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. 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）を対象とします
- APの開発フレームワークは「SpringBoot」を利用します
- APのビルドツールは「Gradle」を利用します

- ビルド成果物は単独のjarファイルです
- 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ます

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のリーバイスプロキシSonarQubeとNexusを配置します。

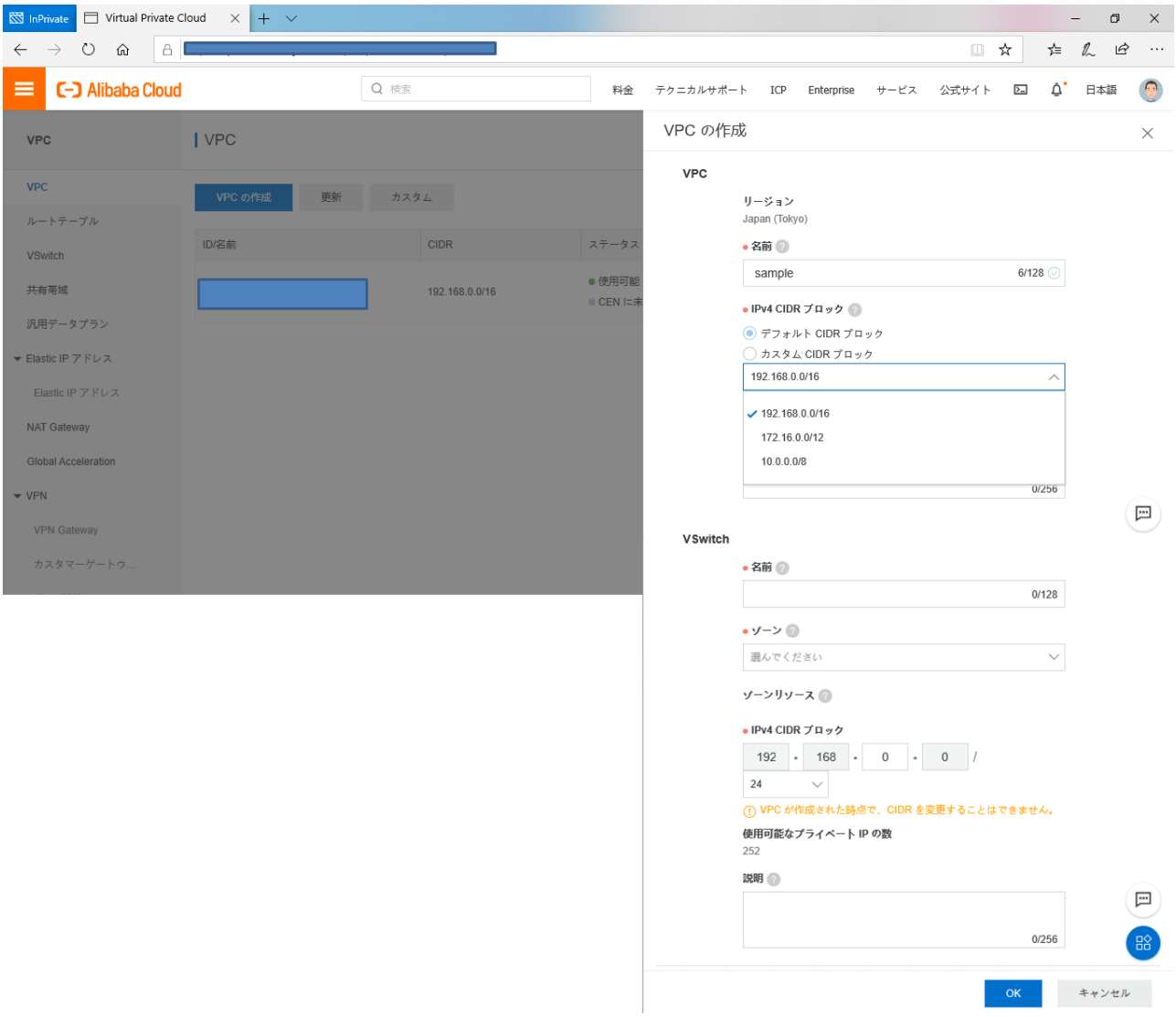
また、各サーバへの接続はロードバランサによるパスルーティングによって振り分けます。

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。


InPrivate
Virtual Private Cloud
Alibaba Cloud
検索
料金
テクニカルサポート
ICP
Enterprise
サービス
公式サイト
日本語
VPC
VPC
ルートテーブル
VSwitch
共有帯域
汎用データプラン
Elastic IP アドレス
Elastic IP アドレス
NAT Gateway
Global Acceleration
VPN
VPN Gateway
カスタマーゲートウ...
VPC
VPC の作成
更新
カスタム
ID/名前
CIDR
ステータス
192.168.0.0/16
使用可能
CEN に未
VPC の作成
VPC
リージョン
Japan (Tokyo)
名前
sample
6/128
IPv4 CIDR ブロック
デフォルト CIDR ブロック
カスタム CIDR ブロック
192.168.0.0/16
192.168.0.0/16
172.16.0.0/12
10.0.0.0/8
0/256
VSwitch
名前
0/128
ゾーン
選んでください
ゾーンリソース
IPv4 CIDR ブロック
192
168
0
0
/
24
VPC が作成された時点で、CIDR を変更することはできません。
使用可能なプライベート IP の数
252
説明
0/256
OK
キャンセル

VPCのCIDRブロックは、下記の3種から選択することができます。

- 今回は検証をメインとした開発のため最小規模の「192.168.0.0/16」を選択しました。
- 実運用においては、「システム規模」や「連携先システム間のプライベートIP重複」、「要件変更によるシステムの拡大」を考慮して大きめに指定する必要があります。

