - node/install-packages:
          pkg-manager: npm
      - run:
          name: Lint
          command: npm run lint

workflows:
  version: 2
  build-and-test:
    jobs:
      - test-integration
      - test-unit
      - build:
          requires:
            - test-integration
            - test-unit
```

最後に「Save and run」ボタンをクリックします。するとCircleBotがCIファイルの構文をチェックし、そ

の後新しくパイプラインを実行します。



YAMLファイルを読み解く

この設定ファイルでは、以下の構成を定義しています。

Orbs（オーブ）の宣言

yaml

```
None
orbs:
  node: circleci/node@7.1.0
```

- Node.jsの環境設定やパッケージインストールを簡単に行うための共通パッケージ

ジョブの定義

- `test-integration`: 統合テストを実行するジョブ
- `test-unit`: ユニットテストを実行するジョブ
- `build`: アプリケーションをビルドするジョブ

ワークフローの定義

yaml

```
workflows:
  version: 2
  build-and-test:
    jobs:
      - test-integration
      - test-unit
      - build:
          requires:
            - test-integration
            - test-unit
```

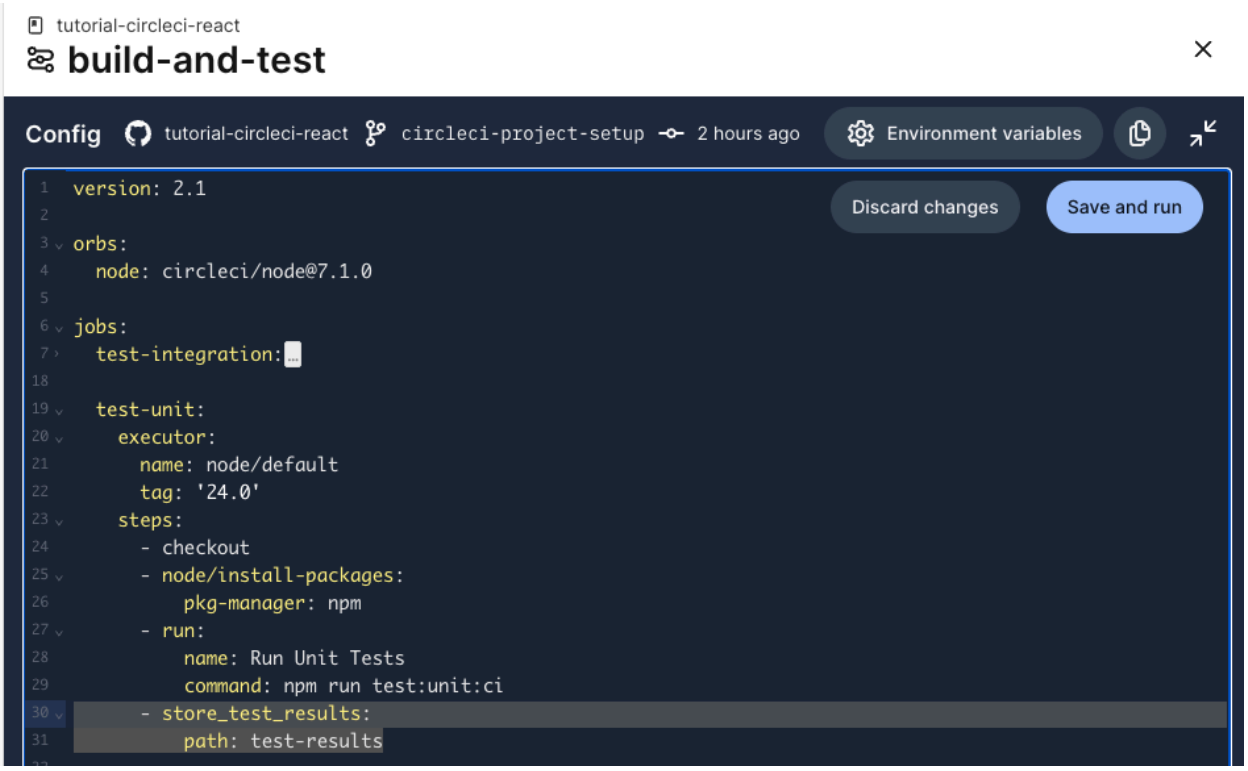
- `test-integration`と`test-unit`は並列実行されます
- `build`ジョブは、両方のテストが成功した場合のみ実行されます

