

CloudとSaaS型CI/CDを利用した開発ガイドライン

https://github.com/check-c-search/gh_pages_acr_result

2020-04-15

目次

1.はじめに	1
1.1.目的	1
1.2.背景	1
1.3.前提事項	1
1.4.全体像	2
2.クラウド環境の設定指針	3
2.1.概要	3
2.2. AlibabaCloudで設定することについて	3
2.3.各サーバの構成について	9
2.4.ネットワークの構成について	11
3.CI/CDツールの設定指針	13
3.1.概要	13
3.2.SonarQube連携について	15
3.3.Nexus連携について	16
3.4.設定ファイルで指定することについて	16
3.5.Webサイト(CircleCI)で指定することについて	22
4.ソースコード/開発チーム管理指針	43
4.1.概要	43
4.2.チーム管理	43

1. はじめに

1.1. 目的

このガイドラインでは、クラウドネイティブな開発を実現する上で要となるCloud環境とCI/CD（継続的インテグレーション/デリバリー）ツールを組み合わせた開発の指針を提示します。

1.2. 背景

CI/CDツール

- CI/CDツールについてはJenkinsが最も有名であり、多くの運用事例がありますが、実運用に足る構成を実現するためにはチューニングに多くの知識が必要となるため、作業の属人化が発生しやすく、また構築難易度・維持コスト共に高くなるという課題があります。
- 一方で、近年ではWerckerやCircleCIと言ったSaaS型のCI/CDツールの運用事例も増えています。また、主要なSaaS型ツールはチューニングが設定ファイル1つであることやサービス利用型の特性から、Jenkins運用で懸念される課題を改善することが期待されます。
- そのため、今回はCI/CDをSaaS型のツールを利用して実施します。

Cloud環境

- 近年のPublicクラウド市場は今だ20%以上の年率で成長していますが、徐々に従来のAWS一強状態からの変化が生まれており、MicrosoftAzureやAlibabaCloudといった脱AWSの選択肢をとる企業も増えてきているため、今後はマルチクラウドへの対応が求められることが予想されます。
- そのため、今回はCloud環境として「AlibabaCloud」を利用します。

1.3. 前提事項

このガイドラインは、下記の事項を前提としています。

- アプリケーションはJavaによるWebアプリケーション（以下AP）を対象とする
- APの開発フレームワークは「SpringBoot」を利用
- APのビルドツールは「Gradle」を利用

- ・ビルド成果物は単独のjarファイル
 - ・APサーバ上では、"java -jar"コマンドでjarを実行する処理をShellに記載し、サービス登録することでWebアプリケーションが動いている
-

1.4. 全体像

このガイドラインで取り扱う環境の全体像は下図の通りです。

[network] | network.png

Cloud環境・CI/CDツール以外の構成要素として、CI/CDのジョブ実行結果を確認する手段として「Slack」を、リソースのバージョン管理ツールとして「GitHub」を利用します。

2. クラウド環境の設定指針

2.1. 概要

このガイドラインでは、AlibabaCloudを利用します。

クラウド上には、下記2つのサーバを立ち上げます。

- Webアプリケーション用サーバ
 - 開発するAPを配置します。
- SonarQube・Nexus用サーバ
 - Nginxのリバースプロキシ制御下にSonarQubeとNexusを配置します。

また、各サーバへの接続はロードバランサによるパスルーティングによって振り分けます。

2.2. AlibabaCloudで設定することについて

2.2.1. VPC (Virtual Private Cloud)

AlibabaCloud上のプライベートネットワークです。

サーバを立ち上げる前に、まずはVPCを作成して他の仮想ネットワークからPJで扱う領域を分離します。

設定は、下図のようにAlibabaCloudサイトのVPCコンソールから行います。

2. クラウド環境の設定指針

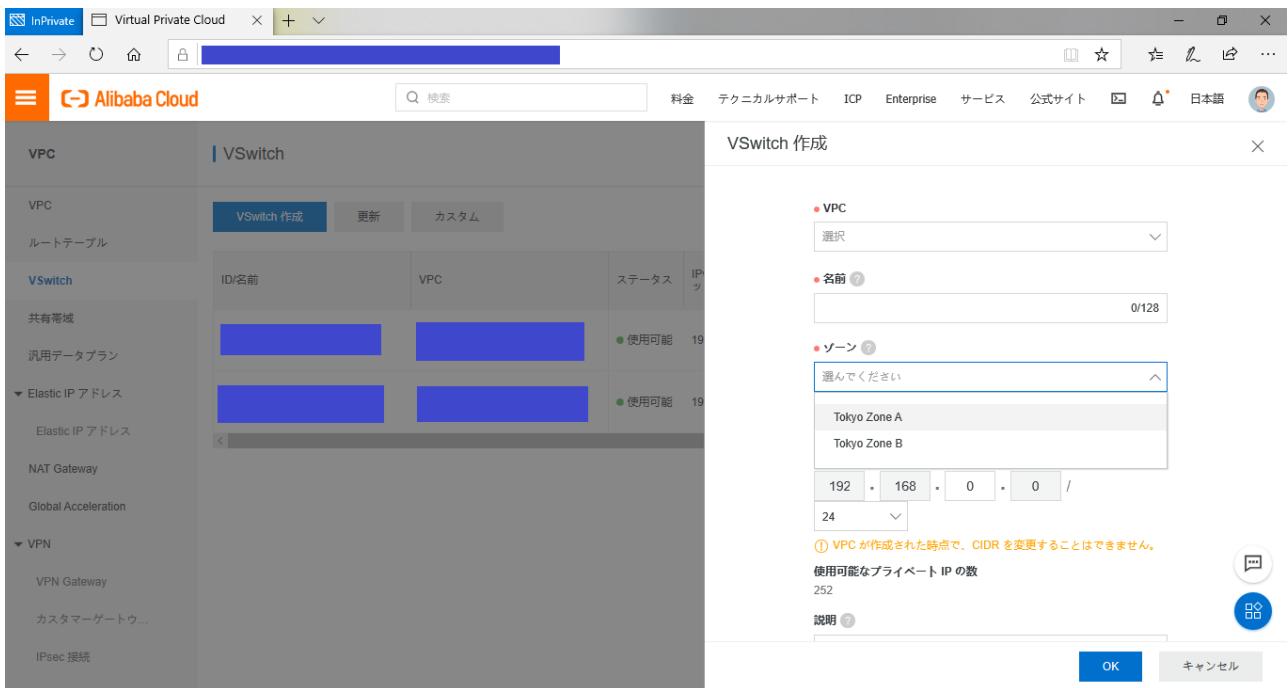
The screenshot shows the Alibaba Cloud VPC creation interface. On the left, a sidebar lists VPC-related services: Route Table, VSwitch, Shared Network, Shared Data Plan, Elastic IP Address, NAT Gateway, Global Acceleration, and VPN. The main panel is titled "VPC の作成" (Create VPC) and shows the configuration for a new VSwitch. The "リージョン" (Region) is set to "Japan (Tokyo)". The "名前" (Name) field contains "sample". Under "IPv4 CIDR ブロック" (IPv4 CIDR Block), the "デフォルト CIDR ブロック" (Default CIDR Block) is selected, and the value "192.168.0.0/16" is listed. Below it, a list of subnets includes "192.168.0.0/16", "172.16.0.0/12", and "10.0.0.0/8". The "ゾーン" (Zone) field is set to "選んでください" (Select). The "IPv4 CIDR ブロック" (IPv4 CIDR Block) for the zone is set to "192.168.0.0/24". A note at the bottom states: "① VPC が作成された時点で、CIDR を変更することはできません。" (Once a VPC is created, changing the CIDR is not possible.) The "OK" button is visible at the bottom right.

2.2.2. VSwitch

VPC上のネットワークデバイスです。

後述するECS(仮想サーバ)やSLB(ロードバランサ)等のインスタンスはVSwitch上に作成することになります。

設定は、VPC作成時にVPCコンソールから行う他、下図のようにVSwitchコンソールから行います



2.2.3. ECS(Elastic Compute Service)

AlibabaCloud上の仮想サーバがESCになります。
下図のようにESCコンソールより設定を行います。



2.2.4. セキュリティグループ

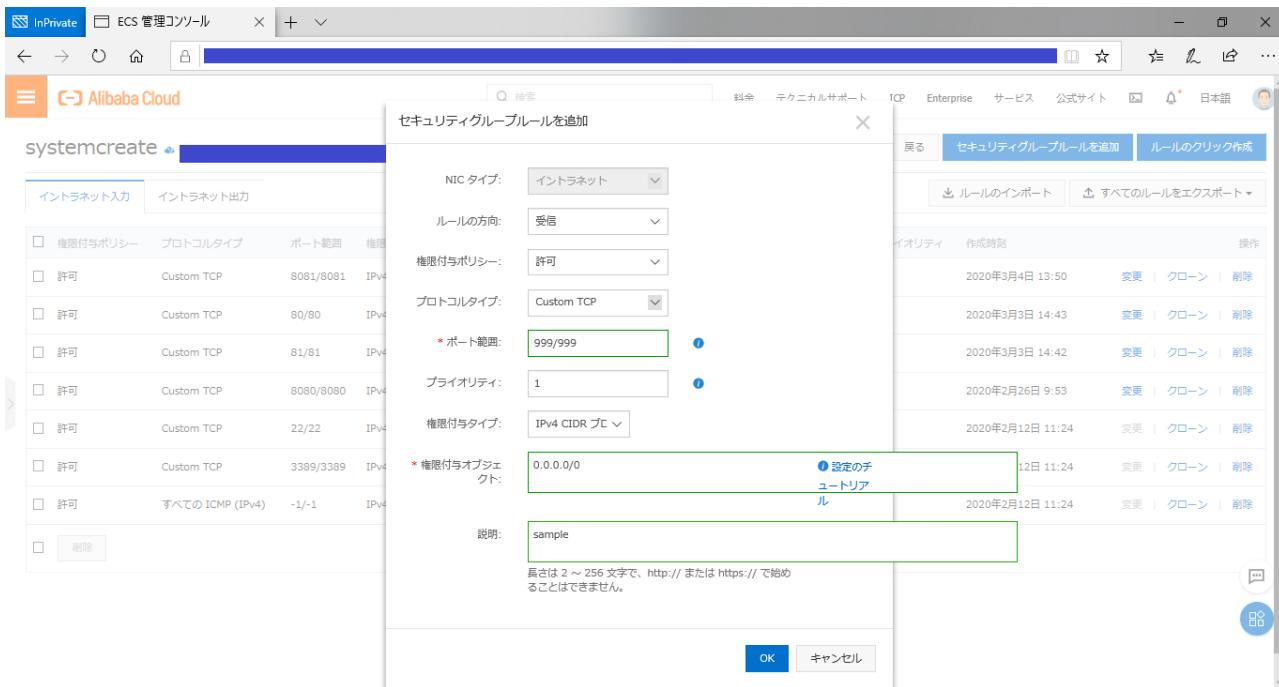
ESCにアクセス可能なポート番号を設定します。

セキュリティグループに設定されていないポートは通信に使用することができません。

また、ポート毎にインバウンドおよびアウトバウンドのアクセス可否を設定することが可能です。

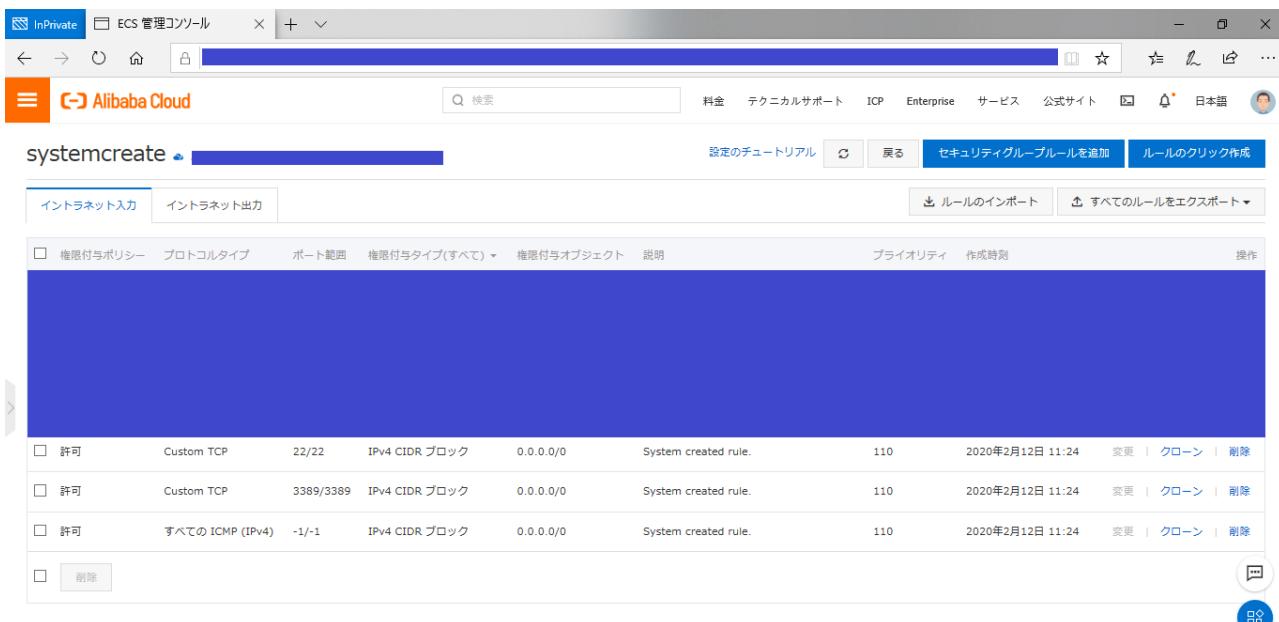
* インバウンド：外部から内部への通信（外部PCからの自社サービスへのアクセス）*

アウトバウンド：内部から外部への通信（インスタンスからの外部システムへのアクセス）



ESCの初期作成時には、下図のように「-

1」「22/22」「3389/3389」ポートのみが利用可能となっているため、適宜編集が必要です。



2.2.5. SLB(Server Load Balancer)

ロードバランサです。

下図のようにSLBコンソールよりSLBの購入を行います。

The screenshot shows the Alibaba Cloud SLB management console with the following details:

- Region Selection:** 日本 (Japan) is selected.
- Zone Type:** マルチゾーン (Multi-Zone).
- Primary Zone:** 日本ゾーン A (Japan Zone A).
- Backup Zone:** 日本ゾーン B (Japan Zone B).
- Instance Name:** sample.
- Instance Type:** インターネット (Internet).
- Instance Spec:** パフォーマンス共有型 (Performance Shared Type). A note below states: "This type of instance does not provide performance guarantees."
- IP Version:** IPv4.
- Network Connection Type:** トライフィックに応じた (Traffic-based) 带域幅に応じた (Bandwidth-based).
- Max Bandwidth:** 1 Mbps.
- Count:** 1.
- Cost:** \$0.016 /時間 (0.016 USD / hour).
- Buttons:** "今すぐ購入" (Buy Now) and "お問い合わせ" (Contact Support).

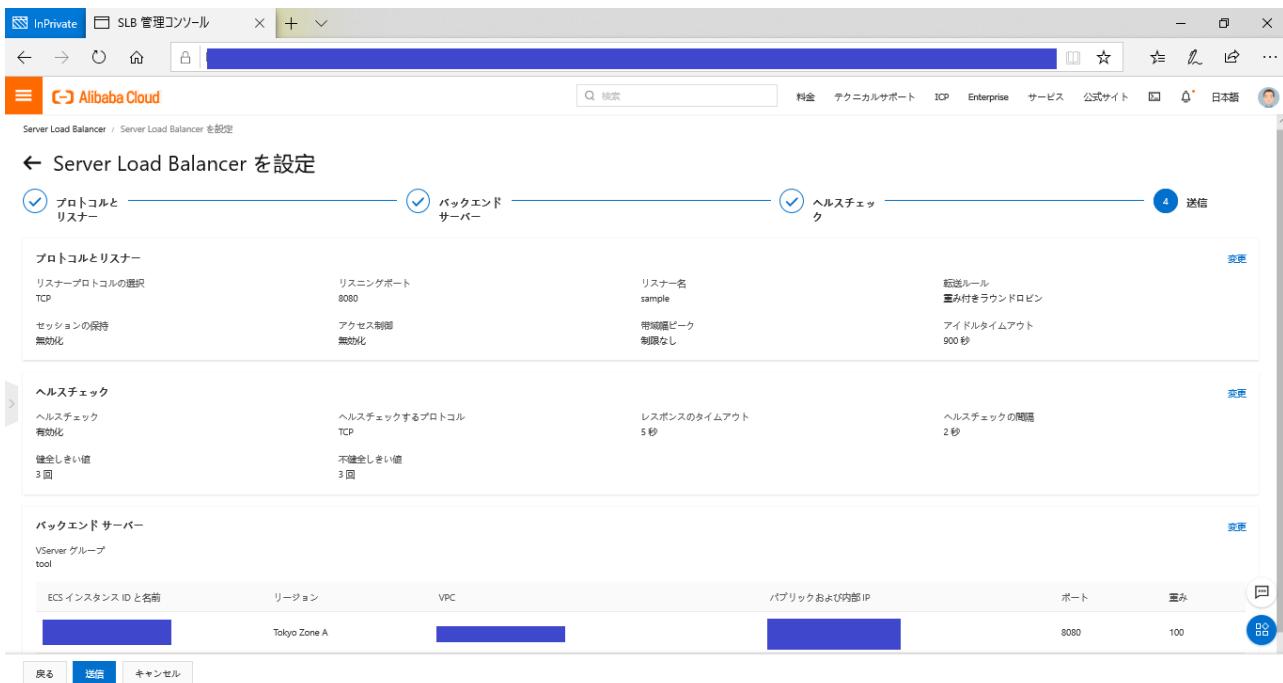
SLB購入後、「リスナーの作成」より振り分けの設定を行います。

2. クラウド環境の設定指針

The screenshot shows the Alibaba Cloud SLB management console. The URL in the address bar is `https://slb.console.aliyun.com/instance/listener/create?loadBalancerId=lb-123456789012345678`. The page title is "Listener creation". The top navigation bar includes links for "Home", "Search", "Billing", "Technical Support", "ICP", "Enterprise", "Services", "Official Site", and "Japanese". The main content area has tabs for "Instance Details", "Listener" (which is selected and highlighted with a red box), "Virtual Server Group", "Default Server Group", "Master-Slave Replication Server Group", and "Monitoring". Below these tabs is a table with columns: Listener Name, Frontend Protocol and Port, Backend Protocol and Port, Status, Health Check Status, Access Control, Monitor, Server Group, and Operations. A single row is shown for "HTTP:80" with status "Running" and health check status "Normal". Buttons at the bottom include "Create", "Stop", "Delete", and "Edit".

The screenshot shows the "Configure Server Load Balancer" page. The URL in the address bar is `https://slb.console.aliyun.com/configure?loadBalancerId=lb-123456789012345678`. The page title is "Configure Server Load Balancer". The top navigation bar is identical to the previous screenshot. The main content area is titled "Configure Listener". It shows four steps: "Protocol & Listener" (selected and highlighted with a red box), "Backend Server", "Health Check", and "Forwarding Rule". Step 1: "Protocol & Listener" contains fields for "Protocol & Listener Name" (Protocol: TCP, Listener Name: sample), "Backend Port" (8080), "Health Check" (Protocol: TCP, Interval: 5s, Unhealthy threshold: 3), and "Forwarding Rule" (Forwarding rule: 転送ルール (Forwarding Rule), Type: 重み付きラウンドロビン (Weighted Round Robin)). Step 2: "Backend Server" lists "ECS Instance ID and Name" (tool), "Region" (Tokyo Zone A), "VPC" (Public IP or Internal IP), "Port" (8080), and "Weight" (100). Step 3: "Health Check" and Step 4: "Forwarding Rule" are partially visible. At the bottom are "Back", "Next", and "Cancel" buttons.