CIDRブロックはVPC作成後に変更することができません。

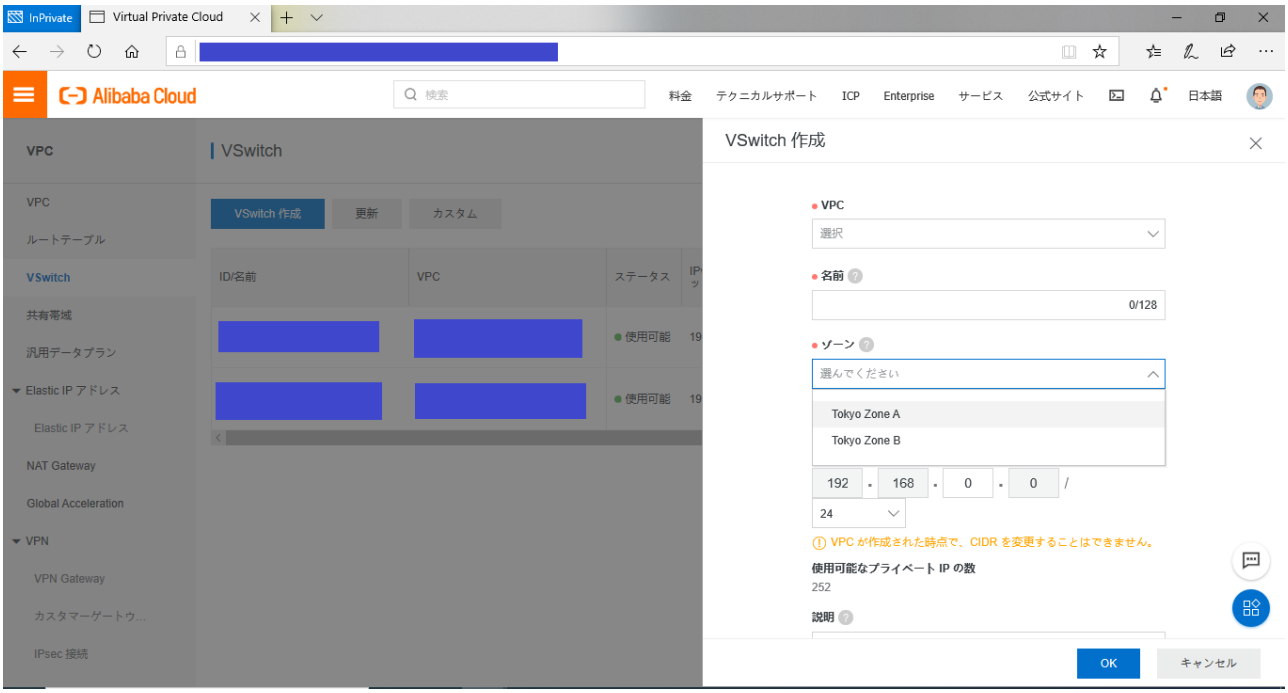
表 1. CIDRブロック

CIDRブロック	利用可能なプライベートIP数
192.168.0.0/16	65,532
172.16.0.0/12	1,048,576
10.0.0.0/8	16,777,212

2.2.2. VSwith

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VSwith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VSwithコンソールから行います



- VSwithを作成することで、VPC内にサブネットを作成することができます。
  ただし、1VPC当たりのVSwith数の上限は24個までとなります。+
- VSwithのゾーンには、VPCのリージョンに対応するゾーンを選択します。
  　例えば、リージョンが「Japan (Tokyo)」の場合は「Tokyo Zone A」または「Tokyo Zone B」を選択します。
  各ゾーン間は別ネットワークとなるため、イントラネットを介して相互通信を行います。
  そのため、異なるゾーンにリソースを配備しておくことで一方のゾーンで障害が発生した際のリカバリーに備えるような運用ができます。