テスト実行結果を可視化する

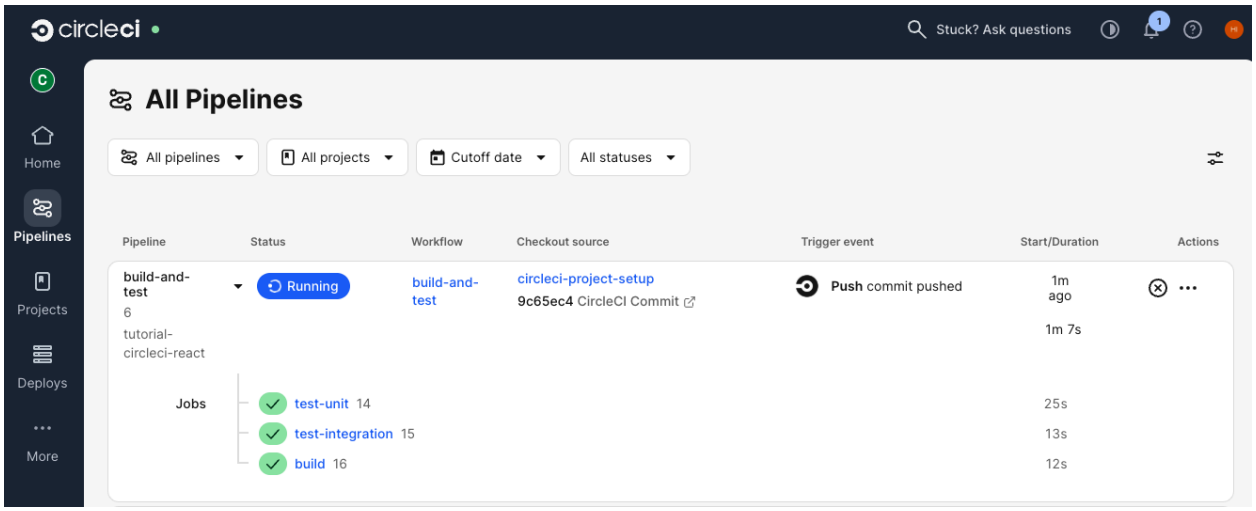
CircleCIでは、実行したテストのサマリーやエラーレポートをダッシュボードから見ることができます。テスト結果を保存するには、`store_test_results`ステップを追加しましょう。設定ファイルの`test-unit`ジョブに、以下のステップを追加してください。

```
  test-unit:
    executor:
      name: node/default
      tag: '24.0'
    steps:
      - checkout
      - node/install-packages:
          pkg-manager: npm
      - run:
          name: Run Unit Tests
          command: npm run test:unit:ci
      - store_test_results:
          path: test-results
```

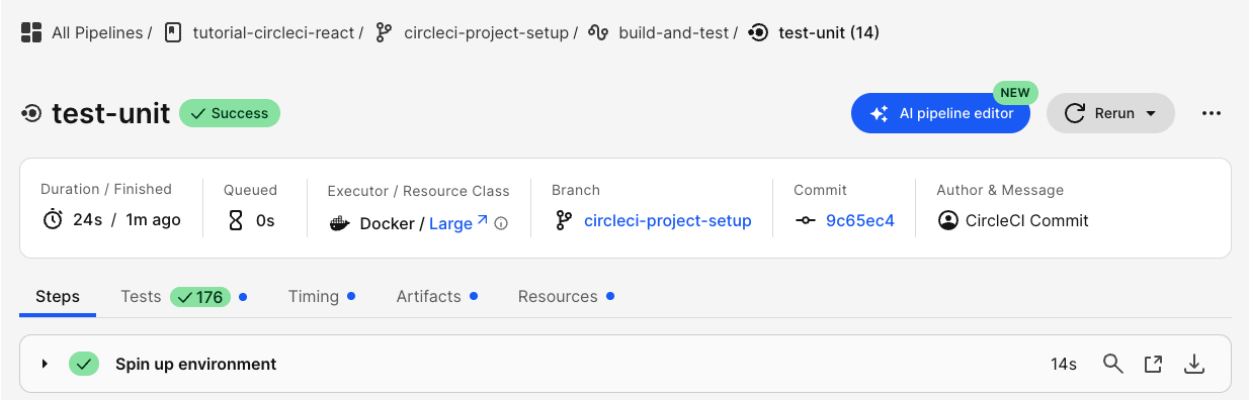
追加したのちに、「Save and run」ボタンをクリックしましょう。



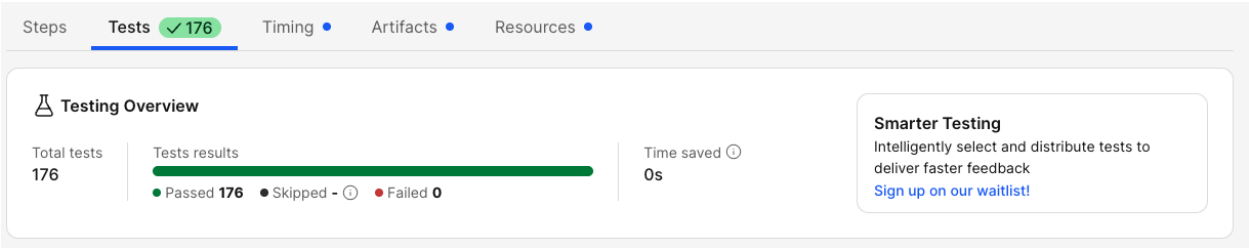
「x」ボタンをクリックして編集画面をクローズします。その後表示されるAll Pipelinesページにて、一番上に表示されているパイプラインの「test-unit」テキストをクリックしてください。



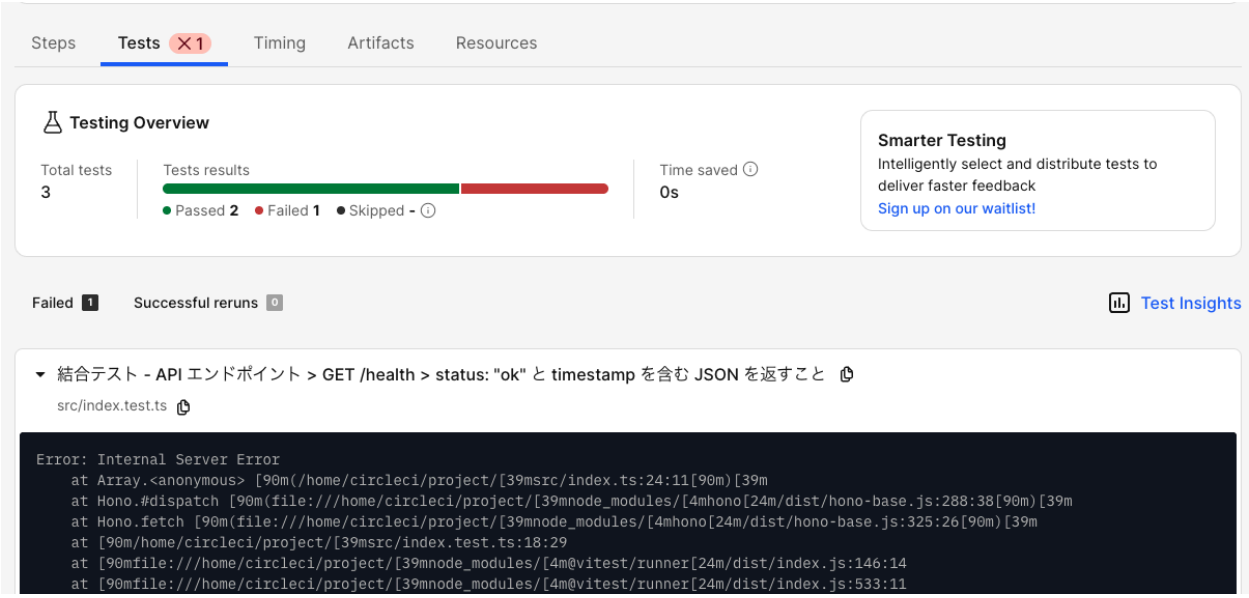
ユニットテストを実行するジョブの詳細や結果を見ることができます。



`store_test_results`ステップを追加したことで、「Tests」タブに「176」という数字が表示されるようになりました。このタブをクリックすると、成功したテスト・失敗したテストそしてスキップされたテストの数が表示されます。



もしテストが失敗した場合は、このタブに失敗したテストの詳細とエラーログが表示されます。