また、「転送ルールの設定」よりパスルーティングを設定します。



2.3. 各サーバの構成について

このガイドラインで取り扱うサーバについて説明します。==== Webアプリケーション用サーバ
このサーバでは、開発したAPが配置されています。* サーバスペック サーバは下記のよう設定しています。

表 1. サーバ

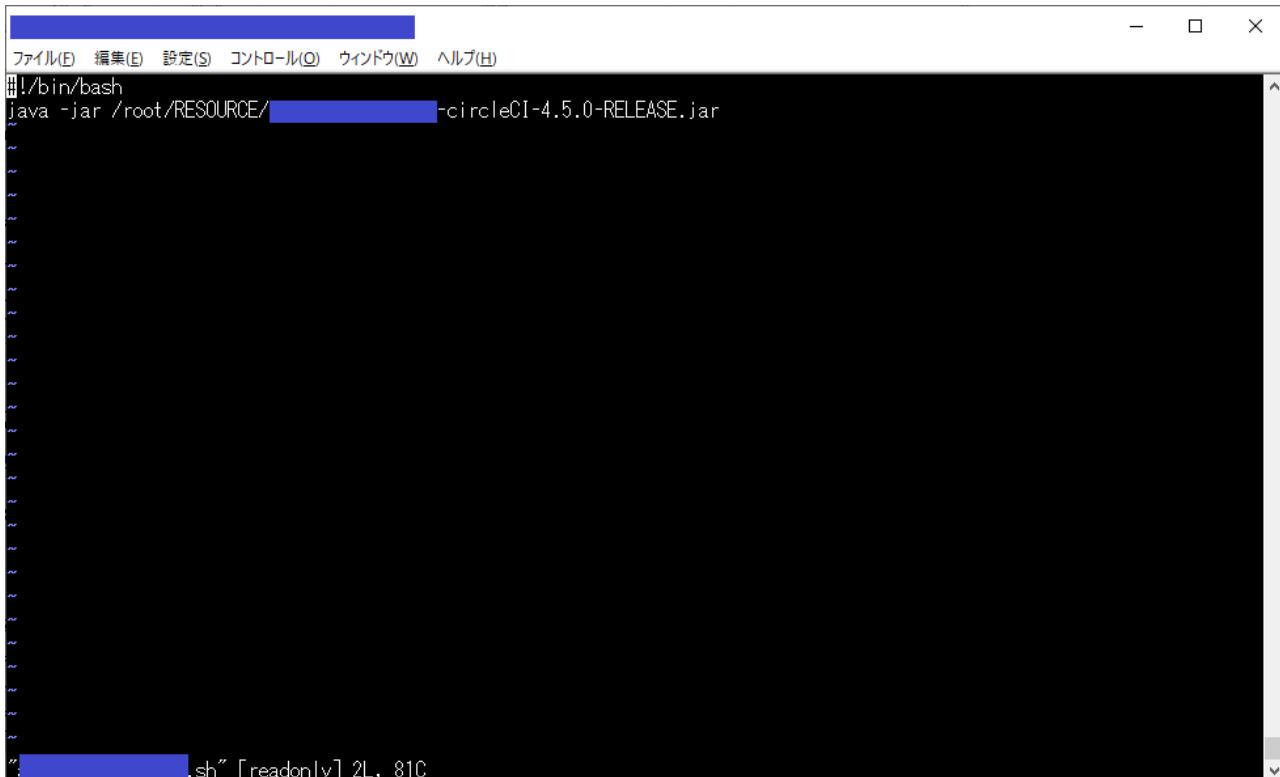
CPU	メモリ	OS	帯域幅	SSH接続
1コア	1GiB	CentOS 7.7 64-bit	1Mbps (ピーク値)	キーペアによる公開鍵認証

表 2. セキュリティグループ

入力/出力	ポート番号	権限付与IPアドレス
入力	80/80	0.0.0.0/0 ※検証のためフルオープン

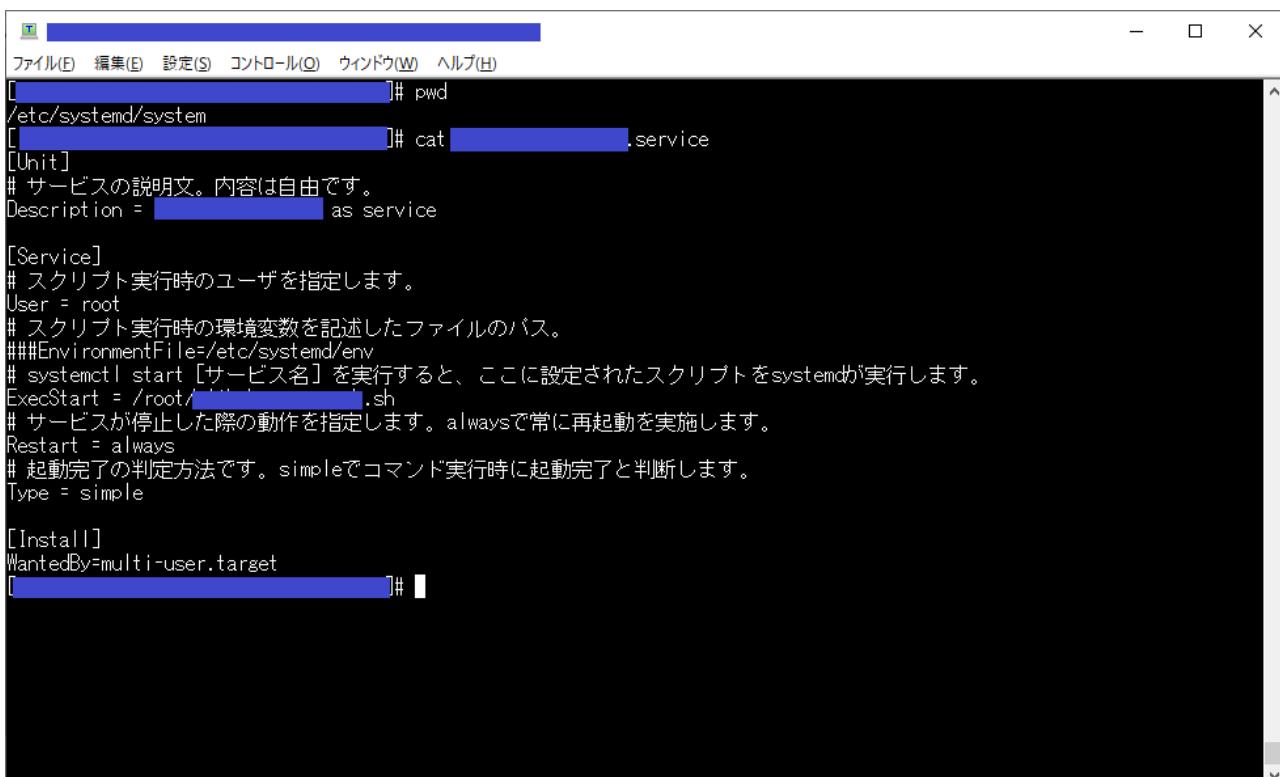
- アプリケーション起動方式 APは、下図のようにAPを「Java -jar」コマンドにより起動するShellにより起動されます。

2. クラウド環境の設定指針



```
!./bin/bash
java -jar /root/RESOURCE/[REDACTED]-circleCI-4.5.0-RELEASE.jar
```

また、AP起動Shellは下図のように「/etc/systemd/system」配下にサービス登録がされており、「systemctl enable」コマンドによりESCインスタンス起動時に自動実行されるようになっています。



```
[root@AP ~]# pwd
/etc/systemd/system
[root@AP ~]# cat [REDACTED].service
[Unit]
# サービスの説明文。内容は自由です。
Description = [REDACTED] as service

[Service]
# スクリプト実行時のユーザを指定します。
User = root
# スクリプト実行時の環境変数を記述したファイルのパス。
###EnvironmentFile=/etc/systemd/env
# systemctl start [サービス名] を実行すると、ここに設定されたスクリプトをsystemdが実行します。
ExecStart = /root/[REDACTED].sh
# サービスが停止した際の動作を指定します。alwaysで常に再起動を実施します。
Restart = always
# 起動完了の判定方法です。simpleでコマンド実行時に起動完了と判断します。
Type = simple

[Install]
WantedBy=multi-user.target
[root@AP ~]#
```

2.3.1. SonarQube・Nexus用サーバ

このサーバでは、CI/CDのパイプライン実行後にテスト結果とビルド資産を配置するためにSonarQubeとNexusが起動しています。* サーバスペック サーバは下記のよう設定しています。

表3. サーバ

CPU	メモリ	OS	帯域幅	SSH接続
2コア	4GiB	CentOS 7.7 64-bit	2Mbps (ピーク値)	キーペアによる公開鍵認証

表4. セキュリティグループ

入力/出力	ポート番号	権限付与IPアドレス
入力	80/80	0.0.0.0/0 ※検証のためフルオープン
入力	8080/8080	0.0.0.0/0 ※Nginxのポート番号

- アプリケーション起動方式

SonarQubeとNexusはNginxによるリバースプロキシ制御下で起動しています。

各アプリケーションは、下図に示す設定でdocker起動しています。

```

[...]
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
81db0ccb8f0a4      nginx              "nginx -g 'daemon off;'"   2 weeks ago       Restarting (1)   16 seconds ago   5432/tcp           dev-support-toolchain_nginx_1
1b51f04720ae        postgres            "docker-entrypoint.s..."   2 weeks ago       Up 3 days         5432/tcp           dev-support-toolchain_database_1
79e79bb8ec5e        sonatype/nexus3    "sh -c ${SONATYPE_DI..."   2 weeks ago       Up 3 days         8081/tcp           dev-support-toolchain_nexus_1
9d79b80309e0        sonarqube          "./bin/run.sh"        2 weeks ago       Exited (127)     5 days ago        9000/tcp           dev-support-toolchain_sonarqube_1
[...]

```

2.4. ネットワークの構成について

各サーバは、下記のようにSLBによるパスルーティングによって外部から接続するように設定しています。

表5. パスルーティング設定

ドメイン	URL	接続先ポート
SLBのIP	/nexus	SonarQube・Nexus用サーバの8080番 ※Nginxのポート番号

ドメイン	URL	接続先ポート
SLBのIP	/sonarqube	SonarQube・Nexus用サーバの8080番 ※Nginxのポート番号
SLBのIP	/web	Webアプリケーション用サーバの80番



SonarQube・Nexus用サーバではNginxのリバースプロキシにより、
「~/sonarqube」は9000番ポートに転送される ※SonarQubeのポート番号
また、「~/nexus」は8001番ポートに転送される ※Nexusのポート番号

3. CI/CDツールの設定指針

3.1. 概要

- CI/CDツールでは、Github上の対象リポジトリについて、ブランチ毎に実行するパイプラインの処理（ビルド・テスト・デプロイ…etc）を指定します。
- SaaS型CI/CDツールを利用する場合、パイプラインで実行する処理は全て設定ファイルに記載し、設定ファイルを対象リポジトリのルートディレクトリ配下に配置することで、Githubでの動作を契機としてパイプラインが実行されます。

SaaS型ツールの代表例としては、特に運用事例の多い下記3ツールが挙げられます。このガイドラインでは、初期構築のしやすさの観点から「CircleCI」を例として指針を提示します。

表 6. SaaS型CI/CDツール



	CI/CDツール	URL	費用
1	Wercker	https://app.wercker.com	無料 (Oracleが買収したため今後は不明)
2	TraviceCI	https://travis-ci.org	\$118/month
3	CircleCI	https://circleci.com	基本無料(1環境、ビルト時間:1500分/月) 1環境増設ごとに\$50/月

3.1.1. CI/CDで行うことについて

ローカル環境で作成・修正したアプリケーション（以降AP）に対して、下記の処理を自動実行します。

ビルドチェック

ビルドツールによるビルドを行うことで、各種ライブラリとの依存関係が解消されており、APが実行環境で動作可能な状態であることを保証します。

テスト

ローカル環境で使用したテストコードでビルドしたAPのJunitテストを行います。
また、SonarQubeによる静的検証（Checkstyle、SpotBugs）を行います。

ビルド資産の管理

ビルドに成功したAPをNexusに登録することで、ビルド資産のバージョン管理を行います。

デプロイ

ビルドに成功したAPを実行環境にデプロイします。

3.1.2. ブランチ戦略

アプリケーションはGitHubのリポジトリで管理し、状態毎に下記のブランチを切って管理を行います。

表 7. ブランチ戦略

ブランチ	ブランチ名	説明	作成/削除タイミング
masterブランチ	master	常に最新版のブランチ	常に存在する
issueブランチ	issue<簡単な説明>	作業用のブランチ	issueに基づいて、作業担当者がmasterブランチより作成する PRをmergeした際にレビュー担当者が削除する
releaseブランチ	release	リリース用のブランチ	リリースの際にライブラリ管理者がmasterブランチより作成する リリース完了後にライブラリ管理者が削除する

3.1.3. タグによるリリースバージョン管理

アプリケーションのリリースバージョンはGitHubのタグで管理し、CI/CDのパイプライン内で下記の様式に従ってタグを生成する。

表 8. タグ

ブランチ	タグ名	作成タイミング
master	version-<メジャーバージョン>.<マイナーバージョン>.<リビジョンバージョン>-SNAPSHOT	PRをmergeした際
master	version-<メジャーバージョン>.<マイナーバージョン>.<リビジョンバージョン>-RELEASE	releaseした際

3.1.4. ブランチ毎のCI/CDフロー

パイプライン上ではブランチ毎に下記の処理を実行する。

表 9. CI/CDフロー

ブランチ	パイプライン起動の契機	実行処理
issueブランチ	ローカル環境よりGitHub上にpushした際	<ul style="list-style-type: none"> ・ ビルドチェック ・ Junitテスト ・ SonarQubeへのJunitテスト結果連携、静的検証
masterブランチ	PRをmergeした際	<ul style="list-style-type: none"> ・ ビルドチェック ・ Junitテスト ・ SonarQubeへのJunitテスト結果連携、静的検証 ・ Nexusへのライブラリ登録 ・ 最新のmasterブランチに対してタグ打ち (名 : version-X.X.X-SNAPSHOT)
releaseブランチ	ローカル環境よりGitHub上にpushした際	<ul style="list-style-type: none"> ・ ビルドチェック ・ Junitテスト ・ Nexusへのライブラリ登録 ・ 最新のmasterブランチに対してタグ打ち (名 : version-X.X.X-RELEASE)

3.2. SonarQube連携について

- ・ Junitテストの結果を連携する
- ・ Checkstyleによる静的検証を実施する
- ・ SpotBugsによる静的検証を実施する
- ・ カバレッジ情報を表示する

3.3. Nexus連携について

- PRをmergeしたタイミングでSNAPSHOTリポジトリにビルド資産を連携する
- releaseブランチをGitHub上に作成したタイミングでRELEASEリポジトリにビルド資産を連携する

3.4. 設定ファイルで指定することについて

3.4.1. 設定ファイルの記載方法

CircleCIの設定ファイル「config.yml」の基本構成は下記のようになっており、適宜PJの方針に合わせて必要な処理を記述する。

設定ファイルは、大きく下記の3ブロックより構成される。

1. パイプラインブロック
→ Job単位で「どのコンテナ環境」を利用して、「どんな処理」を行うかを記述する。
2. ワークフローブロック
→ 各Job間の前後関係・依存関係や各Jobの処理対象ブランチを記述する。
3. 定時実行ワークフローブロック
→ cronベースで「どのタイミング」で起動するのか、またワークフローブロック同様に「どのJob」が「どんな依存関係」で「どのブランチ」を対象に処理するのかを記述する。



config.yml: 基本構成

```

### 基本構成
#CircleCIのバージョン指定
version: 2.1

#以下パイプラインの処理記述
jobs:
  build:
    #実行環境
    docker:
      - image: Dockerイメージ

    #以下、処理「build」の実処理
    steps:
      - checkout ※対象コードチェックアウト
      - 処理②...

  deploy:
    docker:
      - image: Dockerイメージ
    steps:
      - checkout
      - 処理②...

#以下ワークフローの記載
workflows:
  version: 2
  workflow:
    jobs:
      - build:
          filters:
            branches:
              only:
                - 対象とするブランチ名
        post-steps:
          - 定常実行処理
      - deploy:
          requires: ※先行処理
          - build

#以下、定時実行ワークフローの記載
nightly:
  triggers:
    - schedule:
        cron: "25 * * * *"
    filters:
      branches:
        only:
          - master
  jobs:
    - build:
        filters:
          branches:
            only:
              - /^issue\/.+/
    - deploy:
        requires:
          - build

```

設定ファイル記述の詳細については、下記の公式ドキュメントを参照

- ・<CircleCI公式ドキュメント>
<https://circleci.com/docs/>



特に下記ページを起点にすると、設定ファイル全体の内どの部分にあたる記述なのかをイメージしやすく、作業を効率よく進められます。

- ・「CircleCIサイト>リファレンスページURL」
<https://circleci.com/docs/ja/2.0/configuration-reference/#section=reference>

3.4.2. SonarQube連携方法

- ① build.gradleファイルに下記の記載を行う
- ② config.ymlファイルに下記の記載を行う

build.gradle:sonarqubeプラグインの利用

```
plugins {
    id "org.sonarqube" version "2.8"
}
...
sonarqube {
    properties {
        property "sonar.jacoco.reportPath", "${project.buildDir}/jacoco/test.exec"
    }
}
```

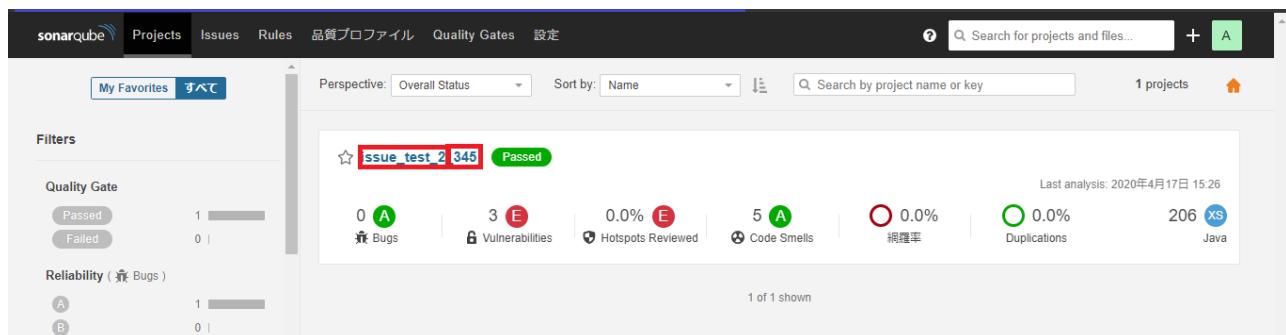
config.yml:sonarqubeタスクの呼び出し

```
##"./gradlew sonarqube"コマンドによるSonarQubeサイトで設定した検証の実施と結果連携を行う。
jobs:
...
sonarqube:
...
steps:
- checkout
- run:
    name: analyze by SonarQube
    command: |
        ./gradlew clean sonarqube \
        -Dsonar.host.url=$SONAR_HOST_URL \
        -Dsonar.jdbc.url=$SONAR_JDBC_URL \
        -Dsonar.jdbc.driverClassName=$SONAR_JDBC_DRIVER \
        -Dsonar.jdbc.username=$SONAR_JDBC_USERNAME \
        -Dsonar.jdbc.password=$SONAR_JDBC_PASSWORD \
        -Dsonar.projectName="${CIRCLE_BRANCH}"_"${CIRCLE_BUILD_NUM}"
```

表 10. sonarqubeタスクのプロパティ

プロパティ	説明	変更要否
sonar.jacoco.reportPath	Jacocoが生成するレポートファイルへのパス ※Jacoco : Java のコードカバレッジライブラリ	デフォルト値のため変更不要
sonar.host.url	接続先SonarQubのURL	PJの設定に併せて変更する ※環境変数の設定については後述します
sonar.jdbc.driverClassName	SonarQubが利用するJDBCのドライバクラス名	PJの設定に併せて変更する ※環境変数の設定については後述します
sonar.jdbc.username	SonarQubが利用するDBのユーザ名	PJの設定に併せて変更する ※環境変数の設定については後述します
sonar.jdbc.password	SonarQubが利用するDBのパスワード	PJの設定に併せて変更する ※環境変数の設定については後述します
sonar.projectName	SonarQubサイトで表示される名称	今回は下記のCircleCIのデフォルト環境変数により自動で定義するようにしているため変更は不要 「CIRCLE_BRANCH」 CI/CD対象のブランチ名 「CIRCLE_BUILD_NUM」 CircleCI上のビルド回数

SonarQub画面には下図のように表示される。



sonarqubeタスクの詳細な記載方法は、下記公式ドキュメントを参照
<https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-gradle/>

3.4.3. Nexus連携方法

- ① build.gradleファイルに下記の記載を行う
- ② config.ymlファイルに下記の記載を行う

build.gradle:maven-publishプラグインの利用

```
plugins {
    id "maven-publish"
}

...
group 'ビルド成果物のパッケージ名'
version '1.0.0-SNAPSHOT'
archivesBaseName = 'ビルド成果物のファイル名'
...

publishing {
    publications {
        mavenJava(MavenPublication) {
            groupId = group
            artifactId = archivesBaseName
            from components.java
        }
    }
    repositories {
        maven {
            url System.getenv("NEXUS_URL")
            credentials {
                username = System.getenv("NEXUS_USERNAME")
                password = System.getenv("NEXUS_USERPASSWORD")
            }
        }
    }
}
}
```



publishingタスクの詳細な記載方法は、下記公式ドキュメントを参照
https://docs.gradle.org/current/userguide/publishing_maven.html#publishing_maven:complete_example