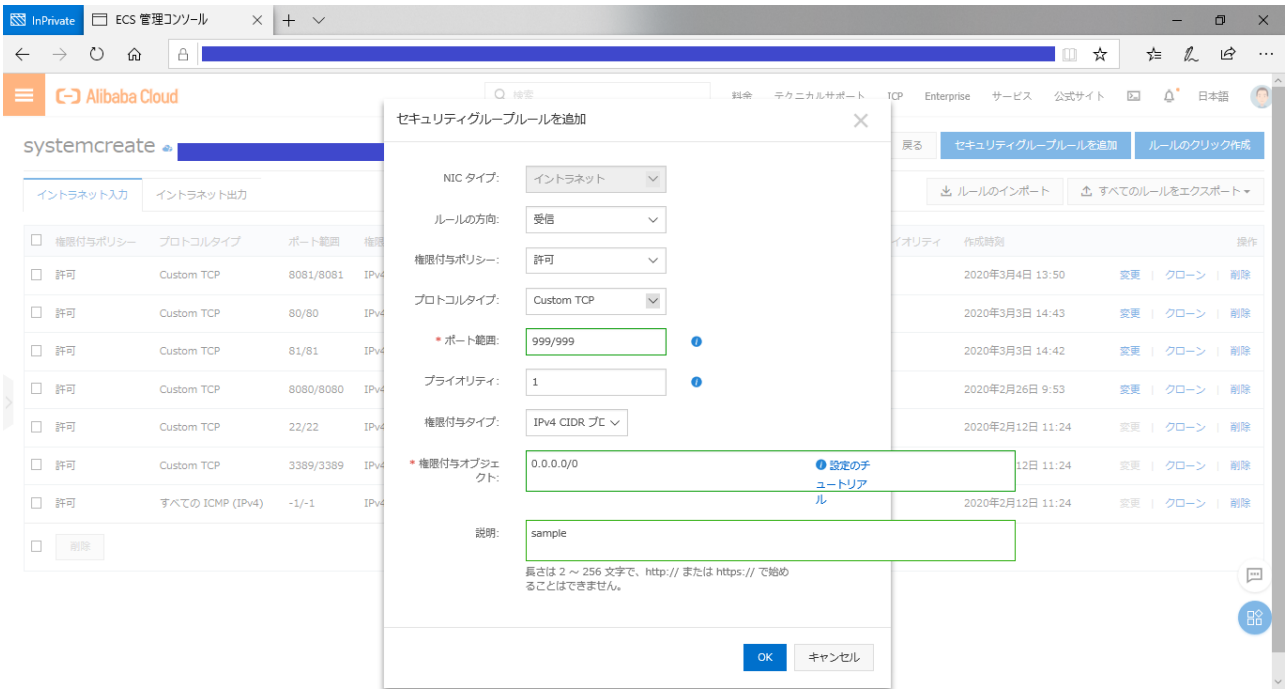
2.2.3. ECS（Elastic Compute Service）

AlibabaCloud上の仮想サーバがECS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。

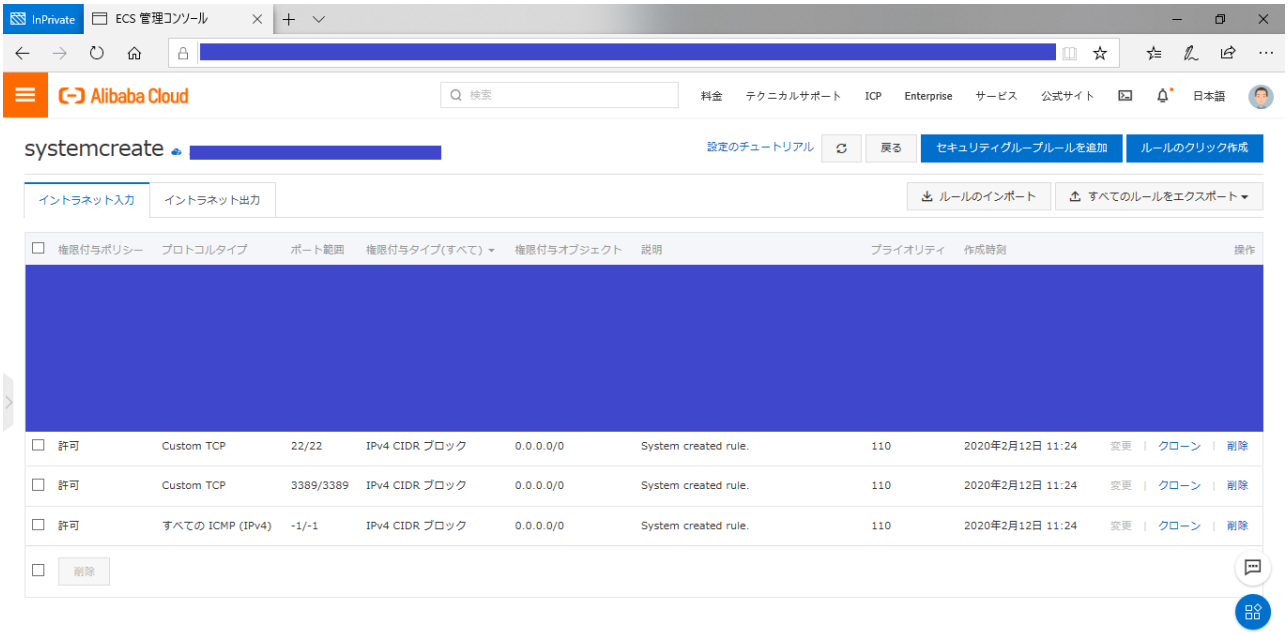
2.2.4. セキュリティグループ

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）

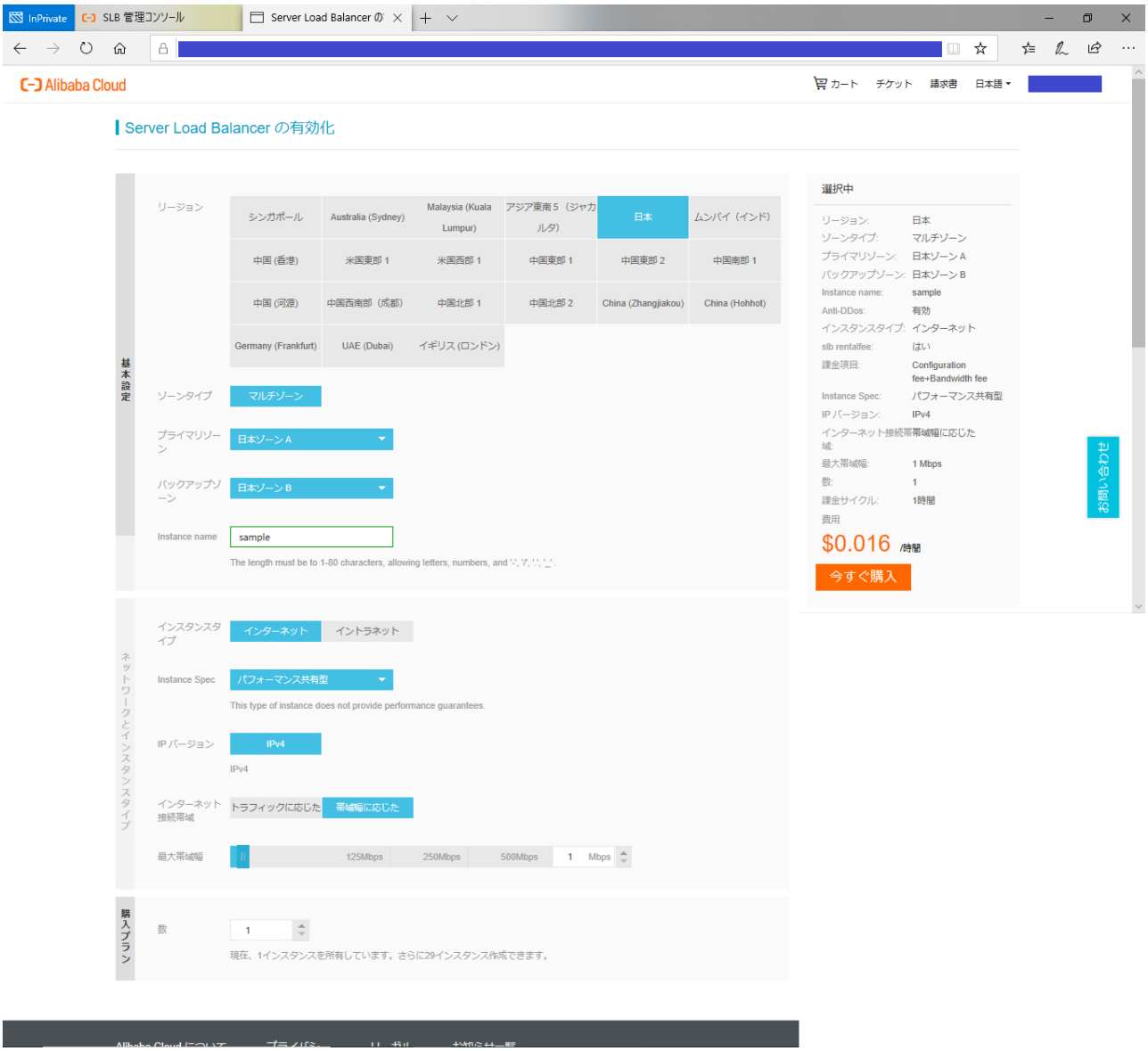


ESCの初期作成時には、下図のように「-
1」「22/22」「3389/3389」ポートのみが利用可能となっているため、適宜編集が必要です。

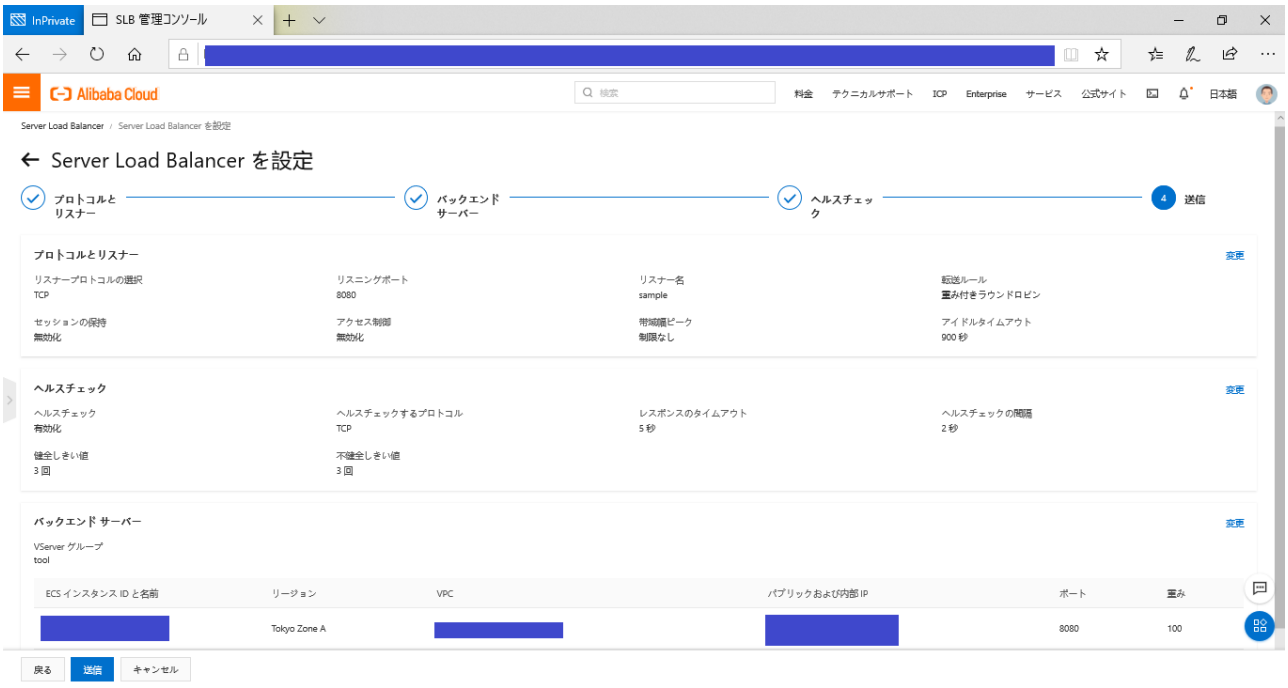