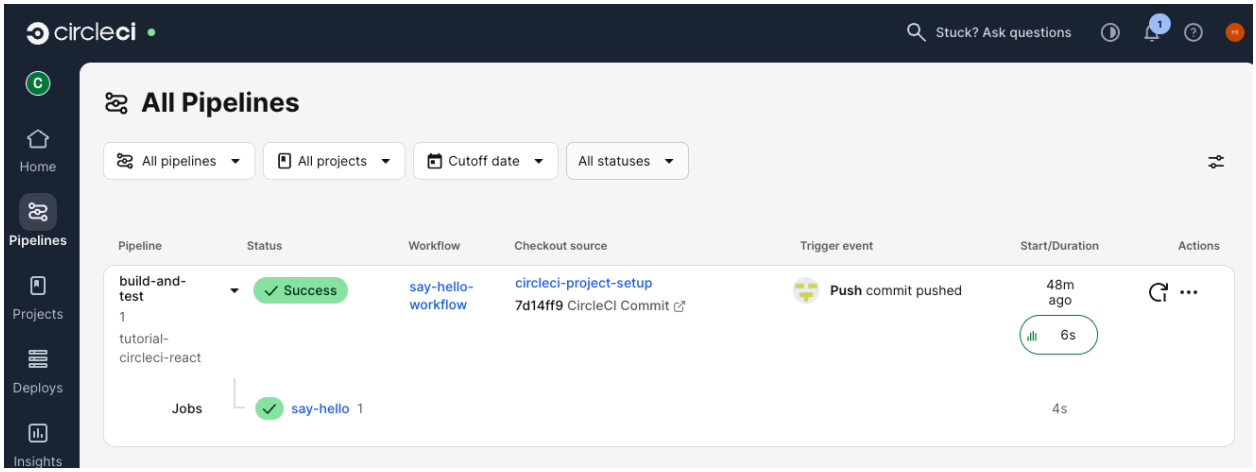


テスト結果をJUnit形式でCircleCIに保存することで、このようにテストの失敗分析や実施状況のレポートなどを複雑な設定や追加開発することなく実現できます。

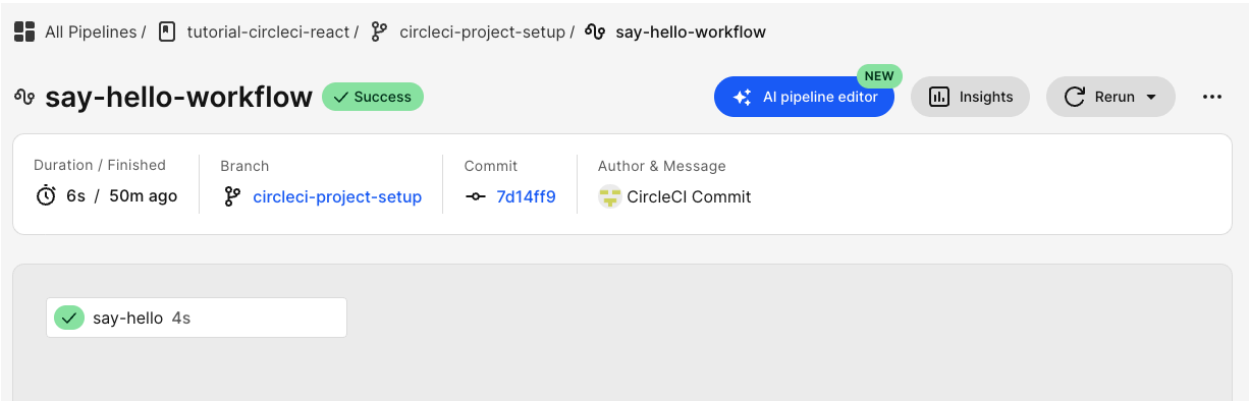
並列実行でテストを高速化する

テスト件数に比例してテストの実行時間は増加します。しかし実行時間を短縮するためにテストの数を減らすことは、品質管理の観点からも推奨されません。CircleCIでは、1行の設定を追加するだけで、テストジョブを並列実行させることができます。

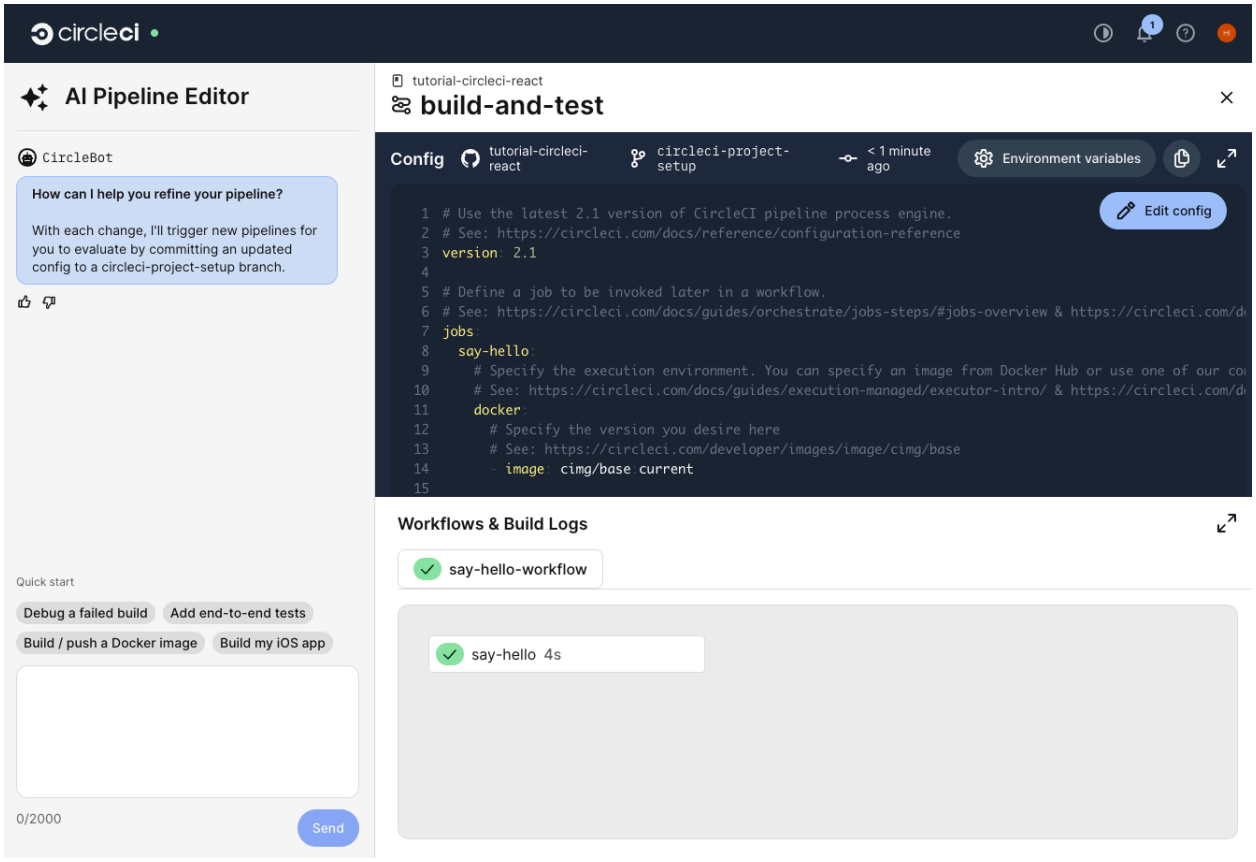
Pipelinesページから、「Success」ボタンをクリックしてください。パイプラインの詳細ページに移動します。



パイプライン詳細ページが表示されたら、「AI Pipeline Editor」ボタンをクリック してください。



AI Pipeline Editorが表示されます。



「Edit Config」ボタンをクリックしましょう。クリックすると、YAMLファイルを編集できるようになります。

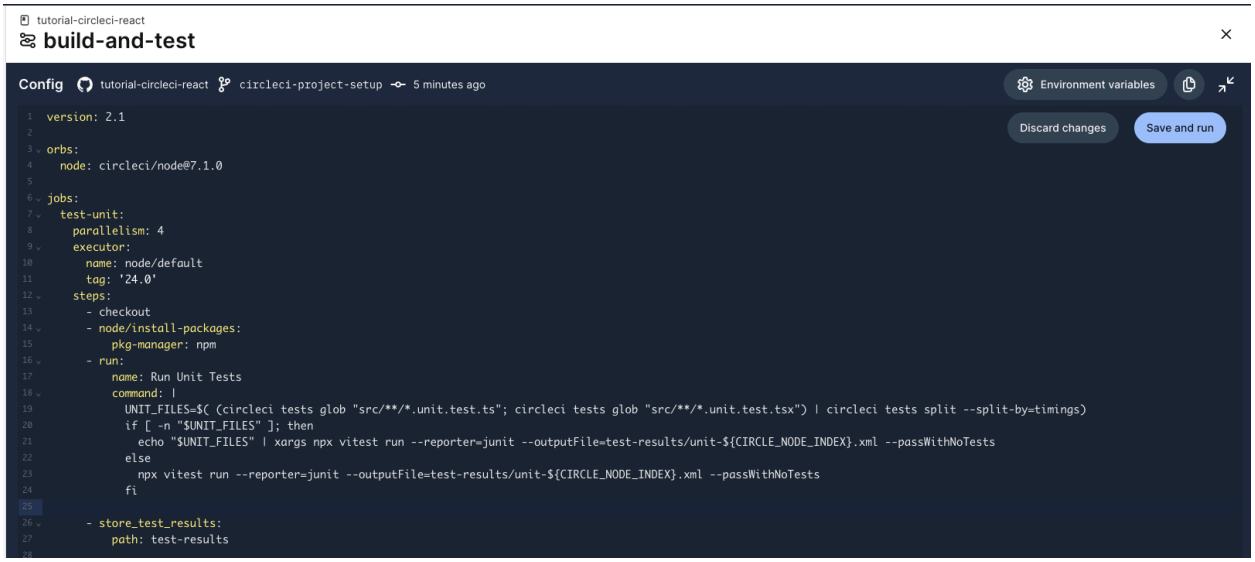
設定ファイルのtest-unitジョブを次のように変更してください。

Python

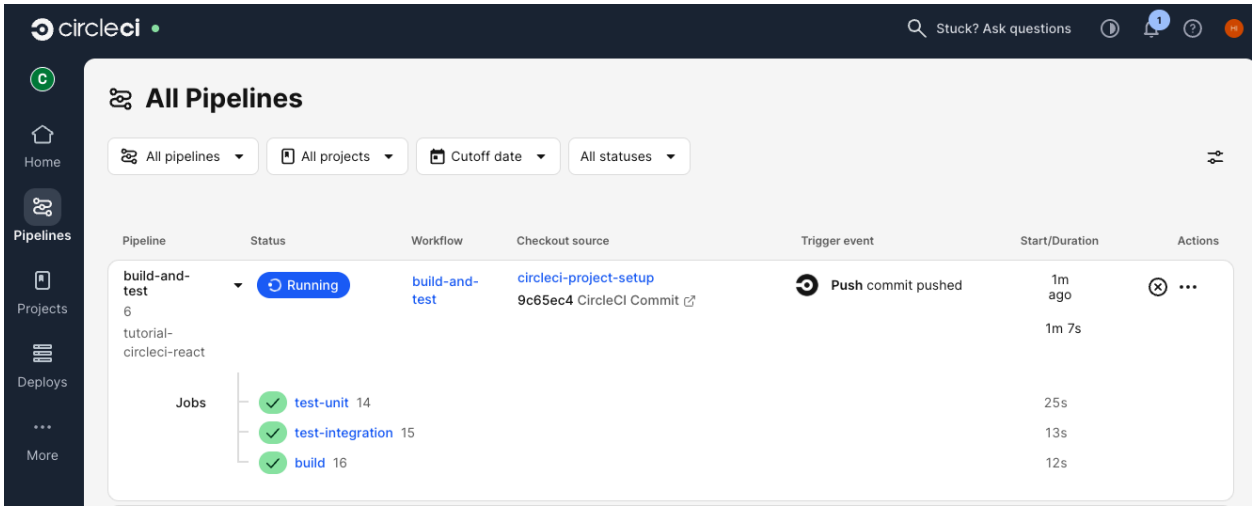
```
  test-unit:
    parallelism: 4
    executor:
      name: node/default
      tag: '24.0'
    steps:
```

```
      - checkout
      - node/install-packages:
          pkg-manager: npm
      - run:
          name: Run Unit Tests
          command: |