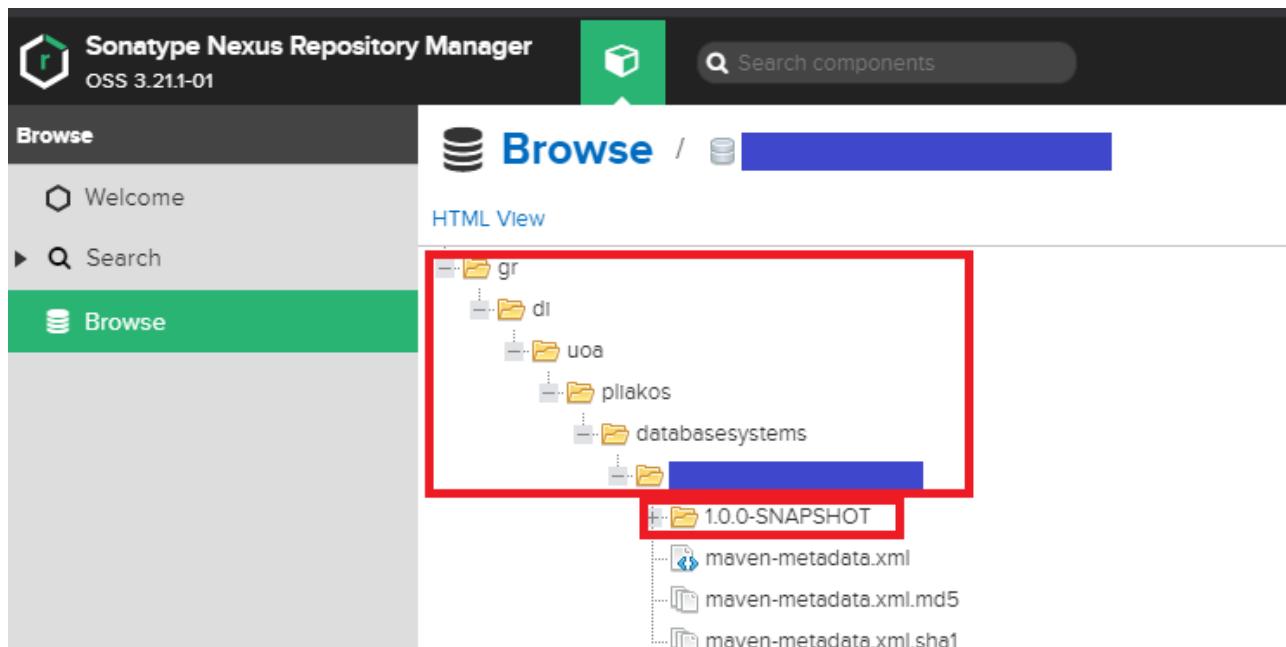
config.yml:publishingタスクの呼び出し

```
##"./gradlew upload"コマンドによるビルド成果物のNexus登録を行う。
jobs:
...
nexus:
...
steps:
- checkout
- run:
    name: upload to NEXUS
    command: |
        ./gradlew clean build publish
```

表 11. publishingタスクのプロパティ

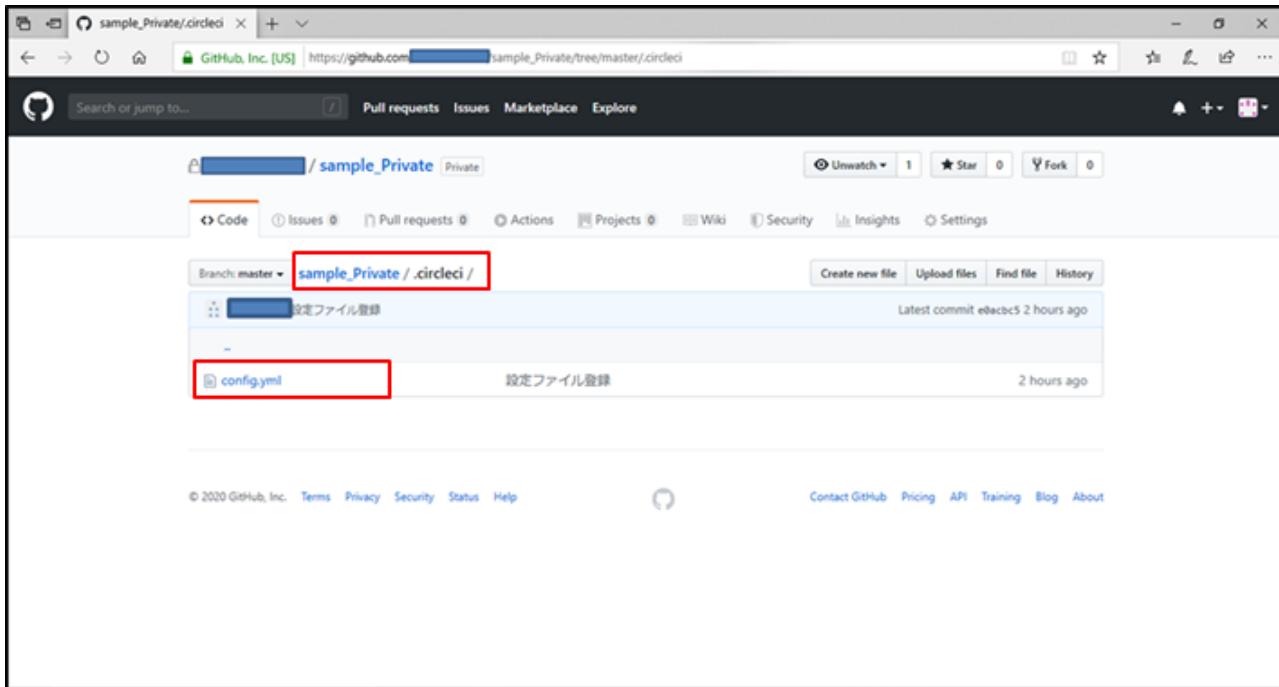
プロパティ	説明	変更要否
groupId	Nexusに登録する際のパッケージ名	PJの設定に併せて変更する ※このガイドラインでは、「group」の値を設定しています
artifactId	Nexusに登録する際のファイル名	PJの設定に併せて変更する ※このガイドラインでは、「archivesBaseName」の値を設定しています
url	接続先SonarQubのURL	PJの設定に併せて変更する ※環境変数の設定については後述します
username	Nexusのログインユーザ名	PJの設定に併せて変更する ※環境変数の設定については後述します
password	Nexusのログインパスワード	PJの設定に併せて変更する ※環境変数の設定については後述します

Nexus画面には下図のように表示される。



3.4.4. 設定ファイルの配置

作成した設定ファイルは、ルートディレクトリ直下の「.circleci」ディレクトリ配下に配置することでGitHubへの操作を契機としCircleCIによりCI/CDのパイプラインが実行されます。



3.5. Webサイト(CircleCI)で指定することについて

- 「CircleCIサイトURL」 <https://circleci.com>

ログイン方法等については、公式ドキュメントを参照

* <CircleCI公式ドキュメント> <https://circleci.com/docs/>

3.5.1. 対象リポジトリの登録

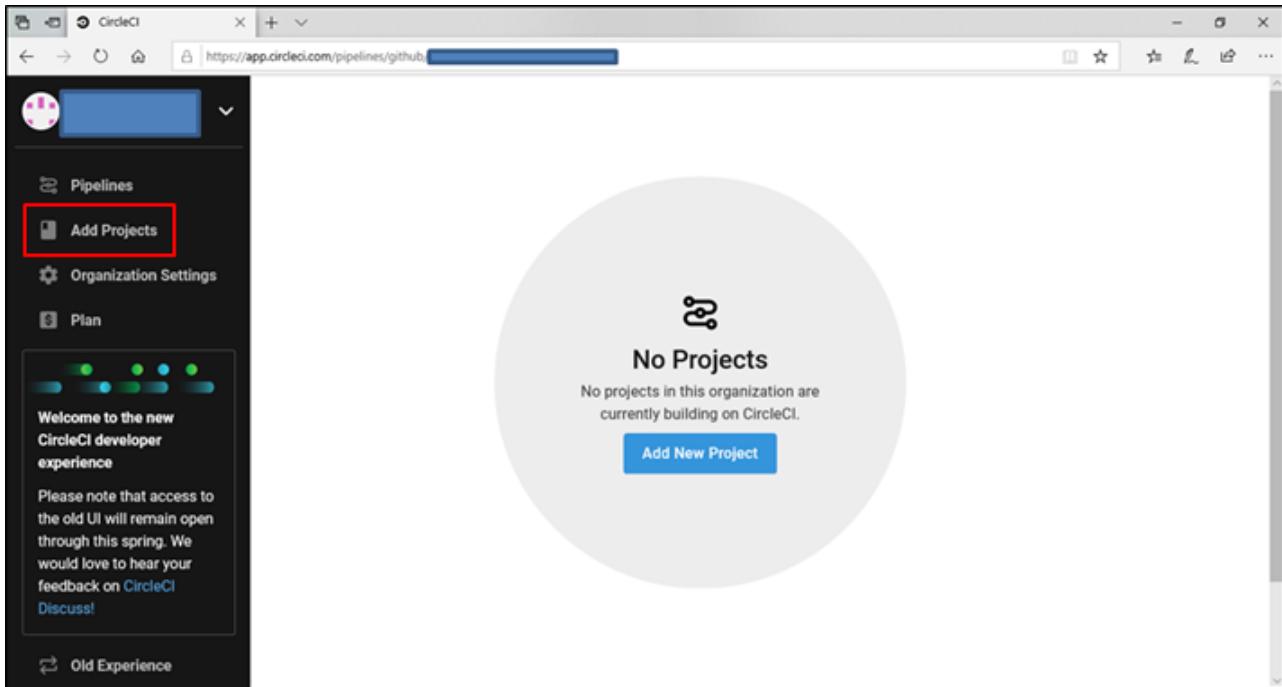
「Add Projects」よりCI/CD対象のリポジトリ登録を行います。

Github上のリポジトリの公開設定に関わらず登録が可能です。

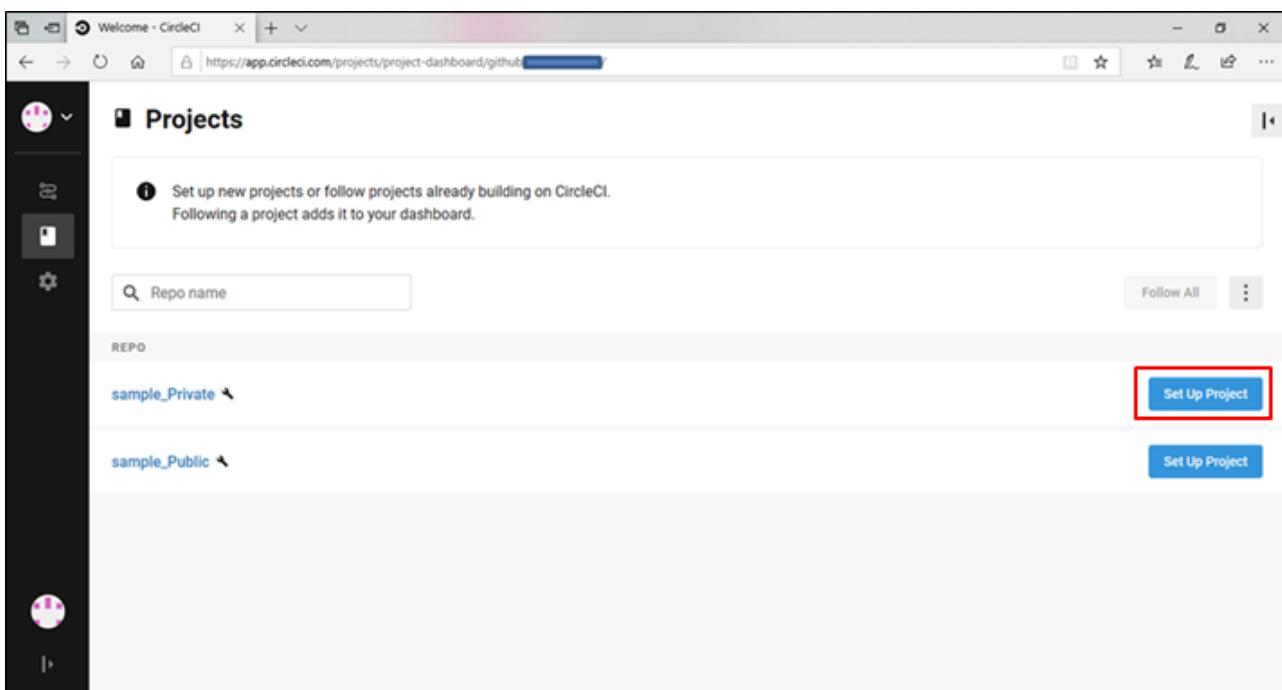
また、後述する設定を行うことで自身が所属するOrganization所有のリポジトリを権限に応じて利用可能です

◦

1 ダッシュボード画面より「Add Projects」を選択する。

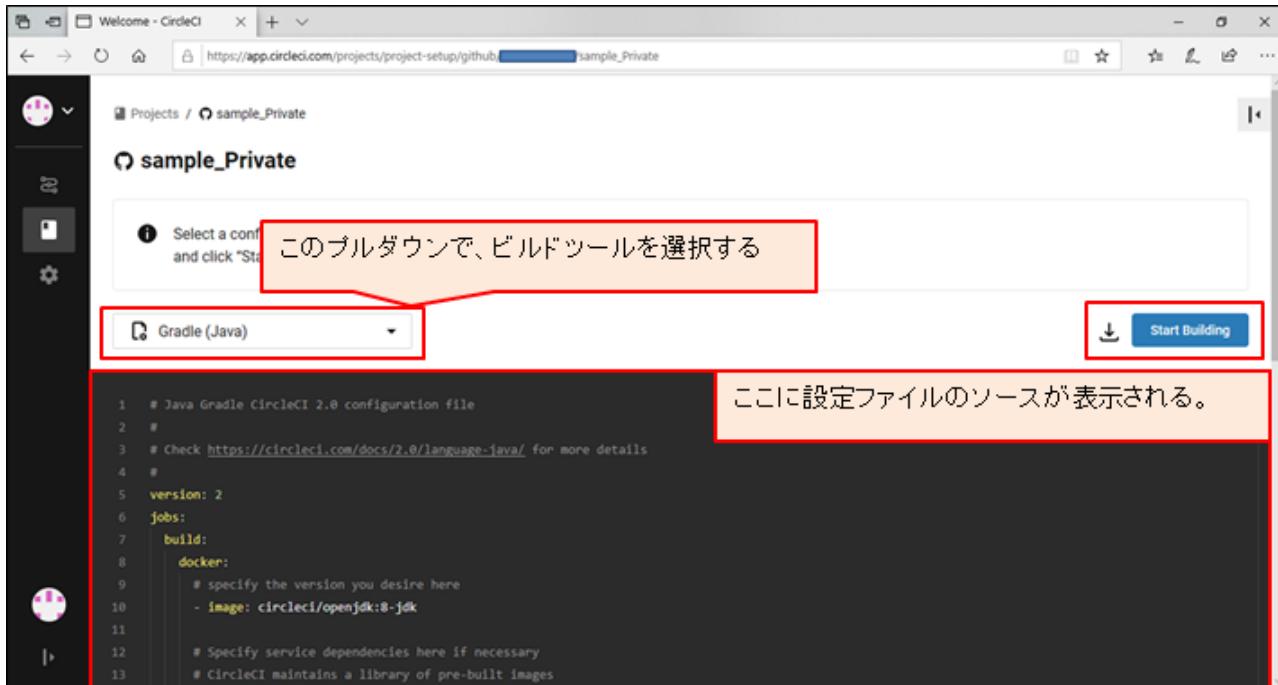


2 GitHub側の公開設定に関わらず、全リポジトリが表示されるので対象とするリポジトリを選択する。

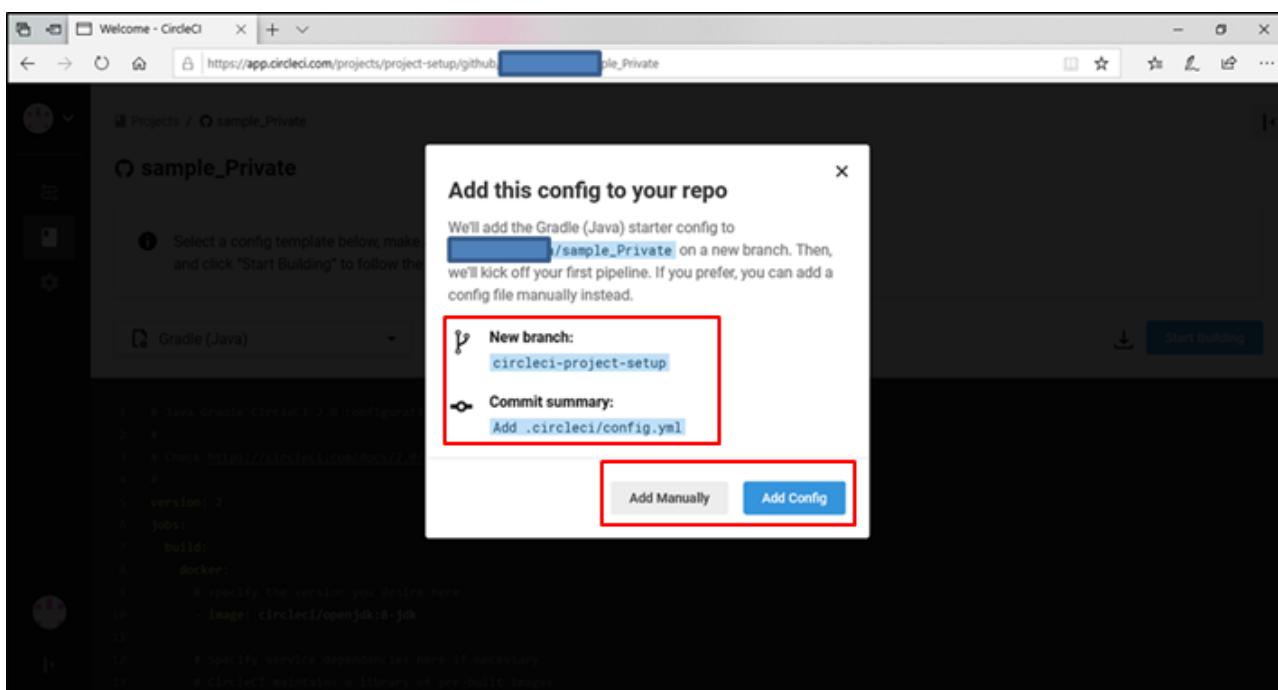


3 ビルドツールを選択して、「Start Building」でリポジトリ登録を開始する。

3. CI/CDツールの設定指針



4 「Start Building」を押すと下記のポップアップが表示されるので適宜「Add Manually」か「Add Config」を選択する。

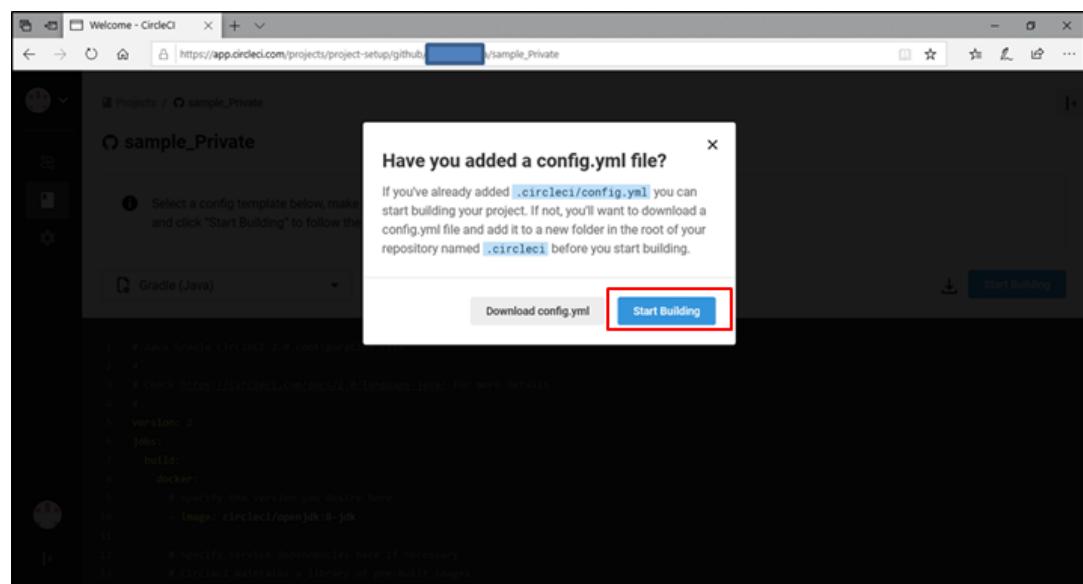


「Add Config」を選択した場合、図に表示されているようにブランチが作成され、.circleci配下に設定ファイル(config.yml)が作成される。
その後、設定ファイルに基づいて初回のパイプライン実行が行われる。
※設定ファイルの内容は前画面で表示されたデフォルトの内容になるので、初回のパイプライン実行は上手くいかないことが多いです。

Add

「Manually」を選択した場合、下記のポップアップが表示されるので適宜設定ファイルをcircleci配下に準備してから「Start Building」をクリックする。

※下図にあるように「Download config.yml」で設定ファイルをダウンロードすることも可能です。
※設定ファイルの内容は、前画面で表示された内容のものです。



4 初期パイプライン実行が行われると、下図のように実行が表示される。

Pipeline	Status	Workflow	Branch	Commit	Started	Duration
#1	FAILED	Build Error	master		7 minutes ago	7s

No more pipelines to load
If you have jobs that ran before pipelines were enabled for this project you can still access them by going to the [legacy jobs view](#).