2.2.5. SLB（Server Load Balancer）

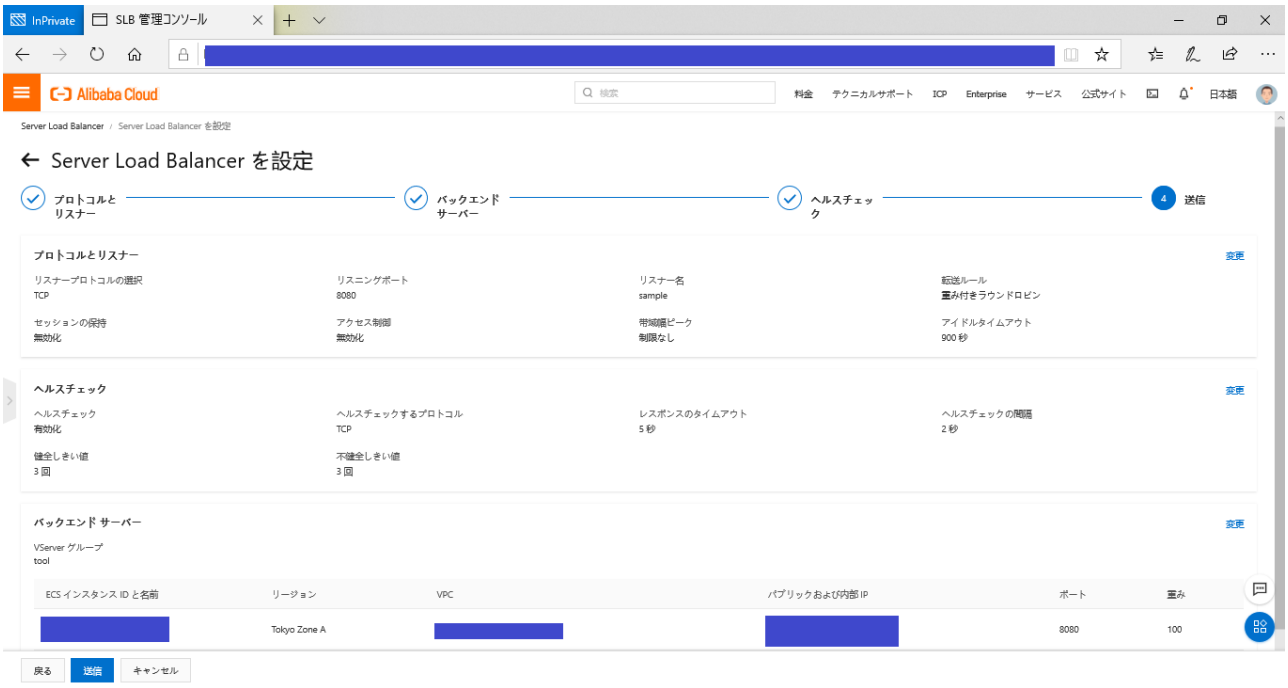
ロードバランサです。
下図のようにSLBコンソールよりSLBの購入を行います。



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。



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。

2.3.1. Webアプリケーション用サーバ

- このサーバには、開発したAPを配置します。
- サーバスペック

> 今回配置するAPは1jarファイルからなる小規模なAPとなります。
> そのため、サーバのスペックは下記のように最低値のものを選択しています。
> 実運用では、開発するAPの規模や同時実行するアプリケーション、想定されるアクセス数に応じて適したインスタンスを購入します。