            UNIT_FILES=$( (circleci tests glob "src/**/*.unit.test.ts";
circleci tests glob "src/**/*.unit.test.tsx") | circleci tests split
--split-by=timings)
            if [ -n "$UNIT_FILES" ]; then
              echo "$UNIT_FILES" | xargs npx vitest run --reporter=junit
--outputFile=test-results/unit-${CIRCLE_NODE_INDEX}.xml --passWithNoTests
            else
              npx vitest run --reporter=junit
--outputFile=test-results/unit-${CIRCLE_NODE_INDEX}.xml --passWithNoTests
            fi

      - store_test_results:
          path: test-results
```

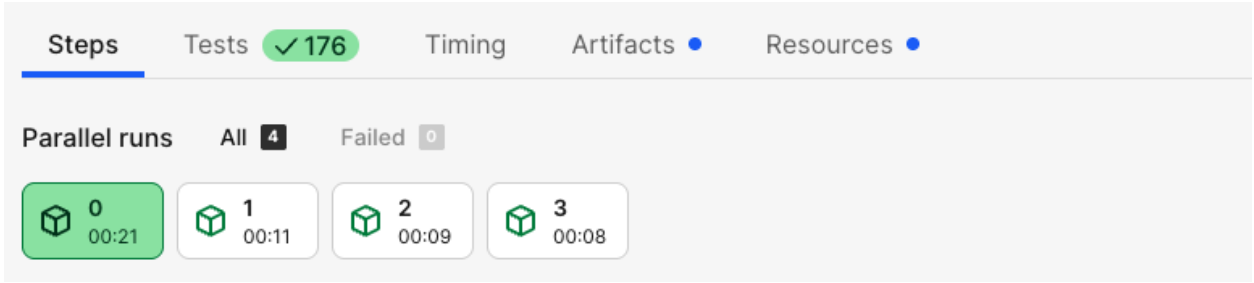
parallelism で並列実行する数を指定しています。加えてテストコマンドを変更し、ciecleci testsコマンドを利用し、テストファイルを分割実行するように設定しました。これによりユニットテストを4並列で実行するようになります。



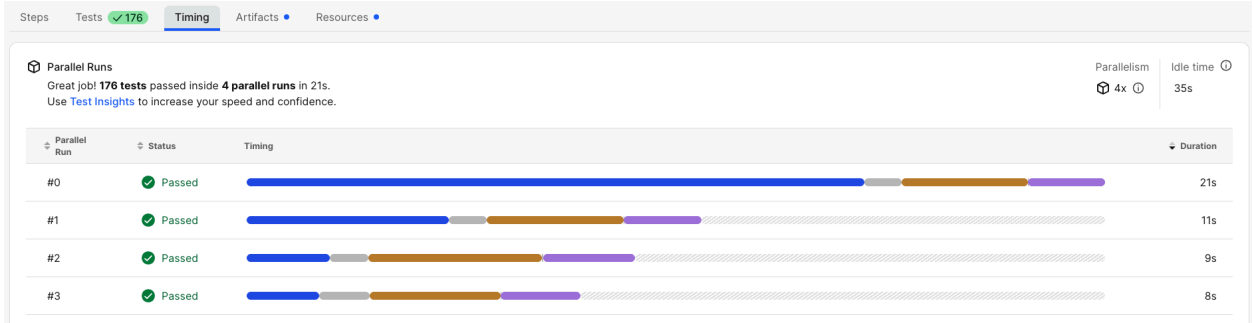
「Save and run」ボタンをクリックすると、ジョブが再び実行されます。「x」ボタンをクリックして編集画面をクローズします。その後表示されるAll Pipelinesページにて、一番上に表示されているパイプラインの「test-unit」テキストをクリックしてください。



Stepsタブに「Parallel runs」が追加表示されています。ここで並列実行したジョブごとの結果などをチェックできます。