5 登録済みのリポジトリは「Set Up Project」ボタンが「Unfollow Project」ボタンに変わっているので、これを押すとリポジトリ登録が解除される。

3. CI/CDツールの設定指針

The screenshot shows the CircleCI web interface. On the left, there's a sidebar with 'Pipelines', 'Projects' (which is selected and highlighted in grey), and 'Organization Settings'. The main area is titled 'Projects' with a sub-section 'REPO'. It lists two repositories: 'sample_Private' and 'sample_Public'. Next to 'sample_Private' is a red-bordered button labeled 'Unfollow Project'. At the bottom right of the list area is another button labeled 'Set Up Project'.

3.5.2. 環境変数の登録

「config.yml」や「build.gradle」に記載した環境変数はパイプライン起動時にCircleCI画面で設定した値に自動的に置き換えられます。



商用利用の場合、Githubはプライベートリポジトリで利用する想定ですが、パブリックリポジトリの場合はconfig.ymlの内容が不特定多数に公開されます。そのため、特に理由がないのであればリポジトリの設定に関わらず秘匿情報は環境変数としてWeb-UIで設定すること。

1 対象リポジトリのパイプライン画面より「Project Settings」を選択する。

The screenshot shows the CircleCI Pipeline dashboard for the 'check-c-sear' project. The sidebar on the left has a red box around the pipeline status filter icon (a small square with a checkmark). The main area is titled 'Pipelines' and shows a table of recent pipelines. The first pipeline listed is '#16' with a 'SUCCESS' status, run on the 'workflow' workflow for the 'issue_buildgradle' branch. Other pipelines listed are '#14', '#12', '#11', and '#10', all in 'CREATED' status. To the right of the table is a 'Project Settings' button, which is also highlighted with a red box. At the bottom of the page, a message says 'No more pipelines to load'.

2 Project Settings画面で「Environment Variables」を選択して、「Add Config Variable」を押すことで環境変数と変数値を設定できる。

Project Settings画面で「Environment Variables」を選択して、「Add Config Variable」を押すことで環境変数と変数値を設定できる。

Name	Value	Action
[REDACTED]	xxxxsearch	X
[REDACTED]	xxxxd8.be	X
[REDACTED]	xxxxses/	X
[REDACTED]	xxxxhot/	X
[REDACTED]	xxxxin	X

3 「Add Config

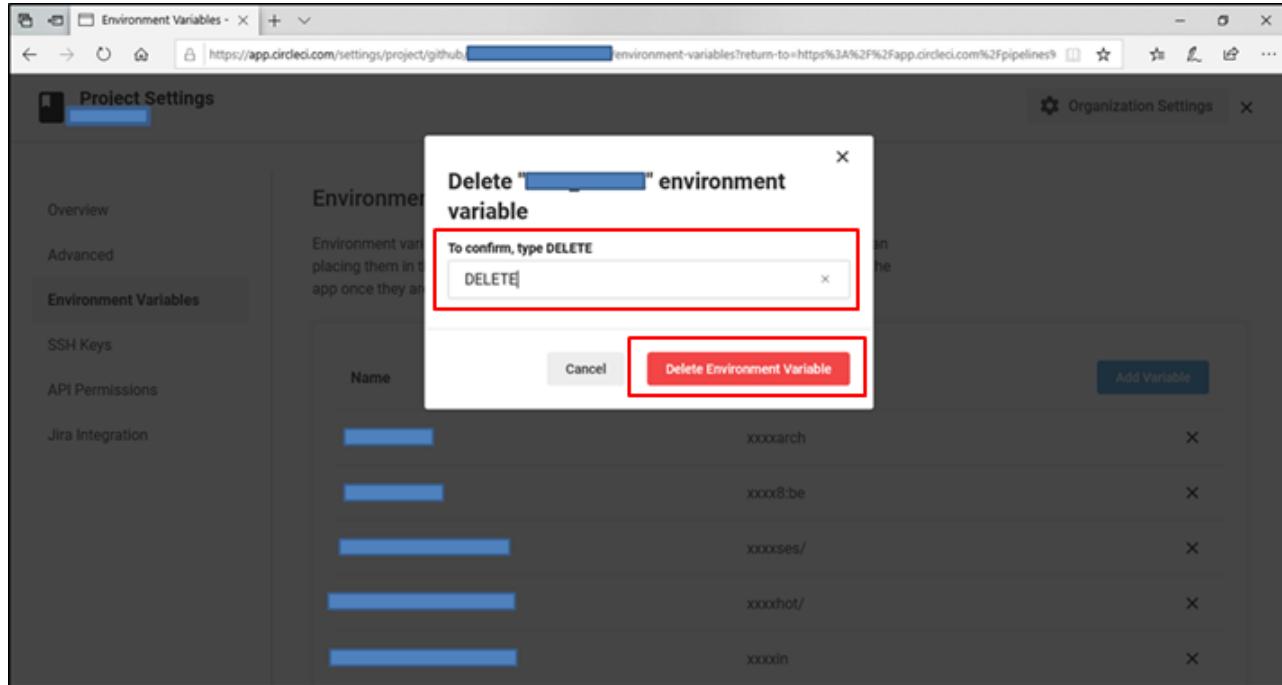
Variable」を押すと下記のポップアップが表示されるので、任意の環境変数と変数値を入力して「Add Environment Variable」を押す。

既存の環境変数を入力すると変数値を置き換えることができる。

4

「X」を押すと下記のポップアップが表示されるので、テキストボックスに「DELETE」と入力して「Delete

「Environment Variable」を押す。



自分で設定する環境変数以外にも、CircleCIデフォルトで準備されている環境変数が存在する。

それらについては、特に宣言することなく設定ファイル中で利用可能です。

詳細は下記URLを参照。



「CircleCIサイト>環境変数の使い方」

<https://circleci.com/docs/2.0/env-vars/#setting-an-environment-variable-in-a-step>

3.5.3. 実行結果の確認

CircleCI画面では実行されたパイプラインについて、下記の手順でワークフロー単位・Job単位での実行結果の確認を行います。

また、エラーが発生した場合には、適宜パイプラインの再実行を設定します。

- サイドバーの「Pipelines」を選択することで、下図のようにパイプライン単位で実行結果を確認することができます。

The screenshot shows the CircleCI Pipelines interface. At the top, there are three filter dropdowns: 'sample_Private' under 'Filters', 'All Pipelines', and 'All Branches'. A red box highlights the 'sample_Private' dropdown with the text 'リポジトリでフィルタ'. Another red box highlights the 'All Pipelines' dropdown with the text 'コミットユーザでフィルタ ※自分or全員'. A third red box highlights the 'All Branches' dropdown with the text 'ブランチでフィルタ'.

Pipeline	Status	Workflow	Branch	Commit	Started	Duration
#8	FAILED	workflow	issue_guid	Guideline_4	3 hours ago	22s
#7	SUCCESS	workflow	issue_guid	Guideline_3	3 hours ago	39s
#6	FAILED	workflow	issue_guid	Guideline_2	3 hours ago	19s
#5	SUCCESS	workflow	master	Merge pull request #2 from sample_Private/issue_guid	3 hours ago	18s
#4	SUCCESS	workflow	issue_guid	Guideline	3 hours ago	1m 33s
#3	Build Error	circleci-project-setup			1 day ago	8s
#2	NOT_RUN	workflow	circleci-project-setup	Add .circleci/config.yml	1 day ago	0s

A note at the bottom of the table says: '#3 No configuration was found in your project. Please refer to <https://circleci.com/docs/2.0/> to get started with your configuration.'



Pipelines画面の出力結果は、Filterとしてリポジトリ/コミットしたユーザ/ブランチでフィルタリングが可能です。

- ・パイプライン番号横の「▶」を押すと、ワークフローに属するJobごとの結果を確認することができます。

The screenshot shows the same CircleCI Pipelines interface as above, but with more detail. The first pipeline (#8) is expanded, showing its sub-jobs: 'build', 'job_1', 'job_2', and 'job_3', all of which are listed as 'Blocked'. The second pipeline (#7) is also expanded, showing its sub-jobs: 'build', 'job_1', 'job_2', and 'job_3', all of which are listed as 'Success'.

- ・パイプライン右の「...」を押すと、各パイプラインの設定ファイル (config.yml) の内容を確認することができます。

3. CI/CDツールの設定指針

Pipeline	Status	Workflow	Branch	Commit	Started	Duration
#8	FAILED	workflow	issue_guid	Guideline_4	3 hours ago	22s
	Failed	build				19s
	Blocked	job_1				
	Blocked	job_2				
	Blocked	job_3				
#7	SUCCESS	workflow	issue_guid	Guideline_3	3 hours ago	39s
	Success	build				5s
	Success	job_1				3s
	Success	job_2				9s
	Success	job_3				
#6	FAILED	workflow	issue_guid	Guideline_2	3 hours ago	19s

設定ファイルの見方は、「Source」と「Compiled」があります。



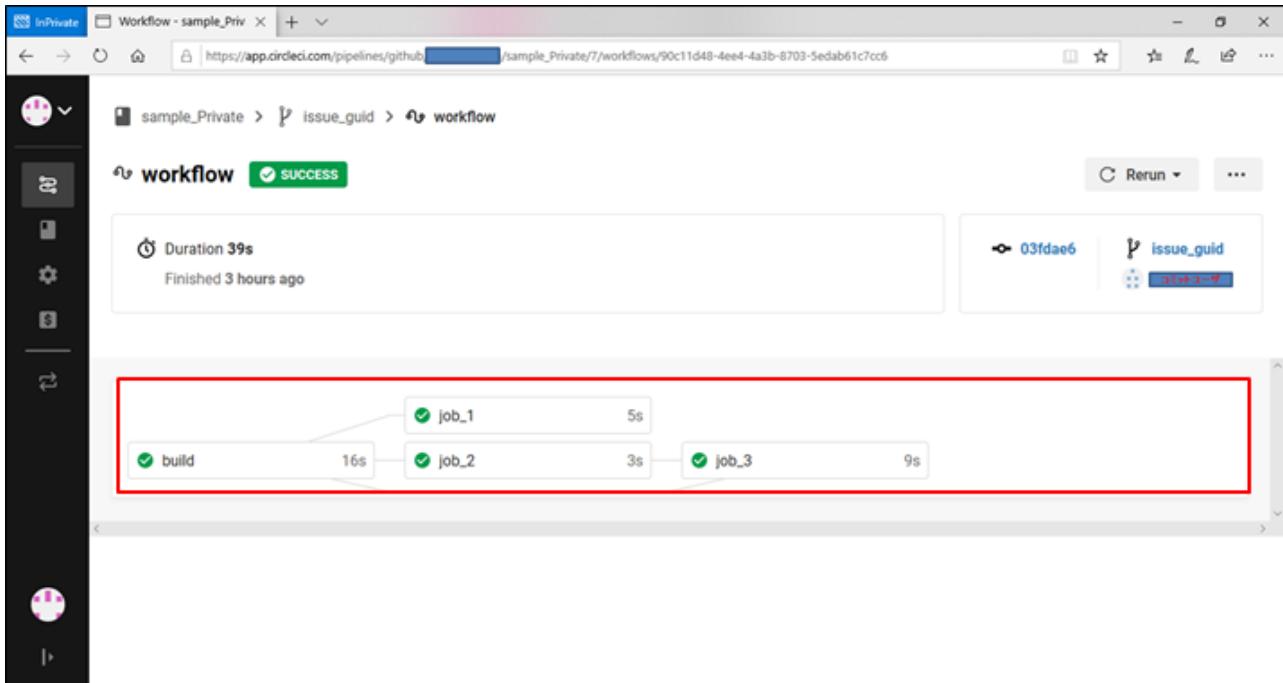
```
# Java Gradle CI circleCI 2.0 configuration file
#
# Check https://circleci.com/docs/2.0/language-java/ for more details
#
##CircleCIのバージョン指定
version: 2.1

##以下バイブルインの記述
jobs:
  # "./gradlew build"コマンドによるビルド・静的検証・Junitテストを実施する
  build:
    docker:
      # specify the version you desire here
      - image: circleci/openjdk:latest

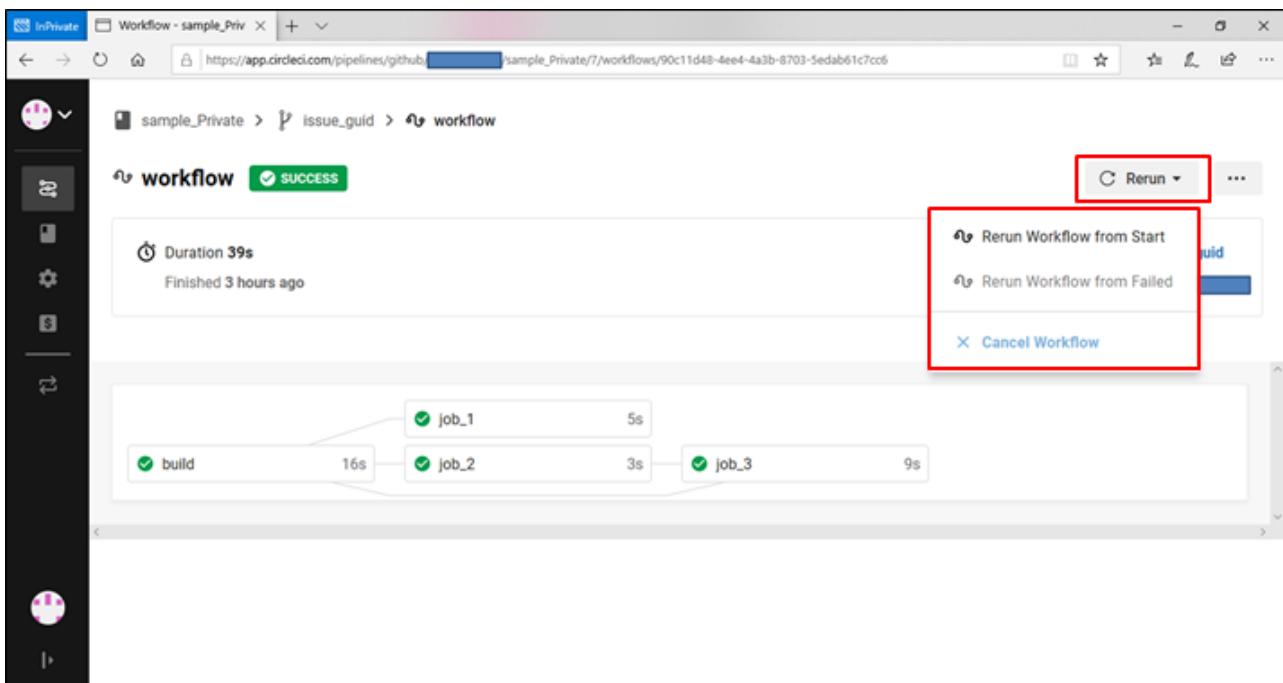
    # Specify service dependencies here if necessary
    # CircleCI maintains a library of pre-built images
    # documented at https://circleci.com/docs/2.0/circleci-images/
    # - image: circleci/postgres:9.4

    working_directory: ~/repo
    environment:
```

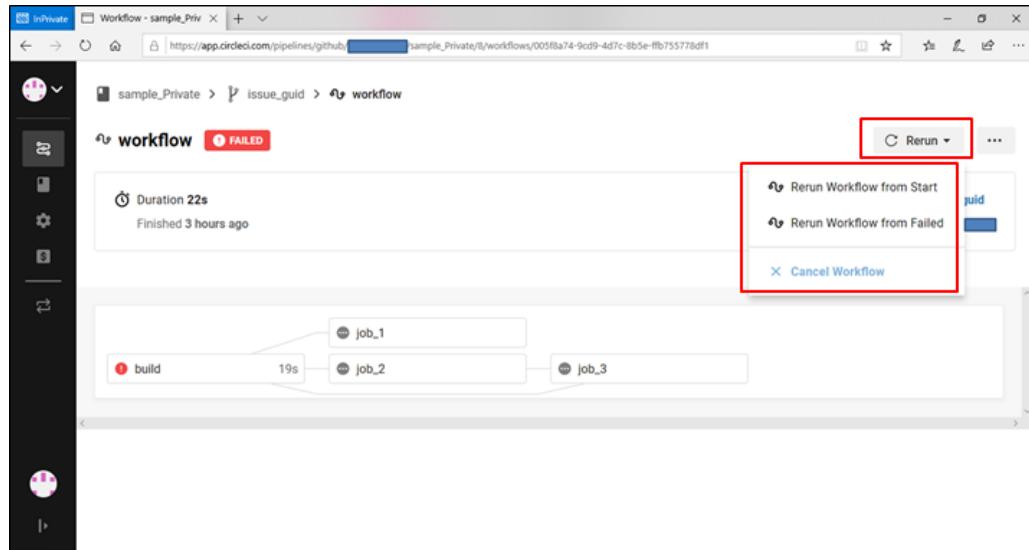
- Pipelines画面でパイプラインの「STATUS」または「WORKFLOW」を押すと下図のようにワークフロー単位での実行結果を確認できます。



ワークフロー確認画面で「Return」を押すと下図のように再実行方法や実行停止を選択することができます。

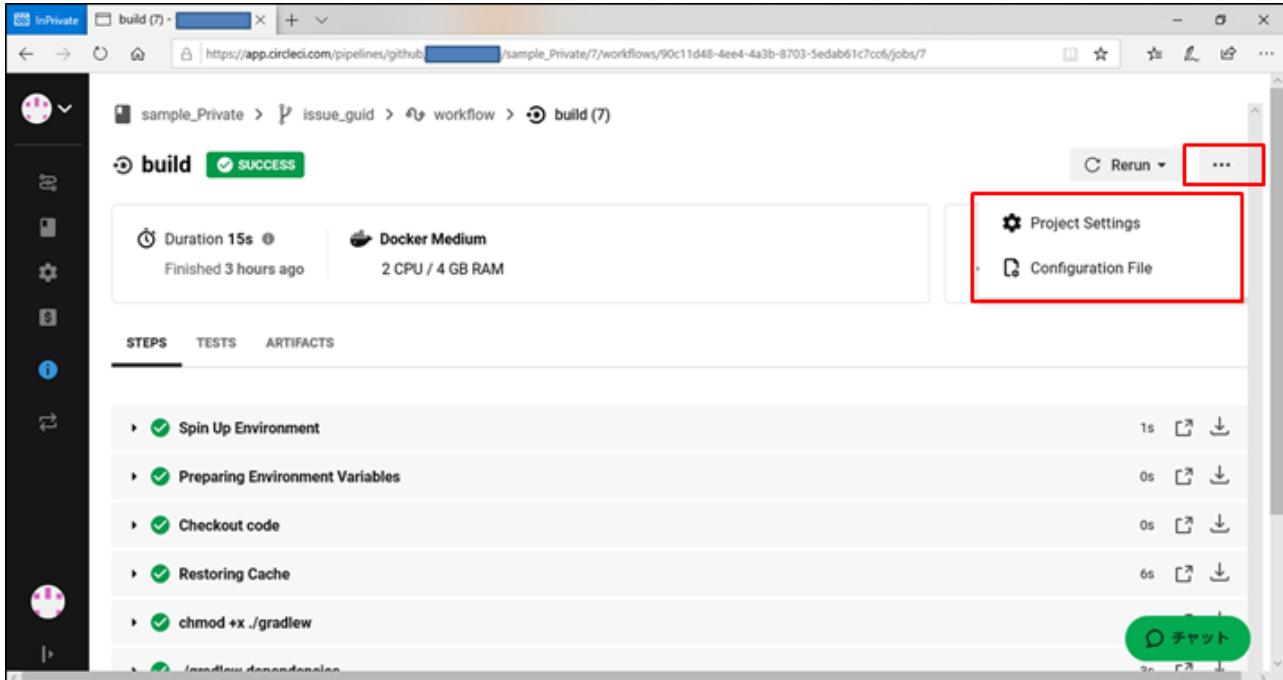


パイプラインが正常終了している場合は、「Rerun Workflow from Start」のみ選択することが可能ですが、異常終了している場合は「Rerun Workflow from Failed」も選択可能です。