表 2. 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

- セキュリティグループ

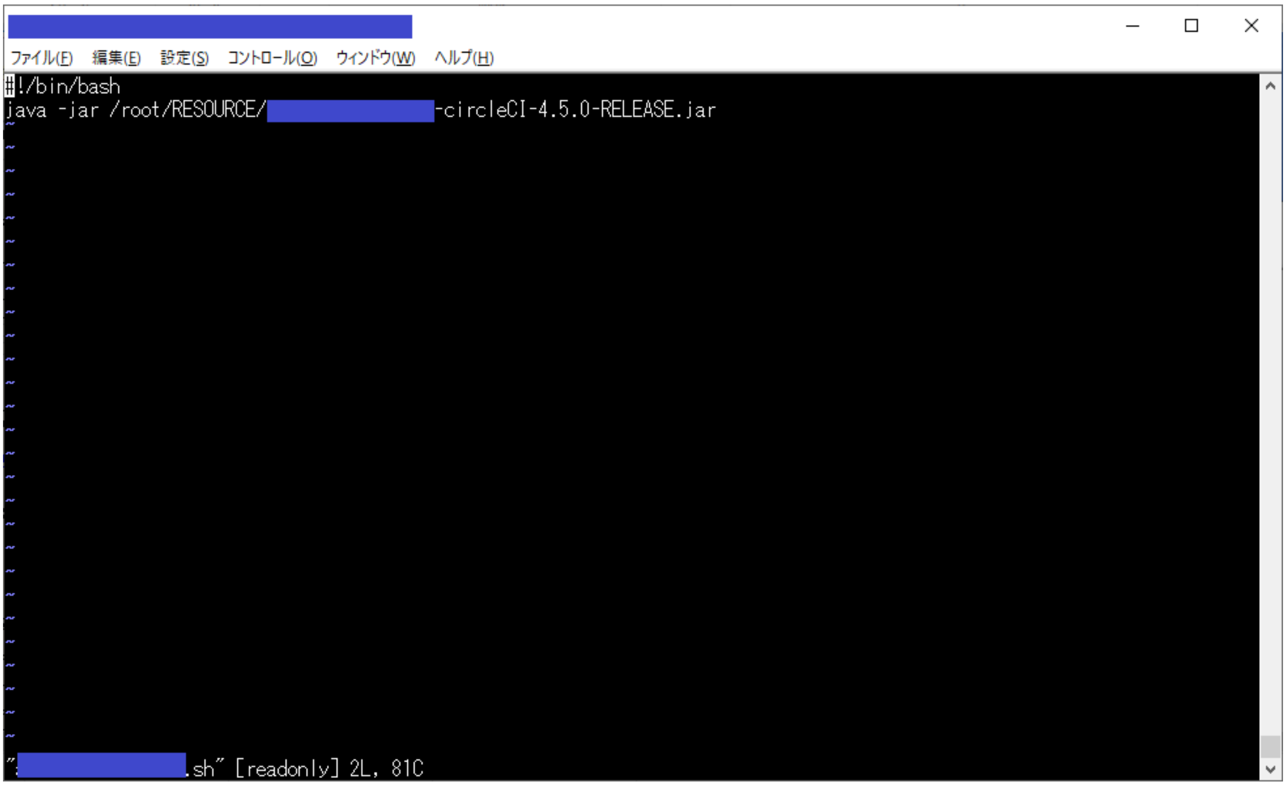
> セキュリティグループについてはAPが80番号ポートで起動しているため、今回は80番ポートのみアクセス可能に設定しています。