Timingタブをクリックすると、それぞれの実行時間などが表示されます。CircleCIでのジョブ並列実行は、レポートデータを元に自動で最適化が行われます。そのため、最適化が行われるまではある程度ジョブごとに終了時間がバラつくことがあります。



AIを使ったCIエラー分析

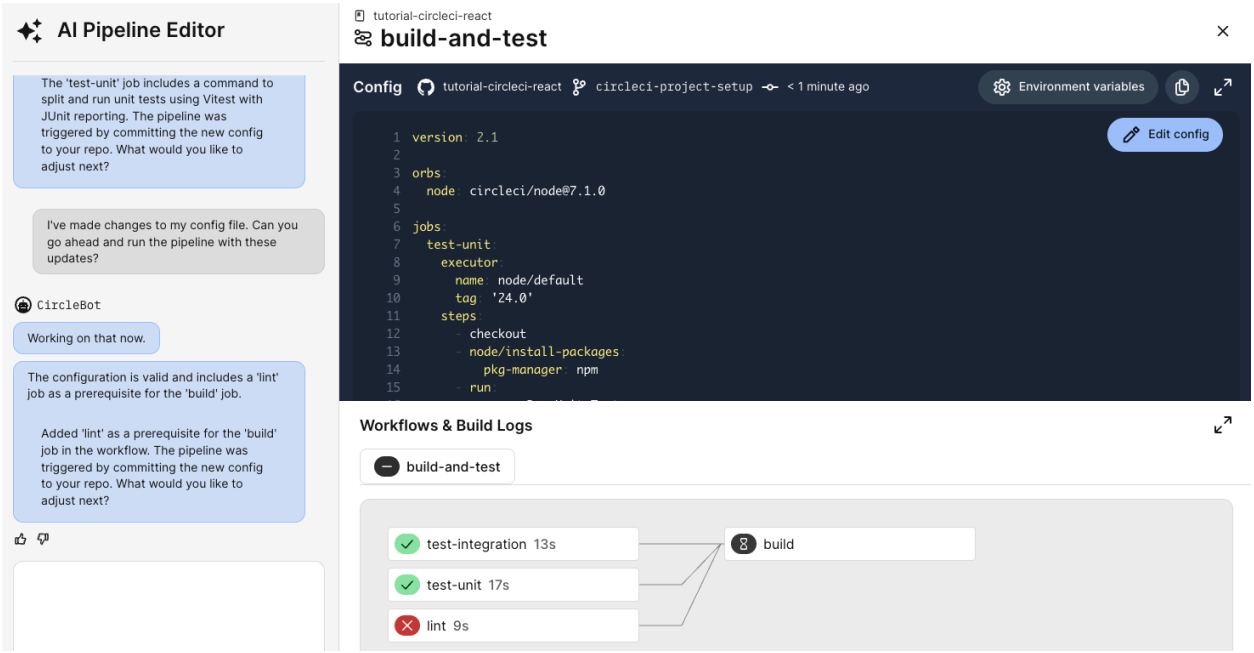
最後にテストやビルドが失敗した場合についてもみてみましょう。
AI Pipeline Editorを再度開き、以下のYAMLをコピーアンドペーストで貼り付けましょう。

```python
version: 2.1
```

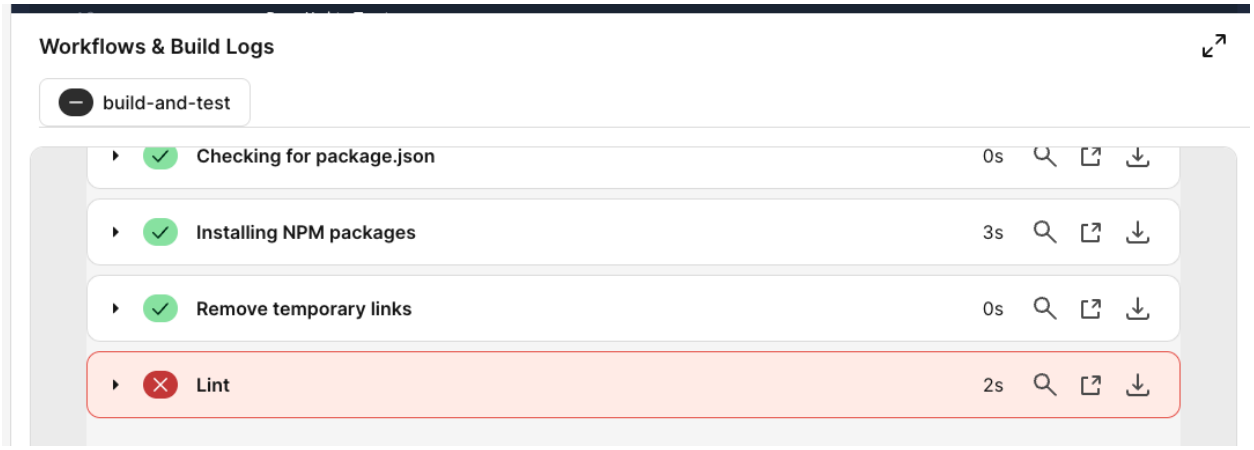
```yaml
orbs:
  node: circleci/node@7.1.0

jobs:
  test-unit:
    executor:
      name: node/default
      tag: '24.0'
    steps:
      - checkout
      - node/install-packages:
          pkg-manager: npm
      - run:
          name: Run Unit Tests
          command: npm run test:unit:ci

  test-integration:
    executor:
      name: node/default
      tag: '24.0'
    steps:
      - checkout
      - node/install-packages:
          pkg-manager: npm
      - run:
          name: Run Integration Tests
          command: npm run test:integration:ci

  build:
    executor:
      name: node/default
      tag: '24.0'
    steps:
      - checkout