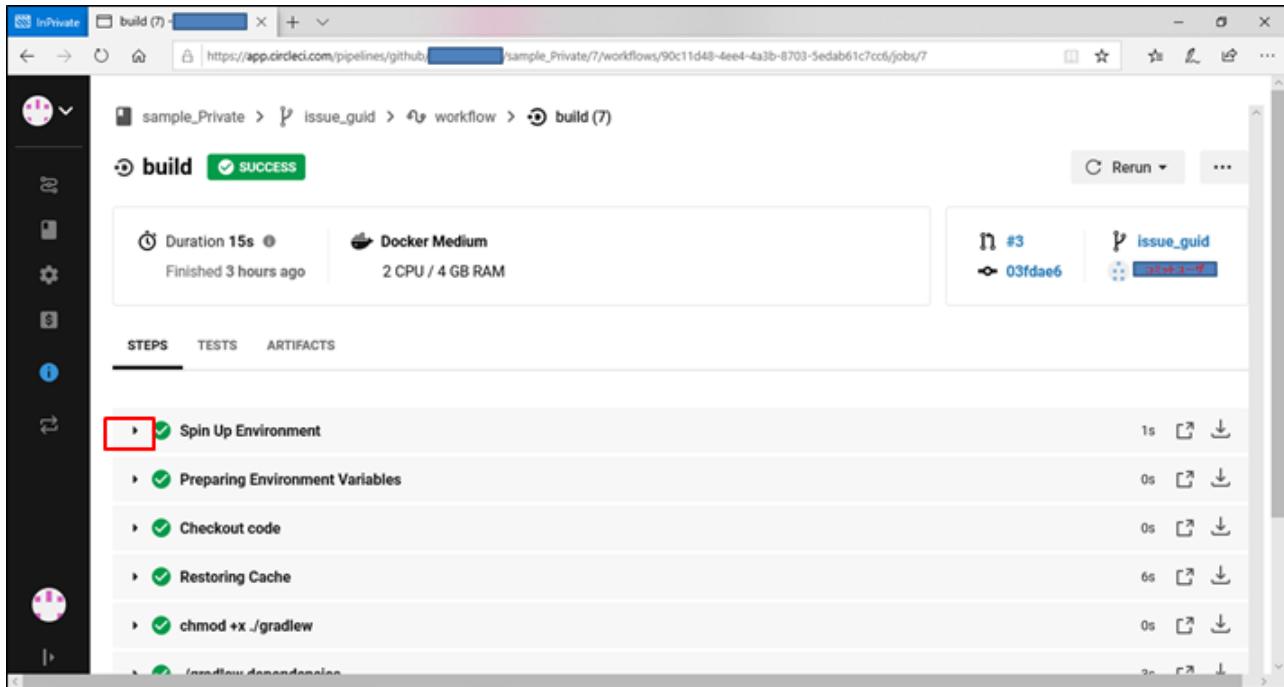
- ワークフロー確認画面で「・・・」を押すと下図のように設定ファイル確認とProjectSettingを行うことができます。

※画面・内容については、既に説明済みのため省略



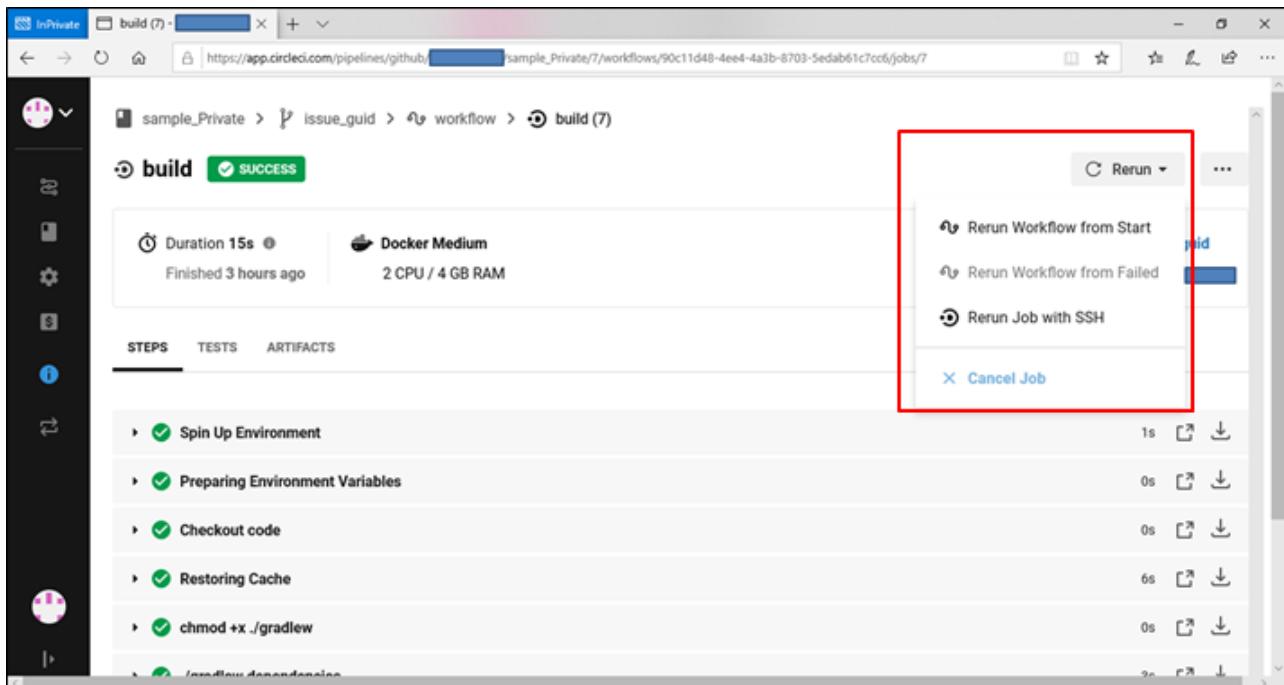
- ワークフロー確認画面で任意のJobを選択すると下図のようにJob単位での実行結果を確認することができます。

また、各ステップは「▶」を押すことで展開可能です。



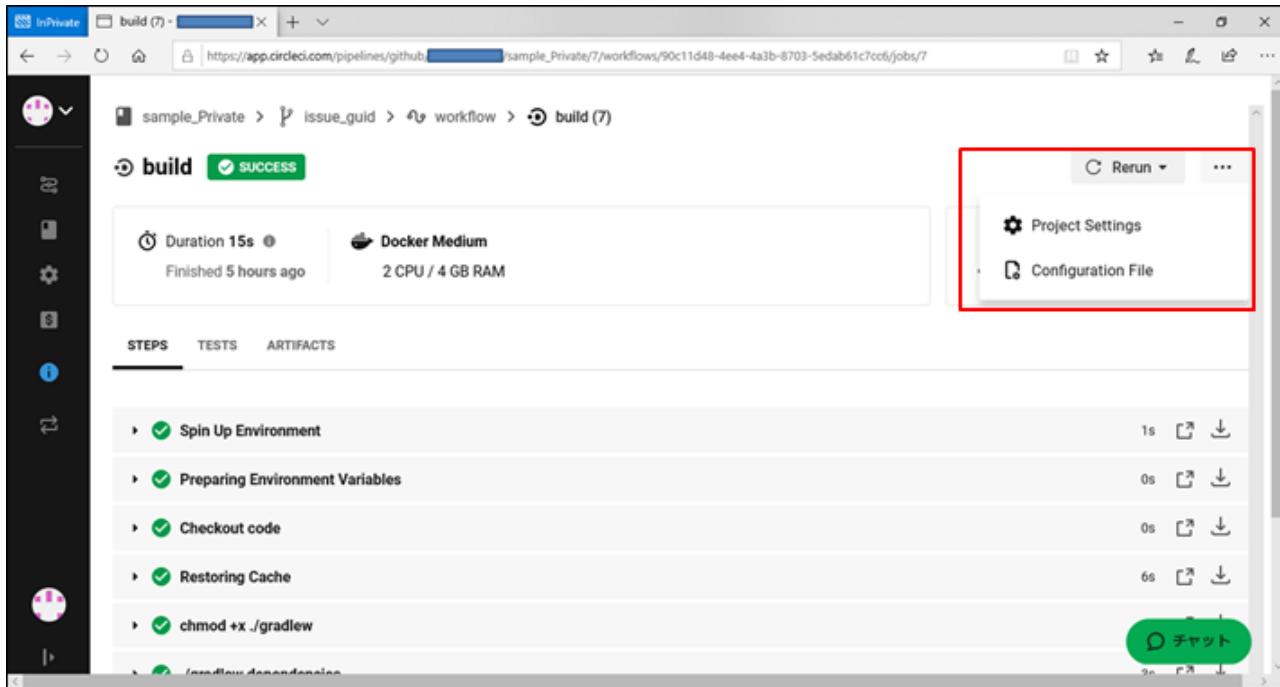
「Spin Up Environment」、「Preparing Environment Variables」については、設定ファイルの記述に関係なく実行されるデフォルトのStepです。
それぞれ実行環境の構築と環境変数の読み込みを行うStepです。

- Job確認画面で「Return」を押すと下図のようにワークフロー確認画面と同様に再実行方法や実行停止を選択することができます。



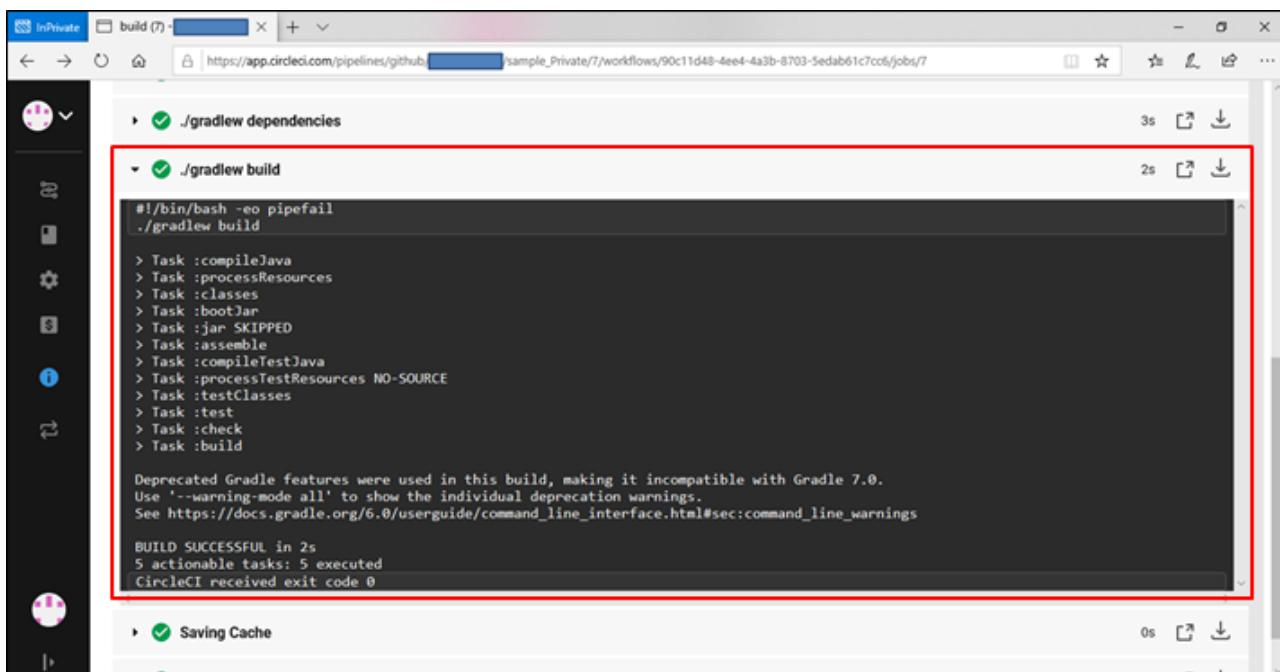
- Job確認画面で「・・・」を押すと下図のようにワークフロー確認画面と同様に設定ファイル確認とProjectSettingを行うことができます。

3. CI/CDツールの設定指針



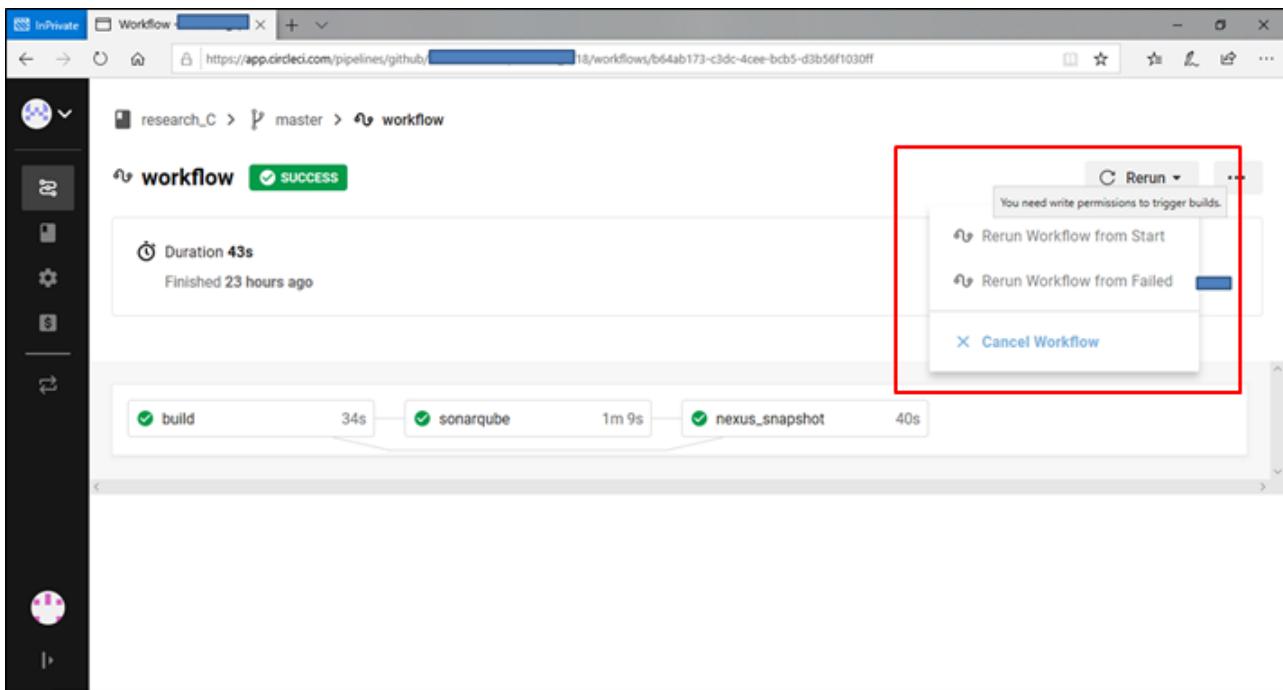
The screenshot shows a browser window displaying a CircleCI pipeline. The pipeline name is 'sample_Private' and the workflow is 'issue_guid'. The specific build shown is labeled 'build (7)' and has a status of 'SUCCESS'. The build duration was 15 seconds, completed 5 hours ago, and used a Docker Medium environment with 2 CPU / 4 GB RAM. The 'STEPS' tab is selected, showing a list of tasks: 'Spin Up Environment', 'Preparing Environment Variables', 'Checkout code', 'Restoring Cache', and 'chmod +x ./gradlew'. The last task, './gradlew dependencies', is currently being executed. A red box highlights the top right corner of the interface, which contains options for 'Rerun', 'Project Settings', and 'Configuration File'.

- Job確認画面で任意のJobを選択することで、下図のようにJobの内容を確認することができます。

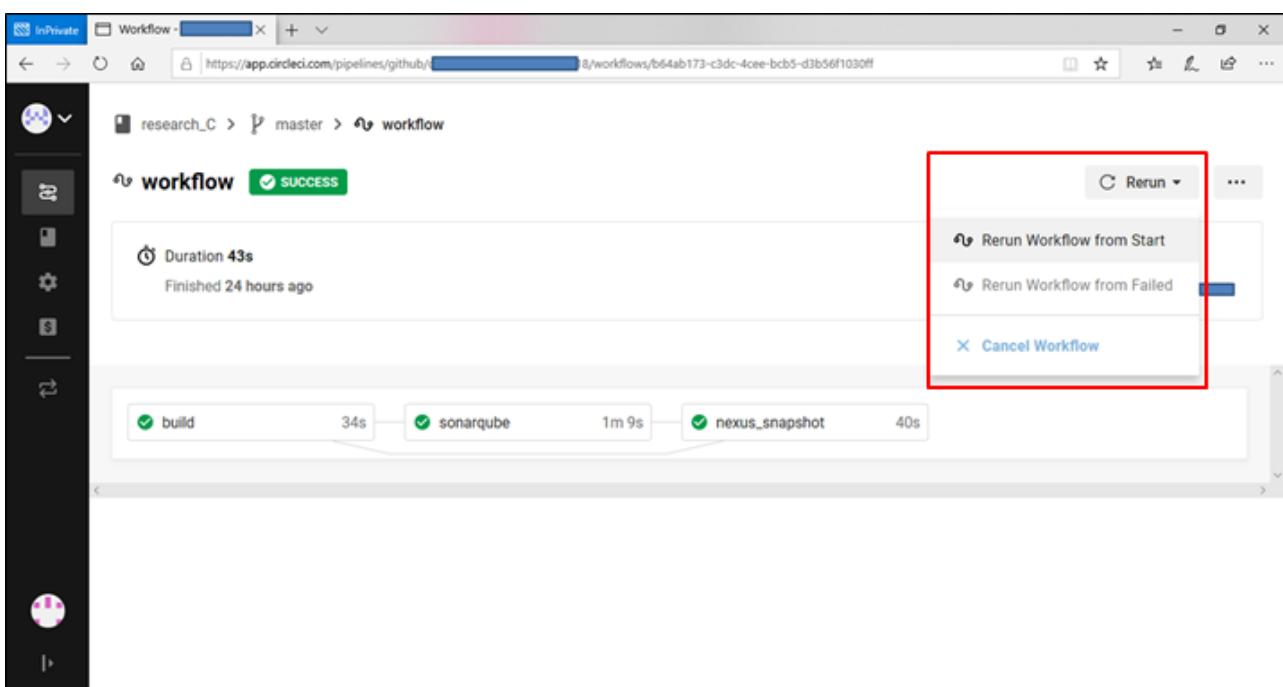


The screenshot shows the same CircleCI pipeline interface as above, but focusing on the details of a specific job step. The step in question is './gradlew build', which is currently executing. The command run is '#!/bin/bash -eo pipefail ./gradlew build'. The output of the command shows the Gradle build process, including tasks like ':compileJava', ':processResources', and ':jar'. A warning message at the bottom states: 'Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0. Use '-warning-mode all' to show the individual deprecation warnings. See https://docs.gradle.org/6.0/userguide/command_line_interface.html#sec:command_line_warnings'. The build was successful, with 5 actionable tasks executed and a CircleCI exit code of 0. A red box highlights the expanded view of the './gradlew build' step.

- Organization所有のリポジトリについて、リポジトリへのアクセス権限によって、パイプラインの操作可能範囲が異なります。
- リポジトリへのアクセス権限が「参照」の場合、下図に示すようにパイプラインの再実施等のRerun操作は不可能です。



- リポジトリへのアクセス権限が「書き込み」以上の場合、Rerun操作が可能です。



このガイドラインでは、2020年4月21日現在の最新画面レイアウトに基づいて説明しましたが、下図のように旧画面レイアウトでCircleCIサイトを利用することも可能です。
見え方は若干異なっていますが、確認できる内容は前述のものと同じなので直感的に使いやすい方を使えばよいです。

3. CI/CDツールの設定指針

The screenshot shows a browser window for CircleCI. The URL is https://app.circleci.com/pipelines/github/sam.../sample_Private/7/workflows/90c11d48-4ee4-4a3b-8703-5edab61c7cc6/jobs/7. The main content displays a build summary for 'build (7)'. The status is 'SUCCEED' with a duration of 15s. It was finished 5 hours ago using a 'Docker Medium' configuration with 2 CPU / 4 GB RAM. A specific job, '#3 issue_guid #03fdæ6', is highlighted. The 'STEPS' tab is selected, showing a list of steps: 'Spin Up Environment' (1s), 'Preparing Environment Variables' (0s), 'Checkout code' (0s), 'Restoring Cache' (6s), and 'chmod +x ./gradlew' (1s). A green 'チャット' (Chat) button is visible at the bottom right.