表 3. セキュリティグループ

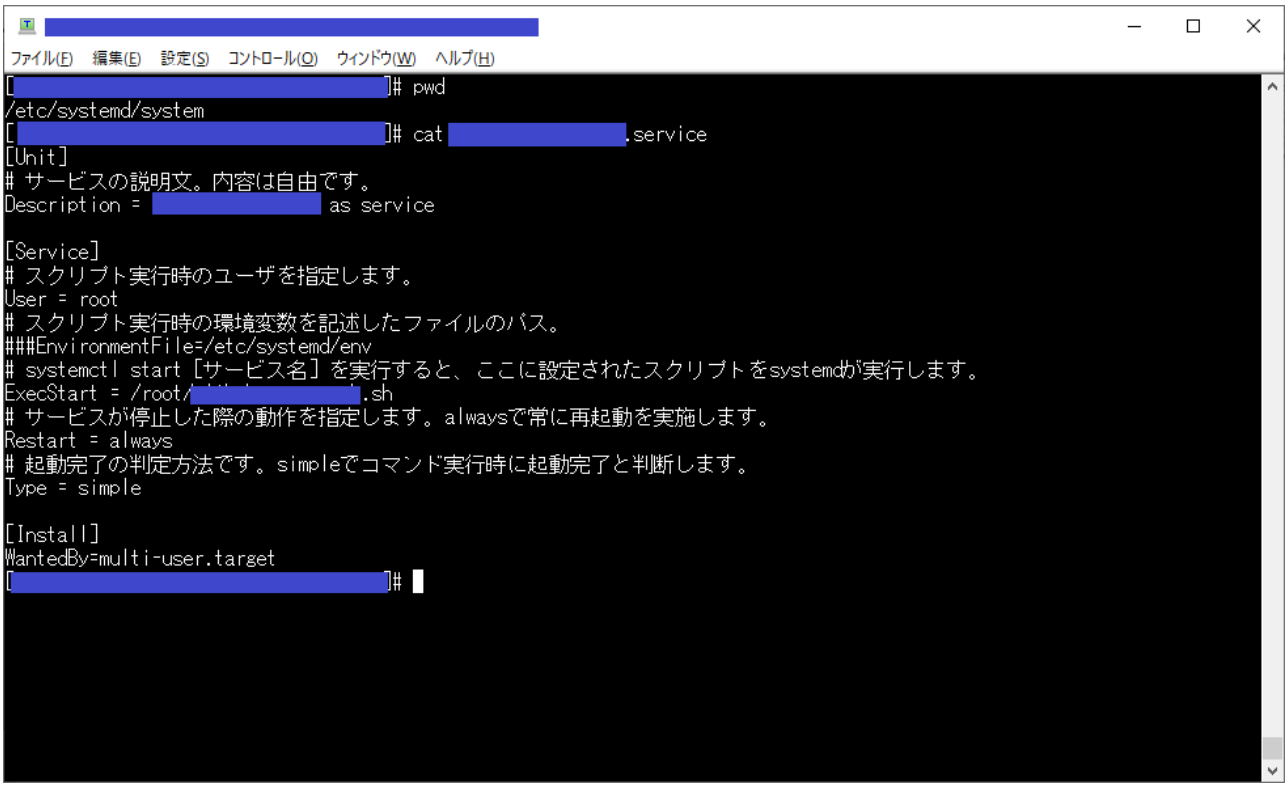
入力/出力	ポート番号	権限付与IPアドレス
入力	80/80	0.0.0.0/0 ※検証のためフルオープン

- 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。



> 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。



2.3.2. SonarQube・Nexus用サーバ

-

  このサーバでは、CI/CDツール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

> このサーバでは、SonarQubとNexusに加えてリバースプロキシ用のNginx、PostgreSQL重めのアプリケーションを動かします。
> そのため、サーバのスペックは下記のように十分に運用に耐えられる値のものを選択しています。

表 4. 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

- セキュリティグループ

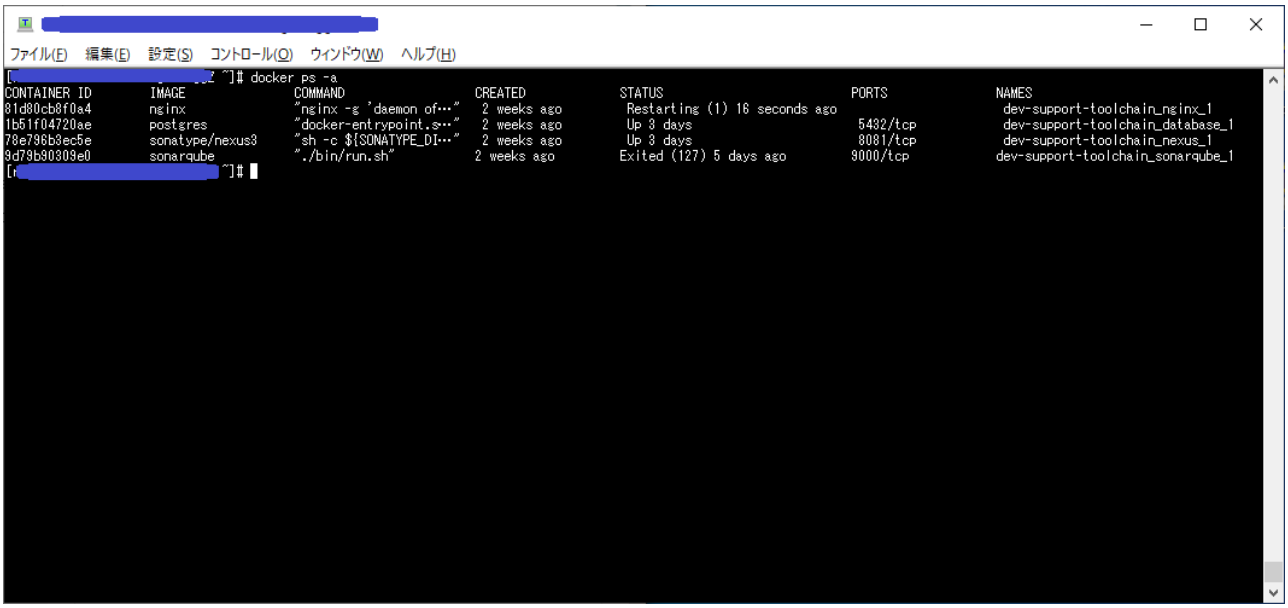
> セキュリティグループについてはSLBのヘルスチェック用に80番号ポートを、Nginxにつなぐために8080番ポートをアクセス可能に設定しています。

表 5. 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 ※検証のためフルオープン

- 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。



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 6. 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
SLBのIP	/sonarqube	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	/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(SonarQubeのポート番号)に転送されます
また、「~/nexus」は8001番ポート(Nexusのポート番号)に転送されます。

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が、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 7. 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、ビルド時間：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いて、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 8. 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内で下記の様式に従ってタグを生成する。