      - node/install-packages:
          pkg-manager: npm
      - run:
          name: Build Application
          command: npm run build

  lint:
    executor: node/default
    steps:
      - checkout
      - node/install-packages:
          pkg-manager: npm
```

```
      - run:
          name: Lint
          command: npm run lint

workflows:
  version: 2
  build-and-test:
    jobs:
      - lint
      - test-integration
      - test-unit
      - build:
          requires:
            - test-integration
            - test-unit
            - lint
```

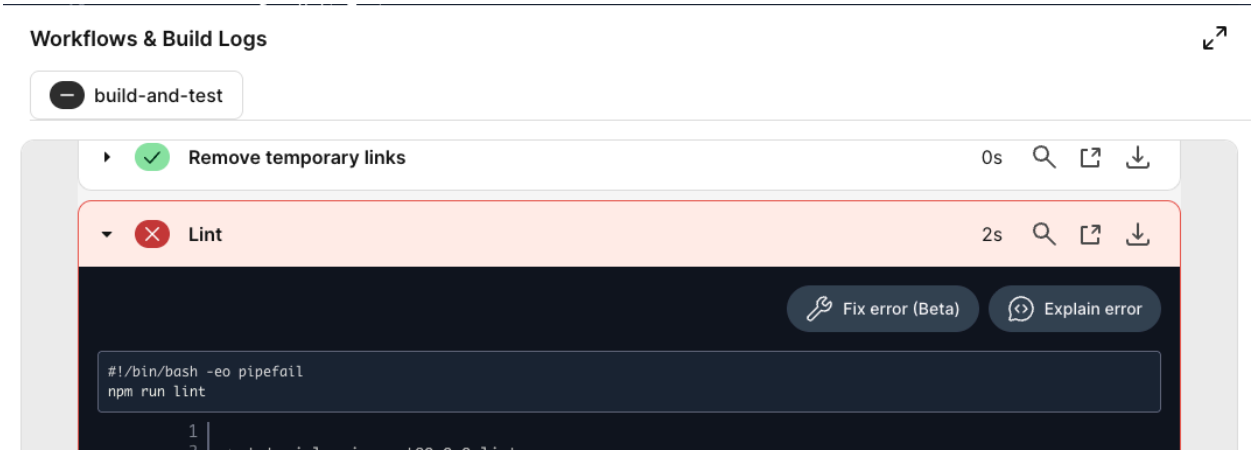
「Save and run」ボタンをクリックしてパイプラインを実行します。すると今度はlintジョブが新しく並列で実行されます。しかしこのジョブは下の画像のように失敗します。



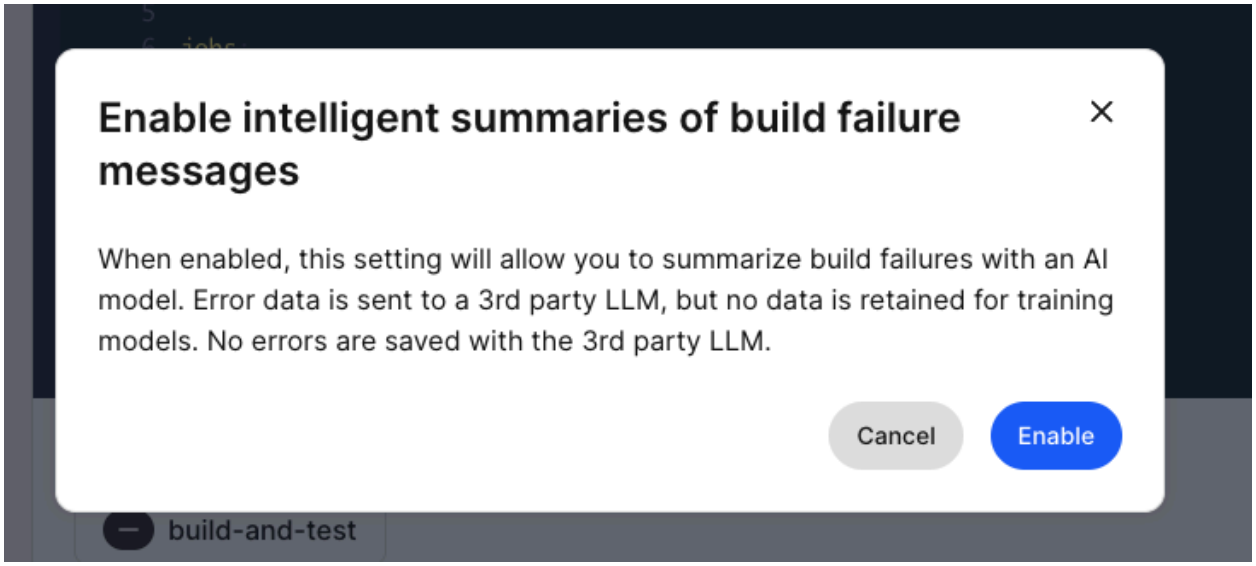
lintをクリックしてジョブの詳細画面を開きましょう。Lintステップが赤くなっていますので、クリックします。



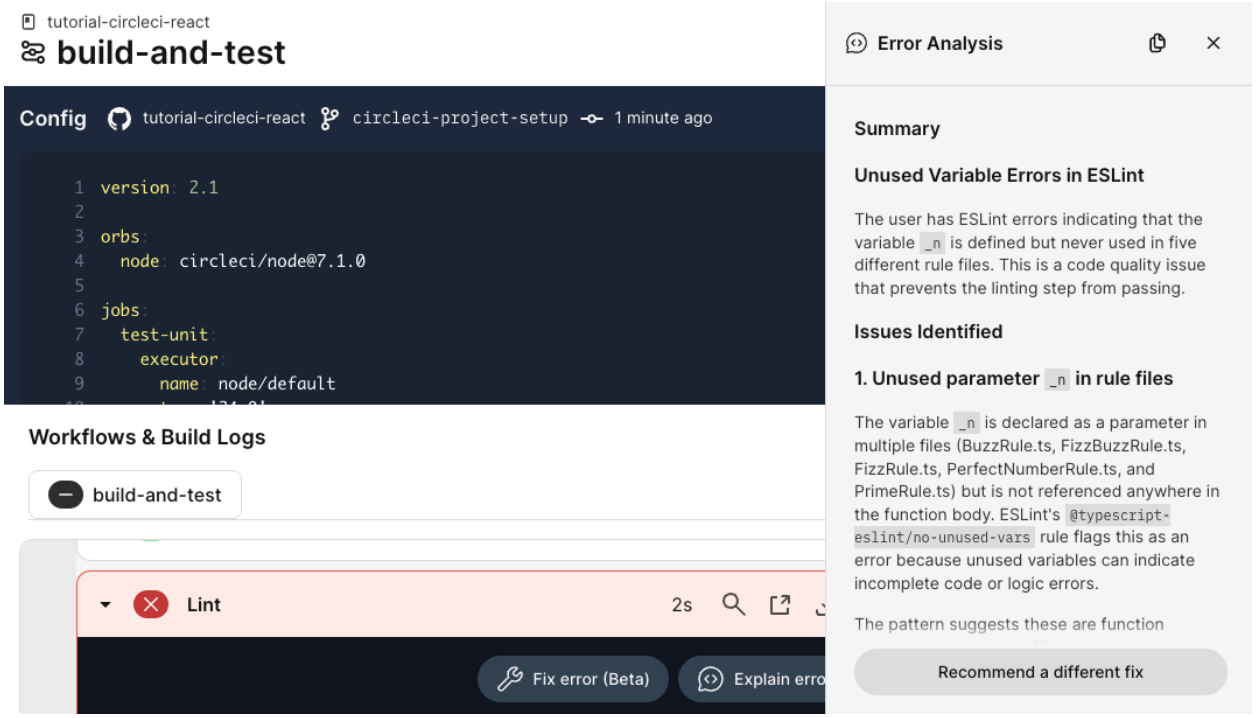
「Explain error」ボタンをクリックしましょう。これはCircleCIが生成AIを利用してエラーログ内容を分析し、改善方法をレポートする機能です。