The screenshot shows a browser window for CircleCI. The URL is https://circleci.com/gh/sam.../sample_Private. The left sidebar includes 'JOBS', 'WORKFLOWS', 'INSIGHTS', 'PROJECTS', 'TEAM', and 'SETTINGS'. The main area shows a list of jobs under 'sample_Private'. The table has columns for 'Status', 'Job ID', 'Workflow', 'Last Run', and 'Duration'. The jobs listed are: 'issue_guid #12' (SUCCESS, Guideline_3, workflow, 22 min ago, 00:03), 'issue_guid #11' (FAILED, Guideline_4, workflow, 5hr ago, 00:18), 'issue_guid #10' (SUCCESS, Guideline_3, workflow, 5hr ago, 00:09), 'issue_guid #9' (SUCCESS, Guideline_3, workflow, 5hr ago, 00:05), 'issue_guid #8' (SUCCESS, Guideline_3, workflow, 5hr ago, 00:03), 'issue_guid #7' (SUCCESS, Guideline_3, workflow, 5hr ago, 00:15), and 'issue_guid #6' (FAILED, Guideline_2, workflow, 5hr ago, 00:07). A green 'チャット' (Chat) button is visible at the bottom right.

Workflows > sample_Private > sample_Private

Showing 1-8

	issue_guid / workflow	5 hrs ago	00:21
● FAILED	Guideline_4	#3	Blc572b
● SUCCEEDED	Guideline_3	#3	03fda6
● FAILED	Guideline_2	#3	1016dc7
● SUCCEEDED	master / workflow Merge pull request #2 from sample/Private/issue_guid	#3	18f2fbf

チャット

3.5.4. 実行結果の発報

パイプライン実行結果をPJのチャットツール（このガイドラインではSlack）に連携する。

1 Slackの「App」よりCircleCIアプリを追加して、「セットアップの手順」に従ってCircleCIに設定する。

CircleCI | Slack App ディレクトリ

App ディレクトリ 検索

検索 管理 ビルド

CircleCI さんによって 2020年4月6日に追加されました

無効にする 削除する

セットアップの手順

CircleCI インテグレーションを追加するために必要なステップは次の通りです。

インテグレーションの設定

チャンネルへの投稿

CircleCI からの通知はここに投稿されます。 #ガイドライン

実行結果は、下記の方式でCircleCI及びGitHubへのリンク付きで連携される。
 正常終了した場合は「どのリポジトリ」の「どのブランチ」が「誰が作成したパイプライン」を正常に通過したのか、「コミットしたGitHubユーザ」は誰かを通知する。
 異常終了した場合は「どのリポジトリ」の「どのブランチ」が「誰が作成したパイプライン」の「どのJob」で異常終了したのか、「コミットしたGitHubユーザ」は誰かを通知する。



3.5.5. Organizationアカウントのrepositoryを参照するための設定

1 Organizationのリポジトリを登録する場合、自身のアイコンをクリックして「User Setting」画面より設定を行う。

2 「User Setting」画面で「Account Integrations」→GitHubの「Check

permissions」を選択して、Github画面を開く。

The screenshot shows the CircleCI User Settings page under the Account Integrations tab. On the left sidebar, there are several options: Notifications, Privacy & Security, Personal API Tokens, Organization Plans, and Beta Program. The main area is titled "Account Integrations". It contains a "Refresh Permissions" section with a "Refresh Permissions" button. Below that is a "GitHub" section with the subtext "Build and deploy your GitHub repositories." and a status message "Connected to [REDACTED] ✓". A red box highlights the "Check permissions" link next to a note about missing a GitHub organization. At the bottom is a "Bitbucket" section with the subtext "Build and deploy your Bitbucket repositories." and a "Connect" button.

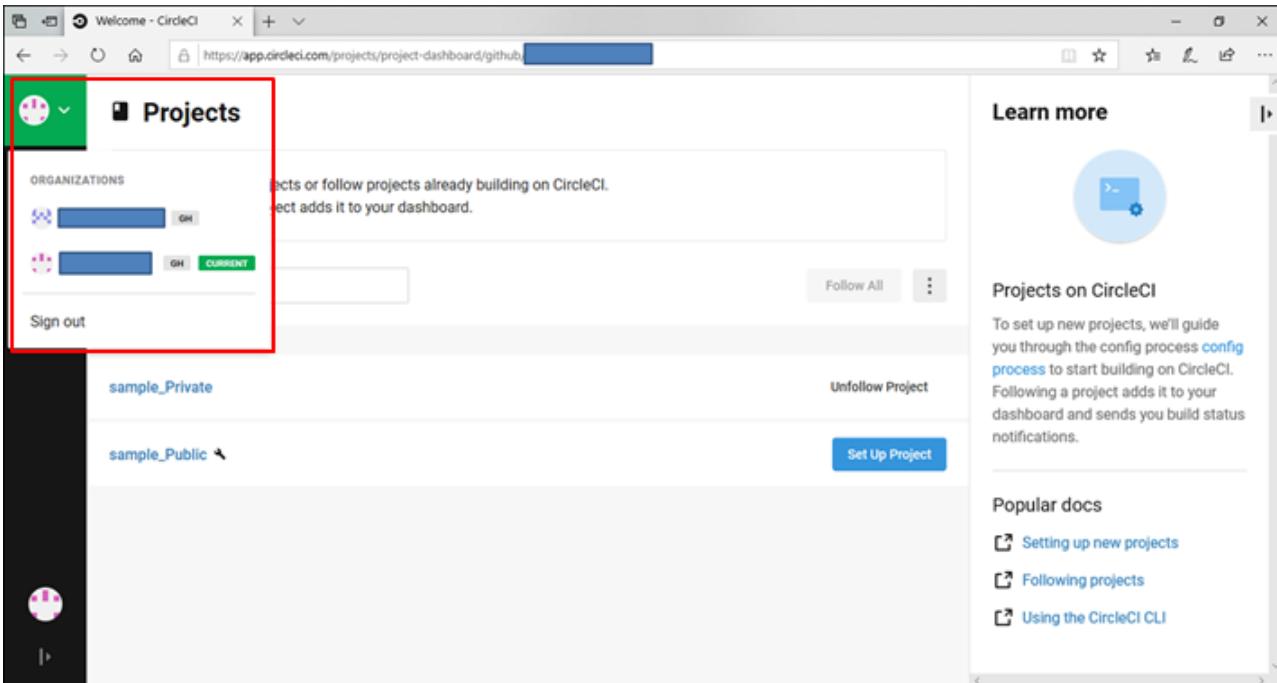
3 画面下の「Organization access」

access」に自信が所属しているOrganizationが表示されるので、その内容を確認して「Grant」ボタンを押下する。

この操作でOrganization所有のリポジトリを参照できるようになる。

The screenshot shows the GitHub Organization settings page for "Connection with CircleCI". The left sidebar lists "Personal settings" like Profile, Account, Security, etc., and "Applications" which includes "CircleCI". The main content area shows the CircleCI application details. It has sections for "Permissions" (with "Revoke access" and "Grant" buttons) and "Organization access" (with a "Grant" button). A red box highlights the "Grant" button in the "Organization access" section, with a note below it stating "ここに「Grant」ボタンが表示されるが、Grant押下後は消えてしまう".

3. CI/CDツールの設定指針



The screenshot shows the CircleCI dashboard with the 'Projects' section. On the left, there's a sidebar with a user icon and a dropdown menu. Below it, a list of organizations is shown, with one organization named 'sample_Public' highlighted with a green 'CURRENT' badge. The main area displays two GitHub projects: 'sample_Private' and 'sample_Public'. For 'sample_Public', there are buttons for 'Unfollow Project' and 'Set Up Project'. To the right, there's a 'Learn more' section about setting up new projects, followed by a 'Popular docs' section with links to 'Setting up new projects', 'Following projects', and 'Using the CircleCI CLI'.

Organization所有のリポジトリについては、Organization内での自アカウントの権限によって操作が制限される。

- リポジトリの参照

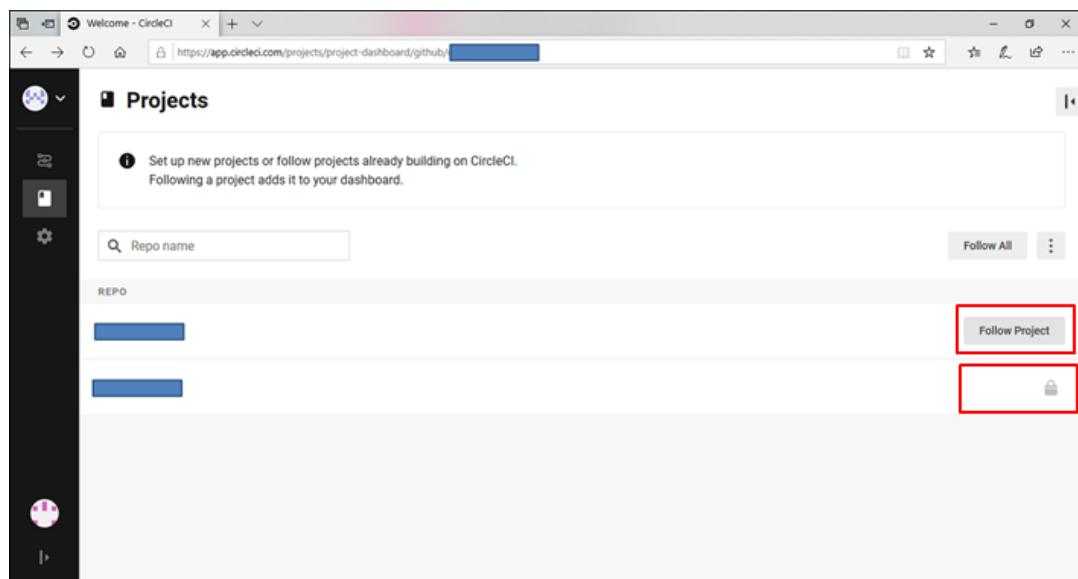
管理者権限のあるリポジトリについては、個人アカウント同様に「リポジトリの登録」「登録解除」を行うことができる。

※ただし、リポジトリ解除できるのは自身が登録したリポジトリに限られる。

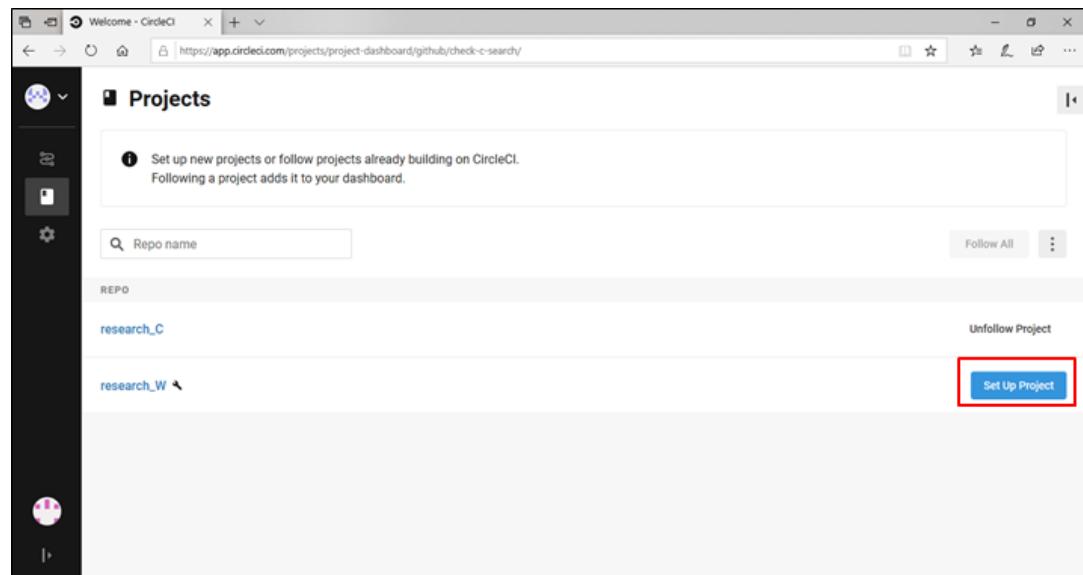
その他のリポジトリについては、「リポジトリの参照」「参照解除」を行うことができる。

※ただし、管理者によって登録が解除されているリポジトリについては参照不可（鍵のマークがつく）

また、GitHub上での権限設定がCircleCIに反映されるまで若干のタイムラグが存在する。

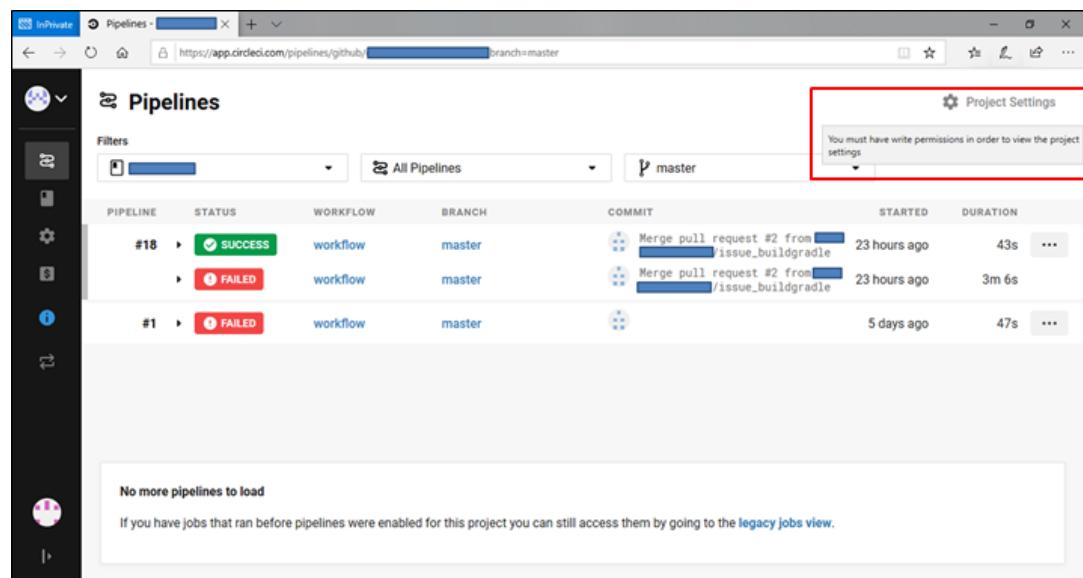


This screenshot shows the same CircleCI dashboard as the previous one, but with a different focus. A red box highlights the 'Follow Project' button and the lock icon next to a repository entry in the list. The rest of the interface is similar to the first screenshot, showing the sidebar with organizations and the main area with GitHub projects.



- ・環境変数の設定

リポジトリへのアクセス権限が「参照」の場合、下図に示すように「ProjectSettings」が非活性になるため環境変数を含め、リポジトリの設定を変更することは不可能です。



リポジトリへのアクセス権限が「書き込み」以上の場合、下図に示すように「ProjectSettings」が活性になるため環境変数を含め、リポジトリの設定を変更することができます。

3. CI/CDツールの設定指針

The screenshot shows the CircleCI Pipelines interface. At the top right, there is a 'Project Settings' button with a gear icon, which is highlighted with a red box. The main area displays a table of pipelines:

PIPELINE	STATUS	WORKFLOW	BRANCH	COMMIT	STARTED	DURATION
#18	SUCCESS	workflow	master	Merge pull request #2 from [redacted]/issue_buildgradle	24 hours ago	43s
	FAILED	workflow	master	Merge pull request #2 from [redacted]/issue_buildgradle	24 hours ago	3m 6s
#1	FAILED	workflow	master	[redacted]	5 days ago	47s

Below the table, a message states: "No more pipelines to load". A note below it says: "If you have jobs that ran before pipelines were enabled for this project you can still access them by going to the [legacy jobs view](#)".

4. ソースコード/開発チーム管理指針

4.1. 概要

- このガイドラインでは、バージョン管理ツールとしてGitHubを利用します。
<Githubの公式サイト>
<https://github.com/>
- GithubではOrganizationアカウントを作成して、ソースコードの管理及び開発チームメンバの管理を行います。
また、CI/CDツール（今回はCircleCI）と連携することでPR時に対象のブランチがCI/CDを通過しているかどうかの確認を併せて行います。

4.2. チーム管理

このガイドラインでは、下記のように開発チームようにOrganizationアカウントを一つ作成して個々のメンバーを招待する形を取ります。

[team_img] | fig_51.png

※Organizationアカウント：リポジトリを作成してPJのソースコードを管理します
オーナ(個別アカウント)：権限に応じてOrganizationアカウントのソースコードを編集します。
メンバー①(個人アカウント)：権限に応じてOrganizationアカウントのソースコードを編集します。
メンバー②(個人アカウント)：権限に応じてOrganizationアカウントのソースコードを編集します。

4.2.1. 開発チーム用アカウント(Organizationアカウント)の作成

GitHubには、下記2種のアカウントが存在します。

表 12. Githubアカウント

アカウント	作成可能なリポジトリ	他ユーザによるリポジトリ操作の制限
個別アカウント	<ul style="list-style-type: none"> パブリック プライベート 	<p>ownerと同様の権限を持った共同編集者を設定可能</p> <ul style="list-style-type: none"> リポジトリの公開設定に関わらずアクセスが可能 共同編集者に設定したユーザの詳細な操作権限を指定することはできない 共同編集者はリポジトリ毎に設定する必要がある 無料で追加できるのは3名まで ※4人目以降はGithub Pro(\$ 4/月)プランに変更する必要がある 
Organizationアカウント	<ul style="list-style-type: none"> パブリック プライベート <p>Github Freeプランでは不可</p>	<p>Organizationアカウントにメンバーを招待することで、メンバー毎に各リポジトリへのアクセス権限を付与することが可能</p> <ul style="list-style-type: none"> ユーザ毎に詳細な操作権限を指定することが可能 メンバーを招待後は、Github画面でGUI的に各リポジトリへの操作権限を指定できる Github Freeプランでも招待できるメンバーに制限がない 

アカウントの詳細は、下記の公式ドキュメントを参照

- <Github公式ドキュメント>
<https://help.github.com/ja/github/setting-up-and-managing-your-github-user-account>

PJでは開発モジュール単位や機能単位にリポジトリを切って開発を行うため、様々なリポジトリが存在します。
また、これらのリポジトリ毎にメンバー(所属するTm)の操作権限を詳細に設定する必要があります。
このような運用を行う場合、開発チーム用アカウントとしては、個別アカウントではなくOrganizationアカウ