表 9. 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 10. CI/CDフロー

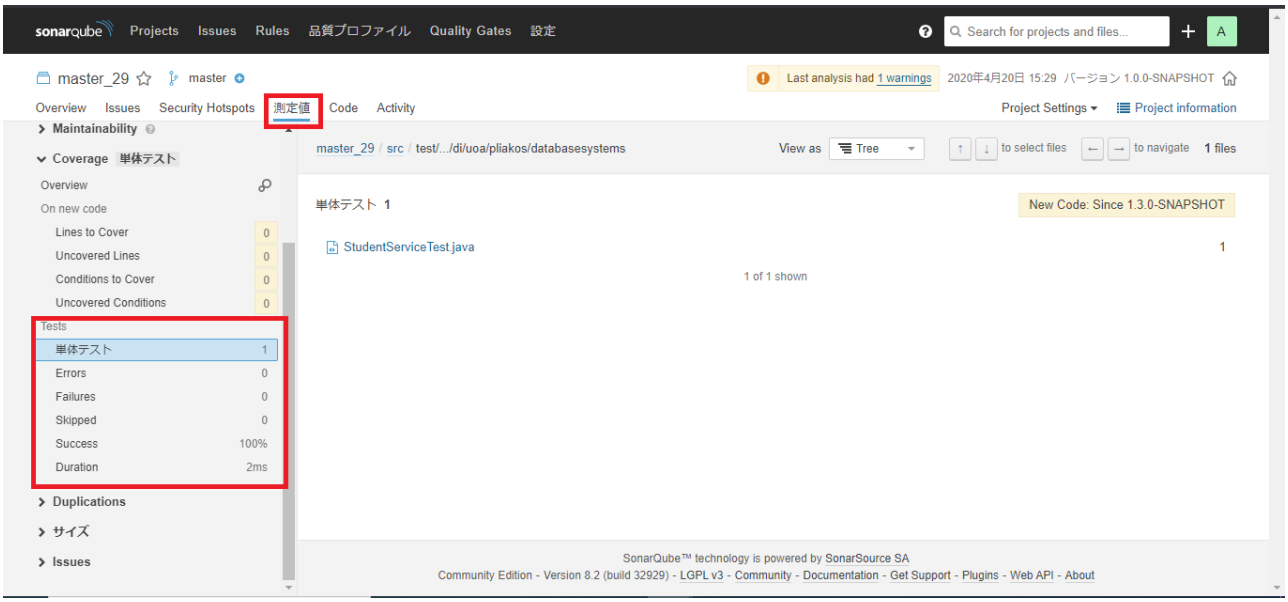
ブランチ	パイプライン起動の契機	実行処理
issueブランチ	ローカル環境よりGitHub上にpushした際	• ビルドチェック • Junitテスト • SonarQubeへのJunitテスト結果連携、静的検証
masterブランチ	PRをmergeした際	• ビルドチェック • Junitテスト • SonarQubeへのJunitテスト結果連携、静的検証 • Nexusへのライブラリ登録 • 最新のmasterブランチに対してタグ打ち（名：version-X.X.X-SNAPSHOT）
releaseブランチ	ローカル環境よりGitHub上にpushした際	• ビルドチェック • Junit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

SonarQubeではリポジトリごとにパイプライン中で実施したJunitテストの結果集積と静的検証の実施、カバレッジ情報の算出を行うことで、開発/修正したソースコードの品質管理を行います。

3.2.1. Junitテスト結果連携

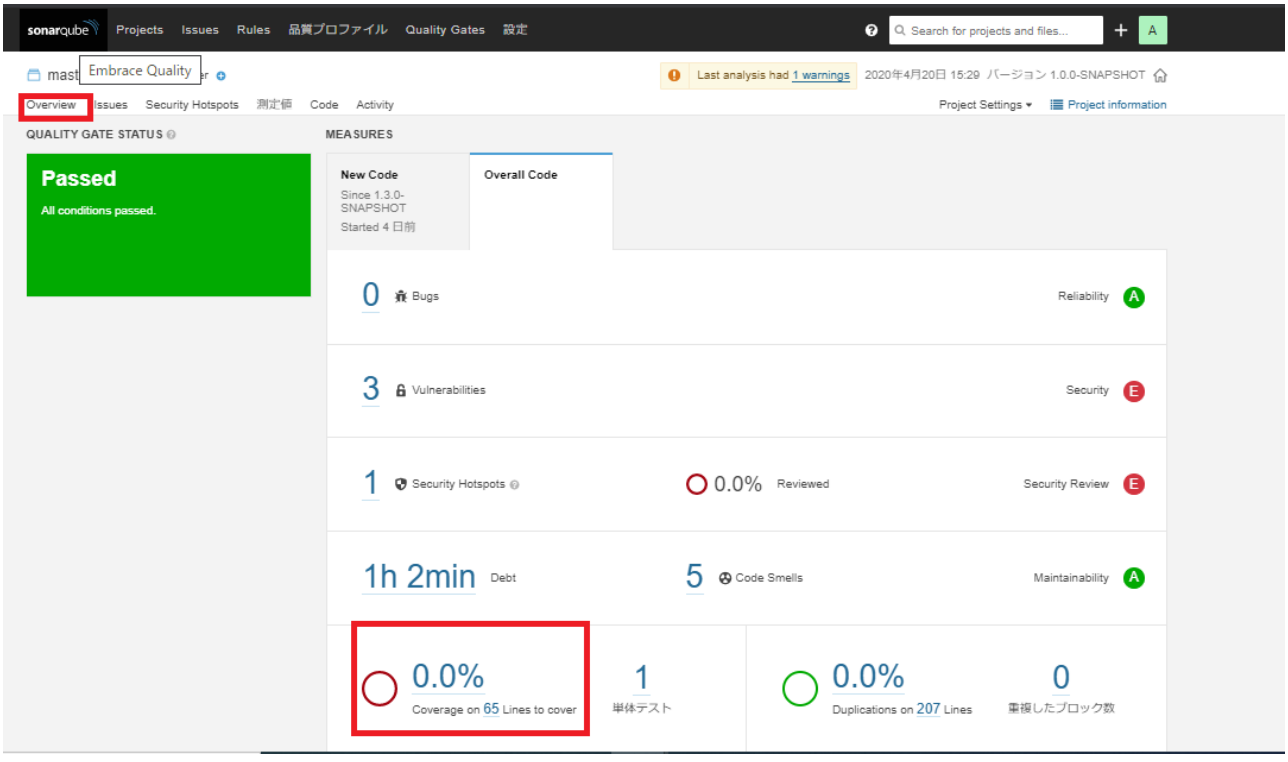
パイプラインで実施された結果は、SonarQubeに連携されて下図のように情報が表示されます。