エラーログを生成AIモデルに送信するため、初めて利用する際は確認画面が表示されます。コードの情報は送信されず、エラーの発生したジョブのログのみが送信され、データは学習に利用されたり保存されることはありません。「Enable」をクリックすると、分析が始まります。



分析が完了すると、画面右側にエラーの詳細と修正方法の提案が表示されます。



英語で表示されますが、ブラウザの翻訳機能を使って日本語で読むことができます。

このようにCI上で問題が発生した場合の調査や対応方法の検討についても、CircleCIの画面上で行うことができます。

このセクションで学べたこと

このセクションでは、実際のCI/CDパイプラインを構築し、CircleCIの主要機能を体験しました。

- 設定ファイルの編集方法: AI Pipeline Editorによる自動生成とYAMLファイルの手動編集の両方を習得
- 並列実行によるパイプライン最適化: 複数のジョブを同時実行し、依存関係のあるジョブは順次実行する仕組みを理解
- テスト結果の可視化: `store_test_results`でテスト結果を保存し、Test Summaryタブで確認
- **AI**を活用したエラー解決:「Explain this error」機能で、エラーの原因と修正案を取得

ここまでで、CI（継続的インテグレーション）の基礎を構築できました。最後のセクションでは、次のステップとしてCircleCIで実現できるCI / CDやDevOpsの種類・機能などについて紹介します。

4. Next Steps：さらなる自動化への道

本チュートリアルを通じて、CircleCIによる基本的なCI体験を実現できました。ビルド・テスト・リントの自動実行、並列処理による高速化、そしてAI機能を活用したエラー解析まで、開発サイクルを支える自動化の基礎を構築いたしました。
ここでは、次のステップとして習得できる機能や、より高度な自動化への展開方法をご紹介いたします。

4-1: 継続的デリバリー（CD）への拡張

CIで品質を担保したアプリケーションを、安全かつ効率的にデプロイする仕組みを構築できます。
実現できること

- アーティファクトの保存と配布：store_artifactsステップでビルド成果物を保存し、チームメンバーや他のシステムから簡単にアクセス可能にする