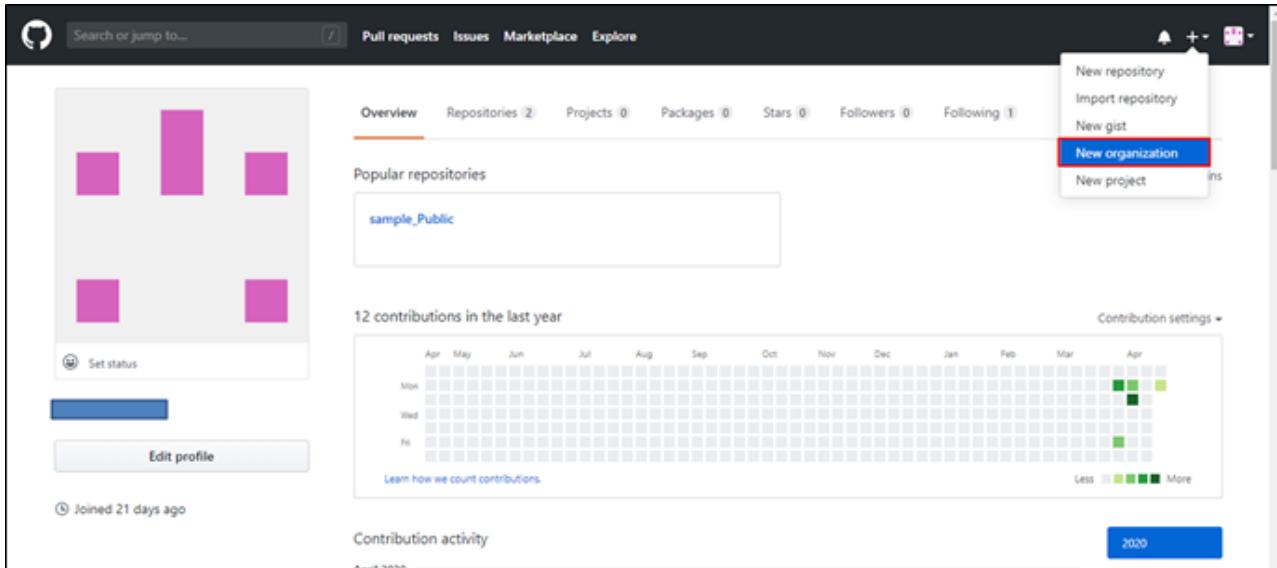
ントが推奨されます。

そのため、開発チーム用アカウントとしてOrganizationアカウントを採用します。

【手順】

<1> Githubに個人アカウントでサインインして、「New organization」を選択する。

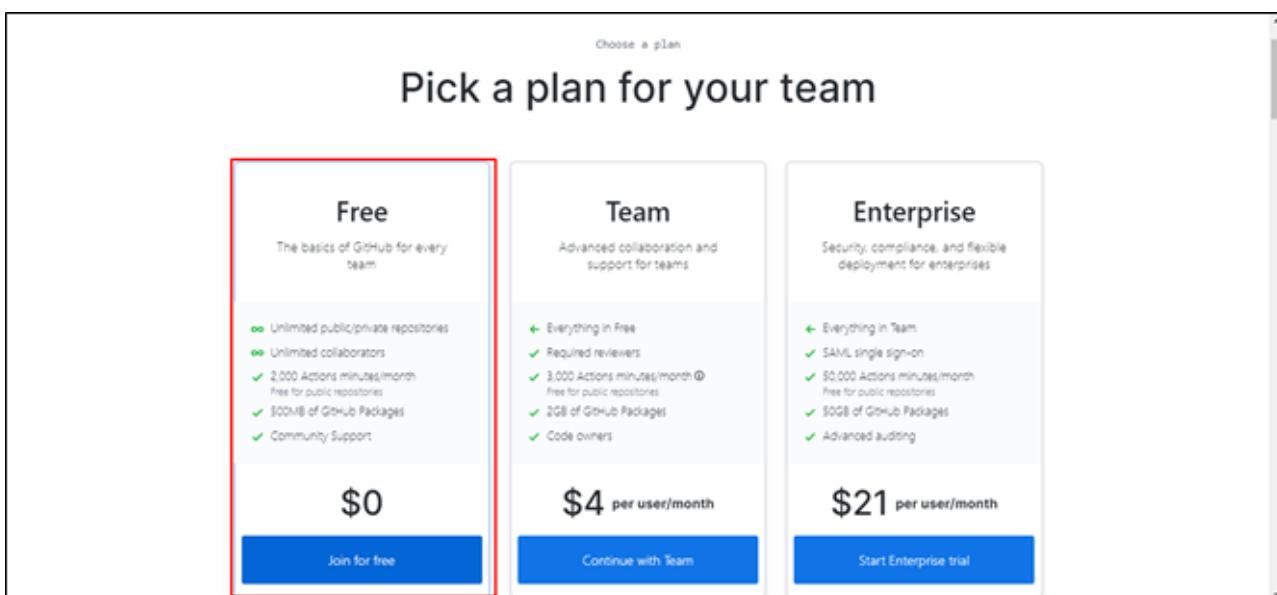
※この時使用したアカウントがOrganizationアカウントのOwnerとなります。



<2> 利用するプランを選択する。

※この例では、Freeプランを選択していますが、

商用利用の場合は基本的にプライベートリポジトリ利用することになるので適宜プランを選択してください。



<3> Organizationアカウントの情報を設定します。

Tell us about your organization

Set up your team

Organization account name *

 ✓

This will be the name of your account on GitHub.
Your URL will be: <https://github.com/sample-sample2>.

Contact email *

 ✓

This organization belongs to: *

My personal account
I.e.,

A business or institution
For example: GitHub, Inc., Example Institute, American Red Cross

[Next](#)

By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related

「This organization belong to」で「A business or institution」を選択した場合は下図のように組織名を指定します。



Tell us about your organization

Set up your team

Organization account name *

 ✓

This will be the name of your account on GitHub.
Your URL will be: <https://github.com/sample-sample2>.

Contact email *

 ✓

This organization belongs to: *

My personal account
I.e.,

A business or institution
For example: GitHub, Inc., Example Institute, American Red Cross

Name of business or institution this organization belongs to *

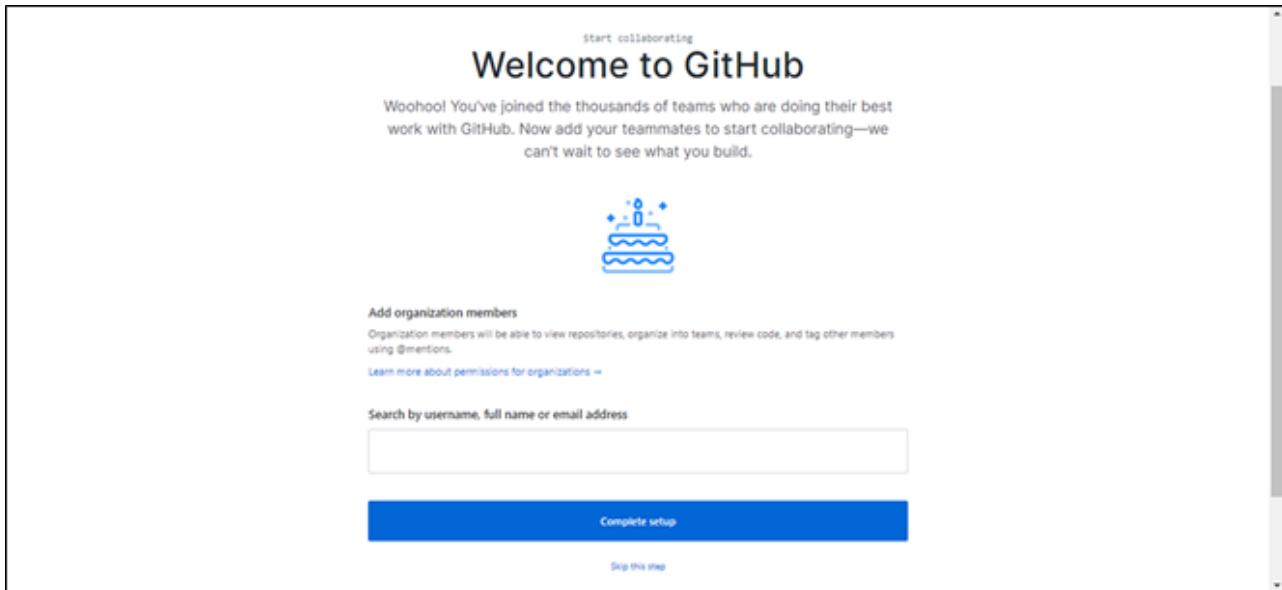
 *

This business or institution --- not taronishizawa (your personal account) --- will control this organization account.

[Next](#)

By creating an account, you agree to the [Corporate Terms of Service](#) on behalf of sample. For more

<4> 最後に自分以外に招待するメンバーを指定して完了です。
※ここでメンバーチャンクはskip可能です。



4.2.2. チーム管理

- Organizationアカウントにメンバーを招待する場合、下図のように予め「Settings>Member privilegeas」にて新規メンバーの操作権限を設定しておきます。

4. ソースコード/開発チーム管理指針

The screenshot shows the GitHub organization settings page for 'sample-sample2'. The left sidebar lists various settings categories. The 'Member privileges' section is currently selected, showing 'Member repository permissions'. Under 'Base permissions', the 'Read' option is selected. A red arrow points from this selection to a modal window titled 'Organization member permissions' which is also highlighted with a red border. The modal lists three options: 'None', 'Read', and 'Write'. 'Read' is checked and highlighted with a blue background. The 'Read' description states: 'Members will be able to clone and pull all repositories.' The 'Write' and 'Admin' options are also listed with their respective descriptions. Below the 'Member repository permissions' section, there are sections for 'Repository creation', 'Repository forking', 'Actions', and 'Admin repository permissions'. Each of these sections has a 'Save' button at the bottom.

Member repository permissions

Base permissions

Base permissions to the organization's repositories apply to all members and excludes outside collaborators. Since organization members can have permissions from multiple sources, members and collaborators who have been granted a higher level of access than the base permissions will retain their higher permission privileges.

Read →

Repository creation

Members will be able to create only selected repository types. Outside collaborators can never create repositories.

Public Members will be able to create public repositories, visible to anyone. [Why is this option disabled?](#)

Private Members will be able to create private repositories, visible to organization members with permission.

Save

Repository forking

Allow forking of private repositories If enabled, forking is allowed on private and public repositories. If disabled, forking is only allowed on public repositories. This setting is also configurable per-repository.

Save

Actions

Automate all your software workflows. Build, test, and deploy your code right from GitHub.

Enable local & third party Actions for this organization This allows all repositories to execute any Action, whether the code for the Action exists within the same repository, same organization, or a repository owned by a third party.

Enable local Actions only for this organization This allows all repositories to execute any Action as long as the code for the Action exists within the same repository.

Disable Actions for the organization This disallows any Action from running on any repository in the organization.

Save

Admin repository permissions

Repository visibility change

Allow members to change repository visibilities for this organization If enabled, members with admin permissions for the repository will be able to change its visibility. If disabled, only organization owners can change repository visibilities.

Save

Repository deletion and transfer

Allow members to delete or transfer repositories for this organization If enabled, members with admin permissions for the repository will be able to delete or transfer public and private repositories. If disabled, only organization owners can delete or transfer repositories.

Save

Issue deletion Beta

Allow members to delete issues for this organization If enabled, members with admin permissions for the repository will be able to delete issues.

Save

Member team permissions

Team creation rules

Allow members to create teams If enabled, any member of the organization will be able to create new teams. If disabled, only organization owners can create new teams.

Save

© 2020 GitHub, Inc. [Terms](#) [Privacy](#) [Security](#) [Status](#) [Help](#) [Contact GitHub](#) [Pricing](#) [API](#) [Training](#) [Blog](#) [About](#)

- また、メンバー毎にリポジトリ別の操作権限を作成するために、「Teams>New team」よりチームを作成しておきます。

The screenshot shows the GitHub 'Teams' section for the organization 'sample-sample2'. The 'Teams' tab is selected. The main heading is 'Seamless communication with teams'. Below it, there are three sections: 'Flexible repository access' (with a plus icon), 'Request to join teams' (with a microphone icon), and 'Team mentions' (with a speech bubble icon). Each section has a brief description and two buttons at the bottom: 'New team' (green) and 'Learn more' (white).

The screenshot shows the 'Create new team' form. The 'Team name' field is filled with 'Write_2' and has a green checkmark. The 'Description' field is empty. The 'Parent team' section states 'There are no teams that can be selected.' The 'Team visibility' section shows 'Visible' (Recommended) is selected, with a note: 'A visible team can be seen and @mentioned by every member of this organization.' The 'Secret' option is also available, with a note: 'A secret team can only be seen by its members and may not be nested.' A red box highlights the 'Create team' button at the bottom.

「Team visibility」で「Secret」を選択した場合は、管理者権限の参照/編集が可能なチームになります。

The screenshot shows the 'Create new team' form in the GitHub interface. In the 'Team visibility' section, the 'Secret' radio button is selected, which is described as 'A secret team can only be seen by its members and may not be nested.' A red box highlights this section.

<1> 管理者権限のメンバーの場合

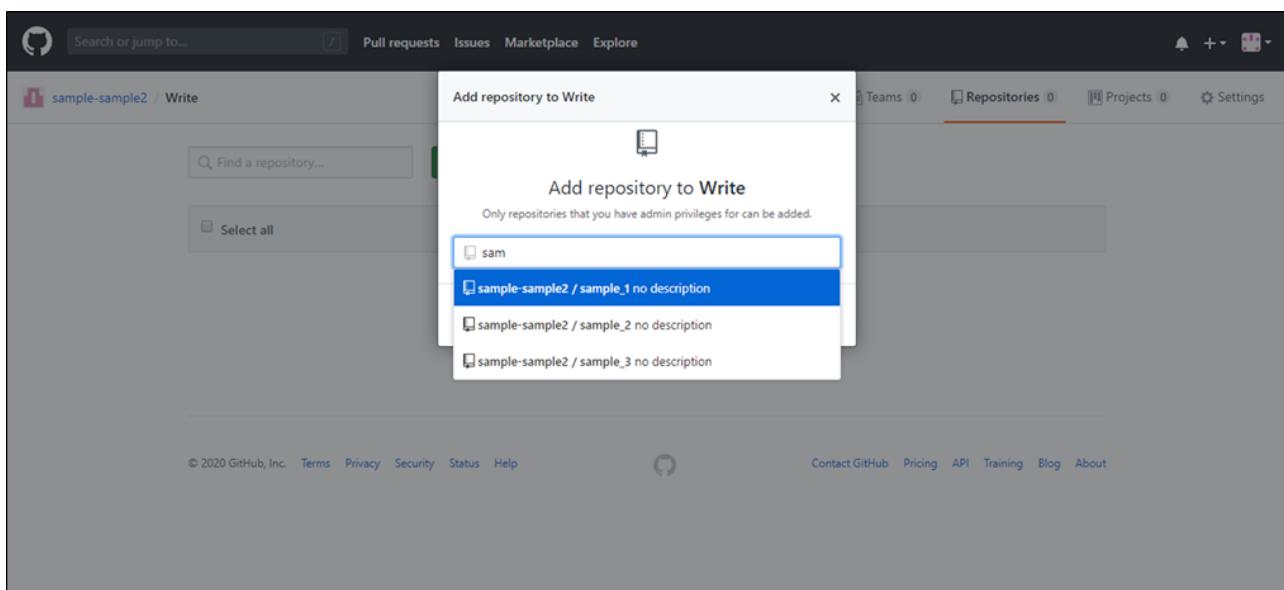
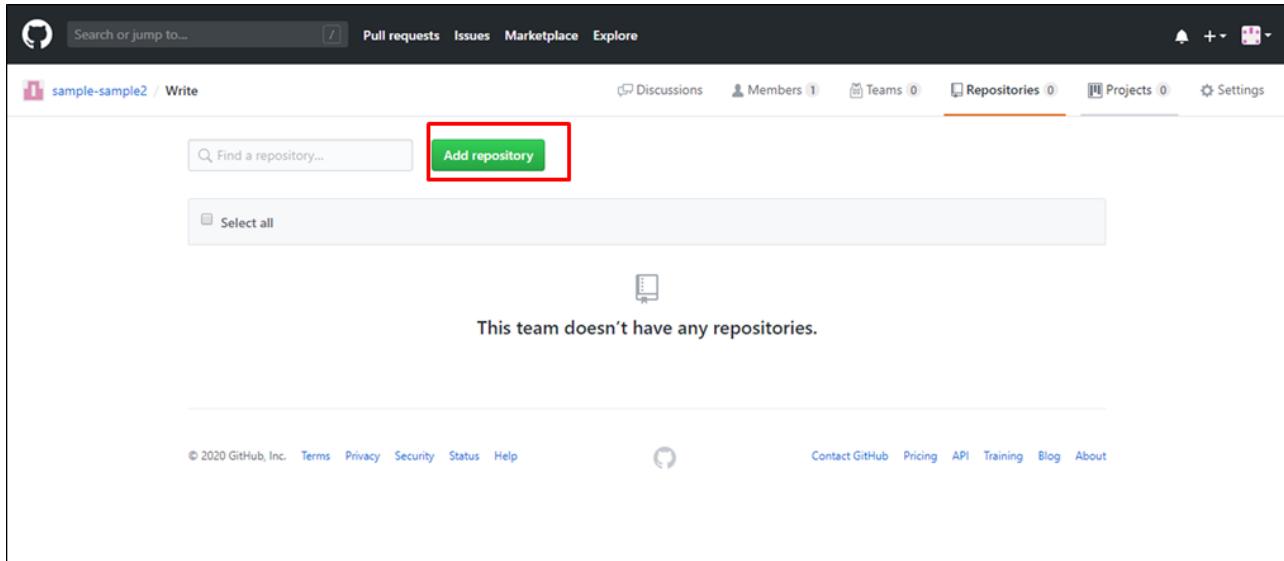


The screenshot shows the 'Teams' page in the GitHub interface. It displays a list of teams, including 'Write' (1 member, 0 teams) and 'Write_2' (1 member, 0 teams). A lightbulb icon is positioned to the left of the screen.

<2> 管理者権限以外のメンバーの場合

The screenshot shows the 'Teams' page in the GitHub interface. It displays a list of teams, including 'Write' (2 members, 0 teams) and 'Write_2' (2 members, 0 teams).

- チームを作成したら、下図のように「Repository」から対象のリポジトリを選択します。



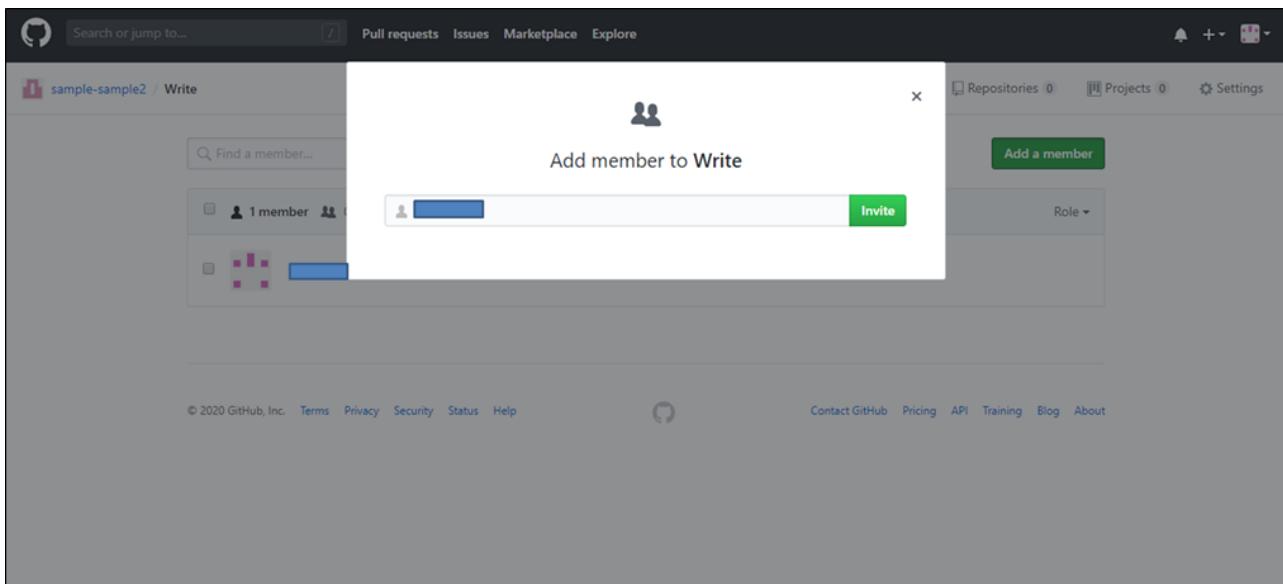
- リポジトリ毎に操作権限を指定します。
これで、チームに所属するメンバには各リポジトリに対して設定した操作権限が付与されます。

4. ソースコード/開発チーム管理指針

The screenshot shows the GitHub repository settings for 'sample-sample2'. It displays two repositories: 'sample-sample2/sample_1' and 'sample-sample2/sample_2'. Both were updated 11 minutes ago. A dropdown menu for 'sample-sample2/sample_1' is open, showing the 'Read' permission level selected. A tooltip provides a detailed description of the Read permission. Other permission levels shown include Triage, Write, Maintain, and Admin, each with its own description.