- 手動承認フロー：本番環境へのデプロイ前に承認ステップを挟み、意図しないリリースを防止する
- AWS・GCP・Azureへの自動デプロイ：専用Orbsを活用し、クラウド環境への安全なデプロイを数行の設定で実現

参考ドキュメント

- Deployment overview
- Storing build artifacts
- Configuration reference - Approval jobs

4-2: テストの最適化とパフォーマンス向上

テストスイートが大規模化した際の実行時間短縮や、より効率的なリソース活用を実現できます。
実現できること

- テスト分割（Test Splitting）：過去の実行時間データに基づき、テストファイルを複数の並列ジョブに最適配分する
- タイミングベースの分割：各ジョブの実行時間を均等化し、全体の待ち時間を最小化
- Docker Layer Caching：Dockerイメージのビルド時間を短縮し、開発サイクルを高速化

参考ドキュメント

- Test splitting and parallelism
- Test splitting tutorial
- Optimization reference

4-3: チーム開発への展開

組織全体で効果的にCircleCIを活用するための、通知機能や可視化ツールを導入できます。

実現できること

- Slack・Email通知の設定：ビルド成功・失敗を即座にチームに通知し、迅速な対応を可能にする
- Insights機能での可視化：ワークフローの実行時間、成功率、クレジット使用量などを時系列で追跡し、ボトルネックを特定
- 失敗したテストの分析：最も頻繁に失敗するテストや実行時間が長いテストを特定し、改善箇所を明確化

参考ドキュメント

- Notifications overview
- Use the Slack orb to set up notifications
- Using Insights

本チュートリアルで学んだ基礎を土台に、これらの機能を段階的に導入することで、より堅牢で効率的な開発フローを実現できます。各機能の詳細な設定方法や活用事例については、上記のドキュメントをご参照ください。