- チームを作成したら、下図のように「Members>Add a member」でチームに加えたいユーザを選択します。
※Organizationアカウントの作成者は、デフォルトで全てのチームに含まれます。

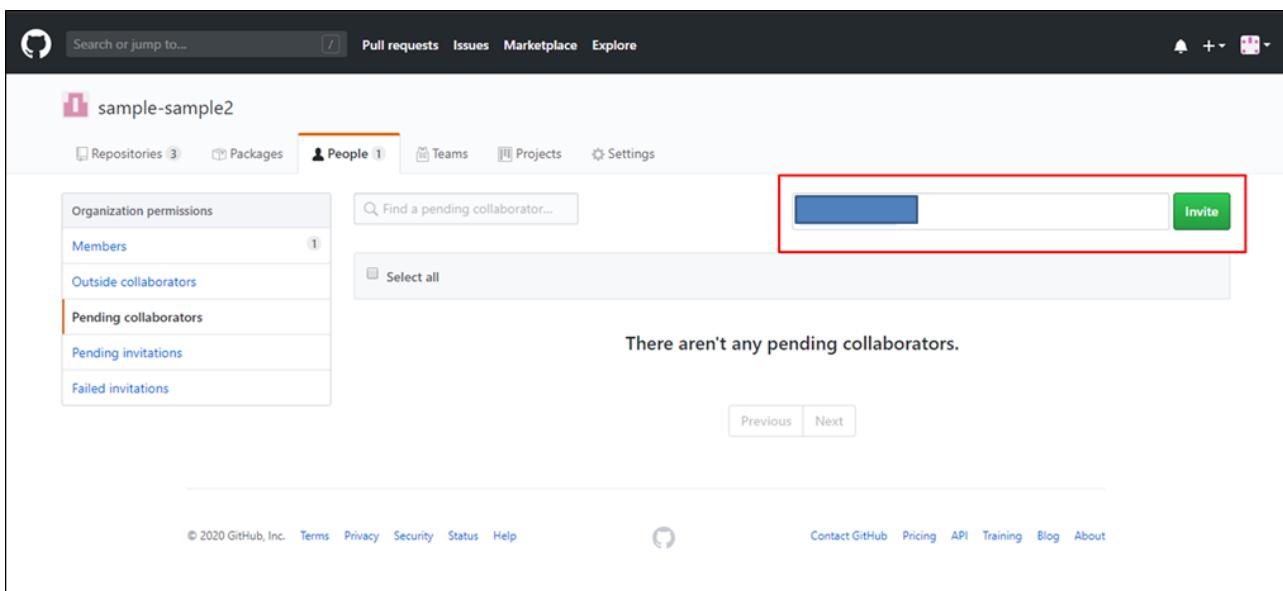
The screenshot shows the GitHub organization members page for 'sample-sample2'. It lists 1 member and 0 child team members. A red box highlights the 'Add a member' button. A tooltip for the button says 'Add a member to your organization'.



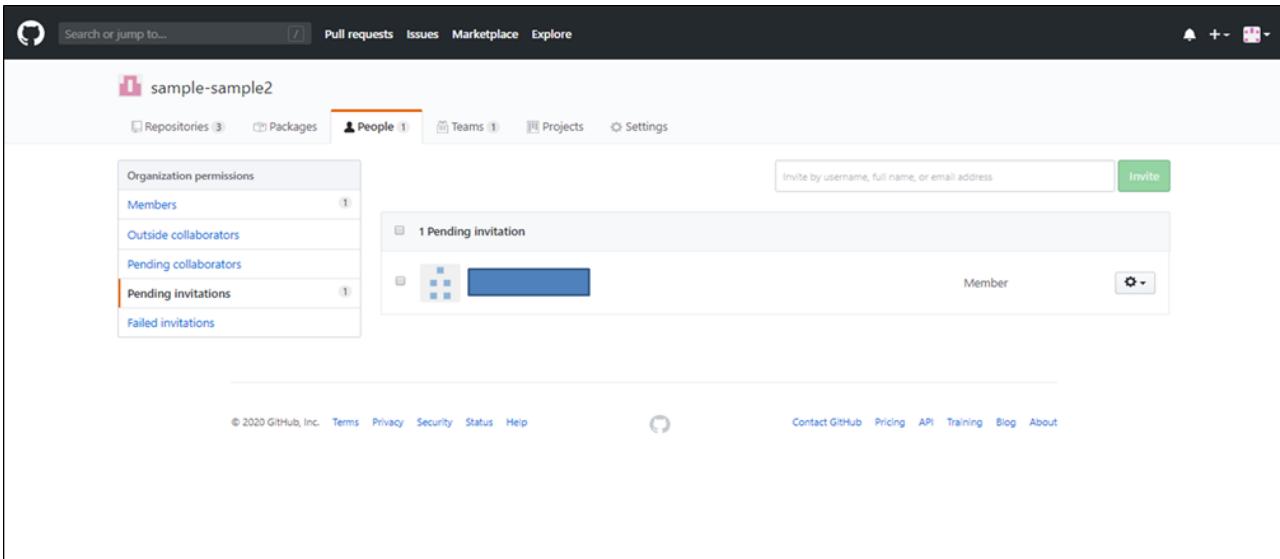
 チームにはGithubアカウントを持っている人なら誰でも招待できてしまうので、Githubユーザ名で検索を行う場合は予期せぬユーザをチームに追加しないように注意が必要です

-

- Organizationアカウントにメンバーを招待するには、下図のように「People>Pending collaborators」にて新規メンバー招待を送ります。



4. ソースコード/開発チーム管理指針



The screenshot shows the GitHub organization settings page for 'sample-sample2'. The left sidebar lists 'Organization permissions' with options like 'Members', 'Outside collaborators', 'Pending collaborators', 'Pending invitations' (which is selected), and 'Failed invitations'. On the right, there's a 'Pending invitation' section with a table showing one invite: a user with a blue profile picture and the role 'Member'. A green 'Invite' button is visible at the top right of this section. The bottom of the screen includes standard GitHub navigation links.

新規メンバーはデフォルトでMemberとして招待されますが下図のように「Edition invitation」でメンバーの権限を「Member/Owner」で変更することができます。

The screenshot shows the GitHub organization settings page for 'sample-sample2'. In the 'People' tab, there is a 'Pending invitation' section. A single invitation is listed, which has been accepted and is now shown as a member. The 'Edit invitation' button is highlighted with a red box.

The screenshot shows the 'Edit tanishizawa's invitation to sample-sample2' dialog box. It includes sections for 'Role in the organization' (radio buttons for Member and Owner), 'Teams — Optional' (a table showing membership in 'Write' and 'Write_2' teams), and buttons for 'Cancel invitation' and 'Update invitation'.

また、事前にチームを作成している場合は下図のように所属するチームを設定することができます。

This screenshot of the 'Edit invitation' dialog box is identical to the previous one, but the 'Teams — Optional' section is more clearly visible, showing the detailed membership counts for each team.

4.2.3. ソースコード管理

ソースコードの管理については、個人アカウントと同様に機能毎等、任意の単位でリポジトリを切ることができます。

また、各リポジトリへの操作権限は前述したようにTeamを作成することで、メンバー毎に詳細に指定すること

が可能です。

4.2.4. CI/CD契機の設定

CircleCIやWerckerといった代表的なSaaS型CI/CDツールを利用する場合、パイプライン起動の契機は全てCI/CDツール側で設定するためGithub上で何か特別な操作をする必要はありません。

- 設定方法はツールによって異なり、CircleCIのように設定ファイル中で設定するものや、WerckerのようにWebサイトで設定するものがあります。

CircleCI設定ファイル: パイプライン起動契機の設定



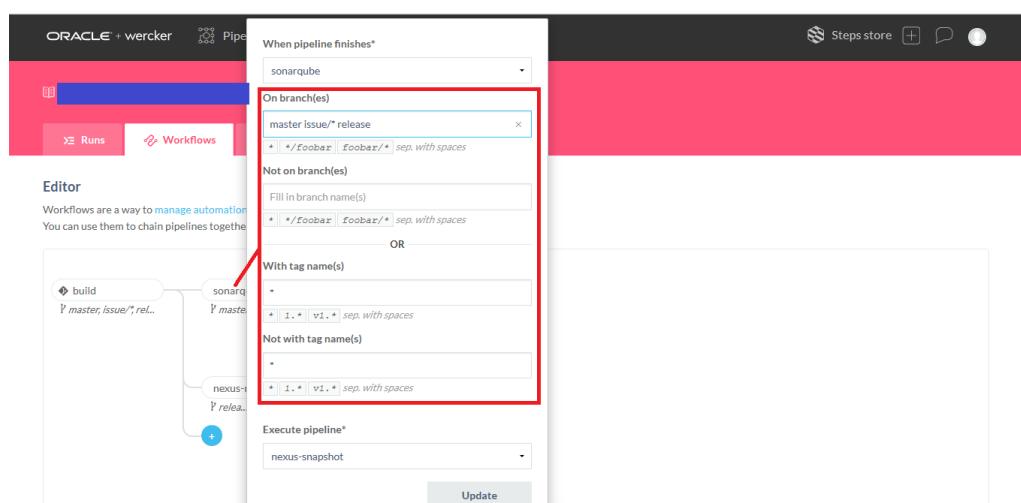
```

...
jobs:
  build:
  ...

workflows:
  ...
  workflow:
    jobs:
      build:
        filters:
          branches:
            only:
              - 対象とするブランチ名①
              - 対象とするブランチ名②
      ...
      nightly:
        triggers:
          schedule:
            cron: "25 * * * *"
          filters:
            branches:
            only:
              - 対象とするブランチ名②
      ...

```

Werckerサイト：パイプライン起動契機の設定



4.2.5. CI/CD結果の確認

Github上でissueブランチからmasterへのPRが作成された際、レビューアはコード修正内容に加えて修正後の

4. ソースコード/開発チーム管理指針

ソースコードが実環境で動作するものであるか確認する必要があります。

そのため、以下の手順でGithub上から各ブランチがCI/CDを正常に通過しているかどうか確認できるように設定します。

<1> PR時にCI/CD結果を確認したいリポジトリで「Settings」を開く。

The screenshot shows the 'Settings' tab of a GitHub repository named 'sample_Private'. The repository has 4 commits, 2 branches, 0 packages, 0 releases, and 2 contributors. A recent commit from CircleCI is listed, indicating a successful merge of pull request #2. The commit was made 21 minutes ago.

<2> SettingsのBranchesで「Branch protection rules」の「Add rule」を選択する。

The screenshot shows the 'Branches' section of the GitHub repository settings. Under 'Default branch', the 'master' branch is selected. In the 'Branch protection rules' section, there is a red box around the 'Add rule' button. Below it, a note states: 'Define branch protection rules to disable force pushing, prevent branches from being deleted, and optionally require status checks before merging. New to branch protection rules? Learn more.'

<3> 下図のように対象ブランチ名とルールを設定する画面になるので、「Branch name pattern」に対象ブランチ名のパターンを入力し、「Require status checks to pass before merging」と「Require branches to be up to date before merging」にチェックを入れる。その後、「Create」を押してルールを定義する。

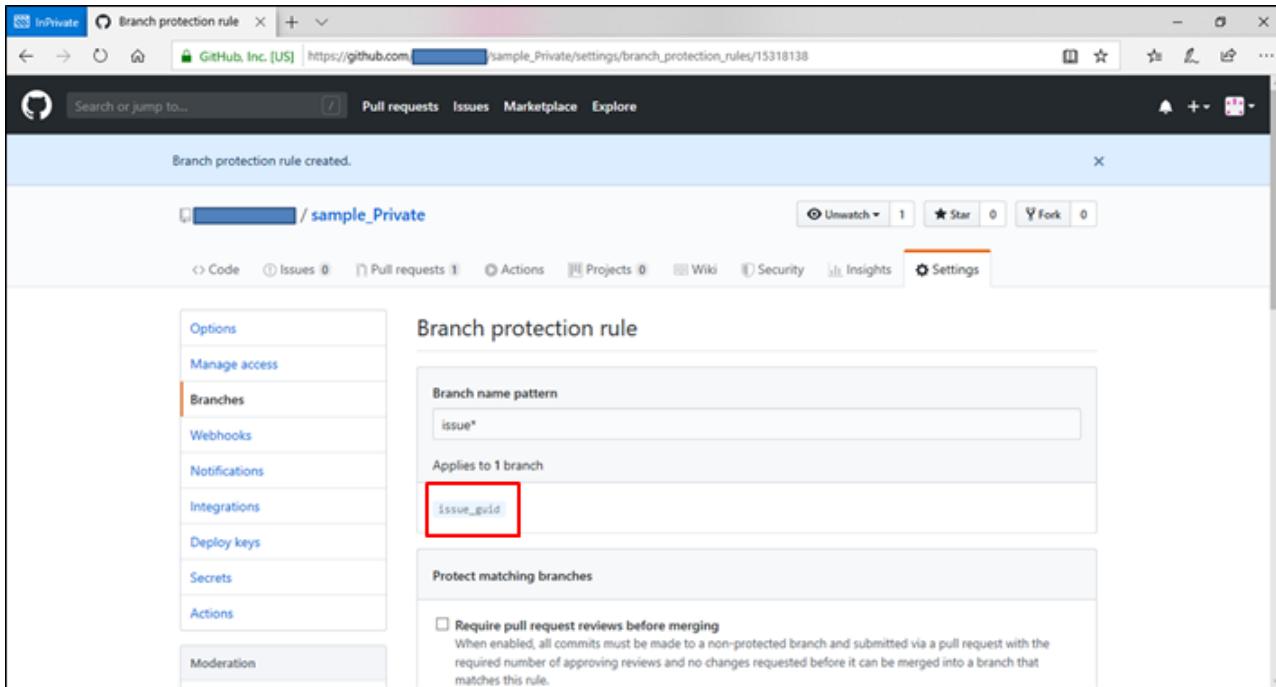
The screenshot shows the 'New branch protection' configuration page on GitHub. On the left, a sidebar lists options like Options, Manage access, Branches (which is selected), Webhooks, Notifications, Integrations, Deploy keys, Secrets, Actions, and Moderation. The main area is titled 'Branch protection rule'. It contains several sections:

- Branch name pattern:** An input field where a pattern like '*/main/*' is entered.
- Protect matching branches:** A section containing two checked checkboxes:
 - Require pull request reviews before merging**: Describes requiring reviews before merging.
 - Require status checks to pass before merging**: Describes requiring status checks to pass before merging. This section is highlighted with a red box.
 - Require branches to be up to date before merging**: Describes requiring branches to be up-to-date before merging. This section is also highlighted with a red box.
- Status checks found in the last week for this repository:** A list of status check items:
 - ci/circleci: Build Error
 - ci/circleci: build
 - ci/circleci: job_1
 - ci/circleci: job_2
 - ci/circleci: job_3
- Rules applied to everyone including administrators:** A section containing two unchecked checkboxes:
 - Allow force pushes**: Describes allowing force pushes.
 - Allow deletions**: Describes allowing users to delete branches.

At the bottom center is a large green 'Create' button, which is also highlighted with a red box.

<4> ルール作成後、下図のように対象ブランチが表示されるので内容が正しいか確認を行う。

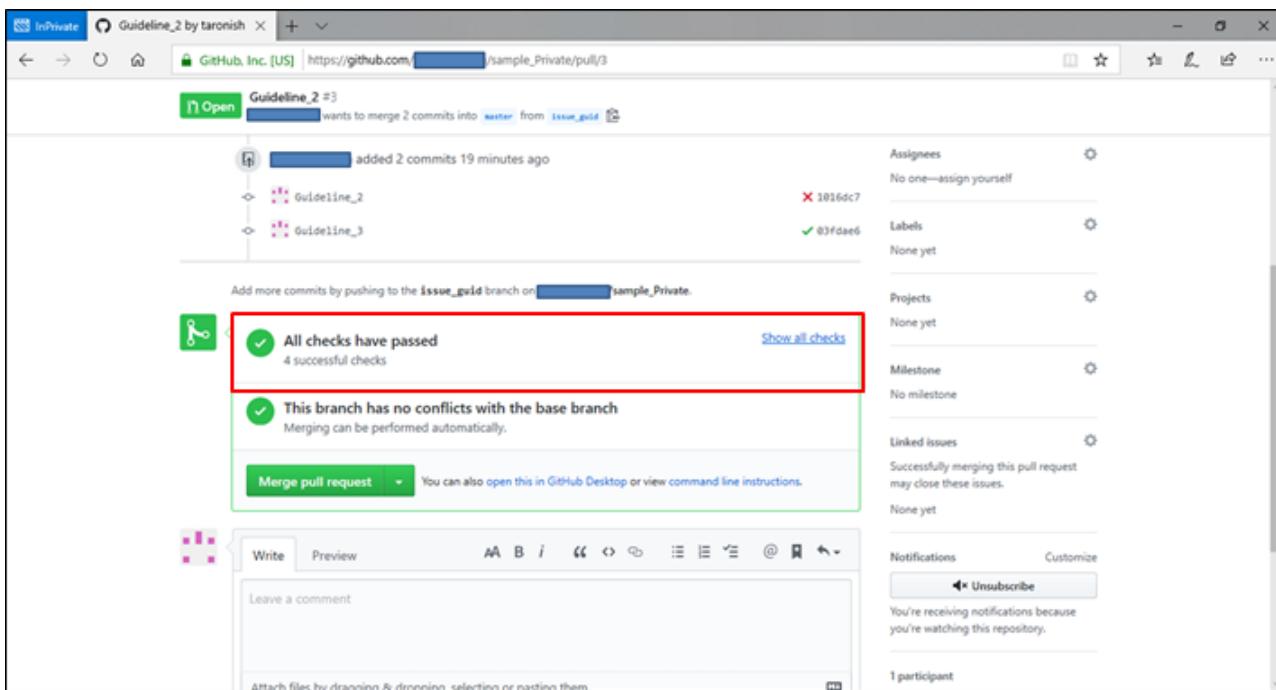
4. ソースコード/開発チーム管理指針



The screenshot shows the GitHub 'Branch protection rule' configuration page for a repository named 'sample_Private'. On the left, a sidebar lists options like 'Manage access', 'Branches', 'Webhooks', 'Notifications', 'Integrations', 'Deploy keys', 'Secrets', 'Actions', and 'Moderation'. The main area is titled 'Branch protection rule' and contains a 'Branch name pattern' field with 'issue*' and an 'Applies to 1 branch' section containing 'issue_guild' (which is highlighted with a red box). Below these are sections for 'Protect matching branches' and 'Require pull request reviews before merging'.

以上で、GitHubでのPR作成時にmerge対象のブランチがCI/CDを正常に通過しているか確認するための手順は完了です。

前述の設定を行うことで、下図のようにPR画面にmerge対象のブランチのCI/CD通過情報が表示されるようになります。



The screenshot shows a GitHub Pull Request (PR) merge screen for a PR titled 'Guideline_2 #3'. It displays a list of commits from the 'issue_guild' branch, including 'Guideline_2' and 'Guideline_3'. To the right, there are sections for 'Assignees', 'Labels', 'Projects', 'Milestone', 'Linked issues', and 'Notifications'. The most prominent feature is a summary box at the bottom left containing two green checkmarks: 'All checks have passed' (4 successful checks) and 'This branch has no conflicts with the base branch'. This summary box is also highlighted with a red box. A 'Merge pull request' button is located below it.

また、「Show all checks」でJob単位の通過情報を確認することができます。

Job単位の通過情報に記載の「Details」がCircleCIへのリンクとなっており、CircleCIアカウントを持っていなくともCircleCI画面で結果を確認することができます。

This screenshot shows a GitHub pull request page for a repository named 'sample_Private'. The pull request is titled 'Guideline_2' and has a status of 'Open'. It shows 2 commits from the user 'blue' merged into the 'master' branch from the 'issue_gold' branch. The commit history includes 'Guideline_2' and 'Guideline_3'. The right sidebar contains fields for 'Assignees', 'Labels', 'Projects', 'Milestone', 'Linked issues', and 'Notifications'. A prominent red box highlights the 'All checks have passed' section, which lists four successful CircleCI tests: 'ci/circleci: build', 'ci/circleci: job_1', 'ci/circleci: job_2', and 'ci/circleci: job_3'. Below this, another section states 'This branch has no conflicts with the base branch'.

CI/CDでエラーが発生している場合は下図のよう画面表示されます。

This screenshot shows a GitHub pull request page for the same repository 'sample_Private'. The pull request 'Guideline_2' is still open and shows 3 commits from 'blue' merged into 'master' from 'issue_gold'. The commit history includes 'Guideline_2', 'Guideline_3', and 'Guideline_4'. The right sidebar remains the same. A red box highlights the 'All checks have failed' section, which shows one failing check: 'ci/circleci: build — Your tests failed on CircleCI'. Below this, the 'This branch has no conflicts with the base branch' message is present.

個々のブランチをpushやmergeした際のCI/CD通過情報についても、CircleCIと連携させるだけで下図のように「☒」「☒」が表示されるようになります。これをクリックすると、PR画面と同様にCircleCI画面へのリンクが張られた通過情報を確認することができます。



The screenshot shows a GitHub repository page for 'sample_Private'. The repository has 5 commits, 2 branches, 0 packages, 0 releases, and 2 contributors. The 'Branch: Issue_guild' is selected. The CI/CD status section shows 'All checks have passed' with 4 successful checks. Each check is listed with a green checkmark and a link to 'Details'. The last check was a 'test commit' made 1 minute ago. The entire status section is highlighted with a red box.

All checks have passed		
4 successful checks		
✓	c/circleci: build — Your tests passed on Ci...	Details
✓	c/circleci: job_1 — Your tests passed on Ci...	Details
✓	c/circleci: job_2 — Your tests passed on Ci...	Details
✓	c/circleci: job_3 — Your tests passed on Ci...	Details