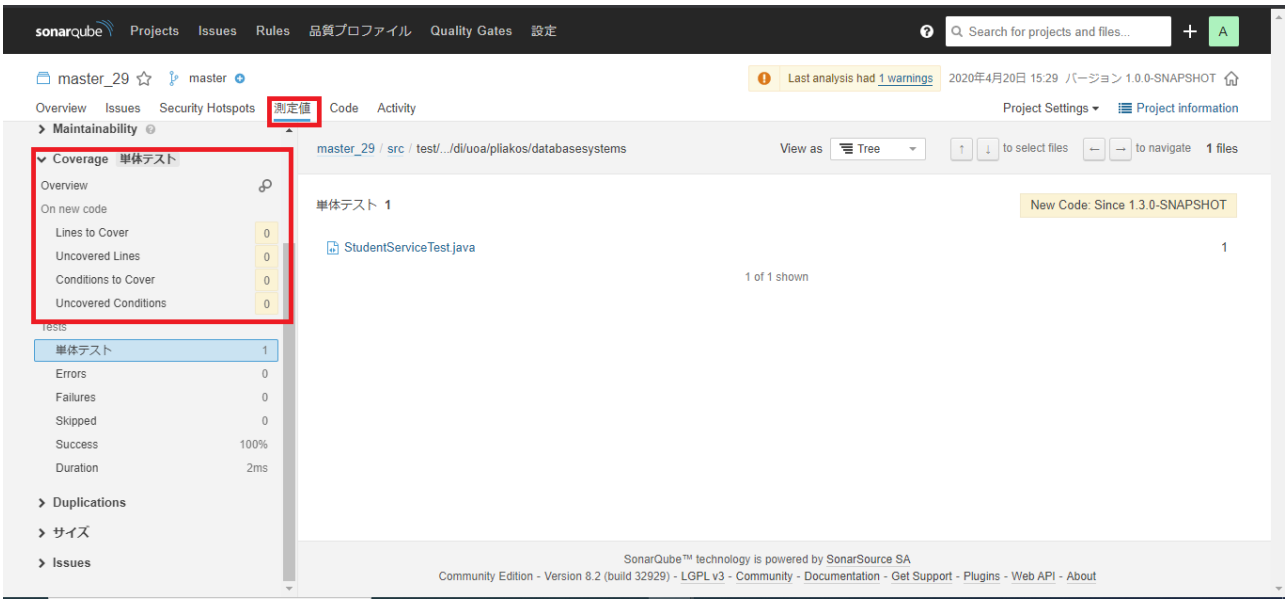
3.2.2. カバレッジ情報

Junitテストコードのカバレッジは以下のように表示されます。

- 概要

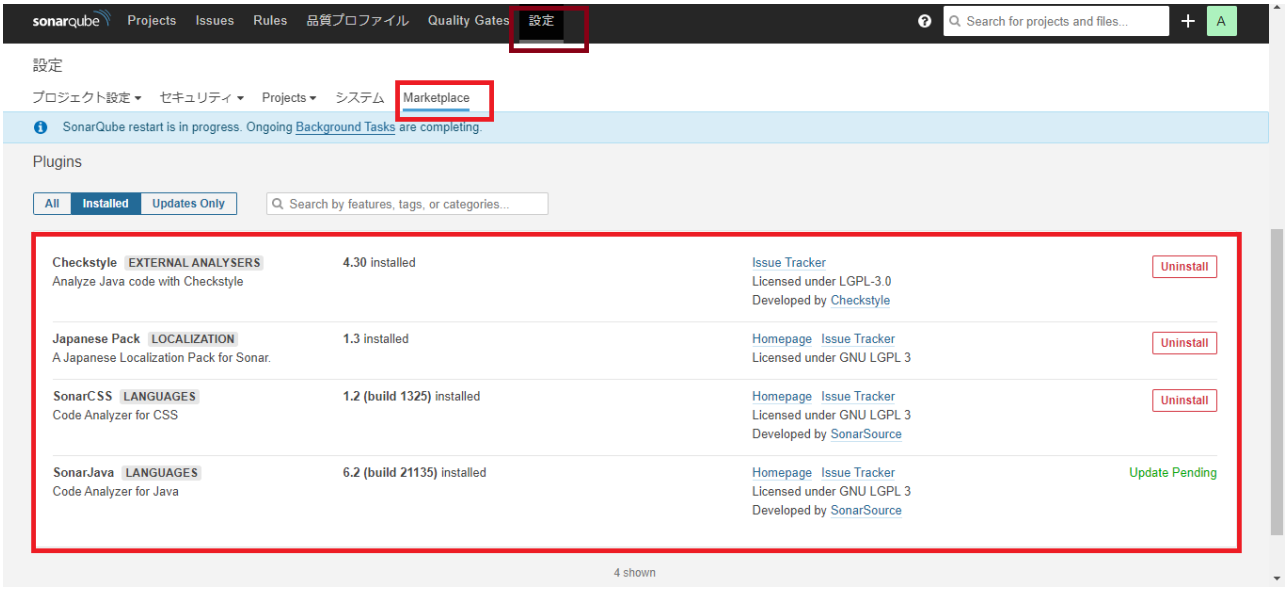


- 詳細



3.2.3. 静的検証

SonarQubeで実施する静的検証は、下記のように「設定＞MarketPlace」から定義をインストールして設定します。
今回は「Checkstyle」と「SpotBugs」を実施していますが、実運用ではPJのルールに従って適宜設定を行います。



3.3. Nexus連携について

Nexusではリポジトリごとにビルド成果物を連携することで、実環境で動くことが保証されたアプリケーションのバージョン管理を行います。

3.3.1. Nexus連携の契機

- 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します

後述するように同バージョンのビルド資材格納を認めない運用とするため、issueブランチをGithubに登録するタイミングで、バージョンの設定を行う必要があります。
また、SNAPSHOTリポジトリは「XXX-SNAPSHOT」形式のバージョン名しか登録できません。

build.gradle：バージョンの変更

```
…
##issueブランチ作成時にバージョンのを一つ上げます。
version "1.1.0-SNAPSHOT"
…
```

- 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します。

SNAPSHOT同様に同バージョンのビルド資材格納を認めない運用とするため、releaseブランチをGithubに登録するタイミングで、バージョンの設定を行う必要があります。

build.gradle：バージョンの変更

```
…
##issueブランチ作成時にバージョンのを一つ上げます。
version "1.1.0-RELEASE"
…
```

3.3.2. リポジトリの設定

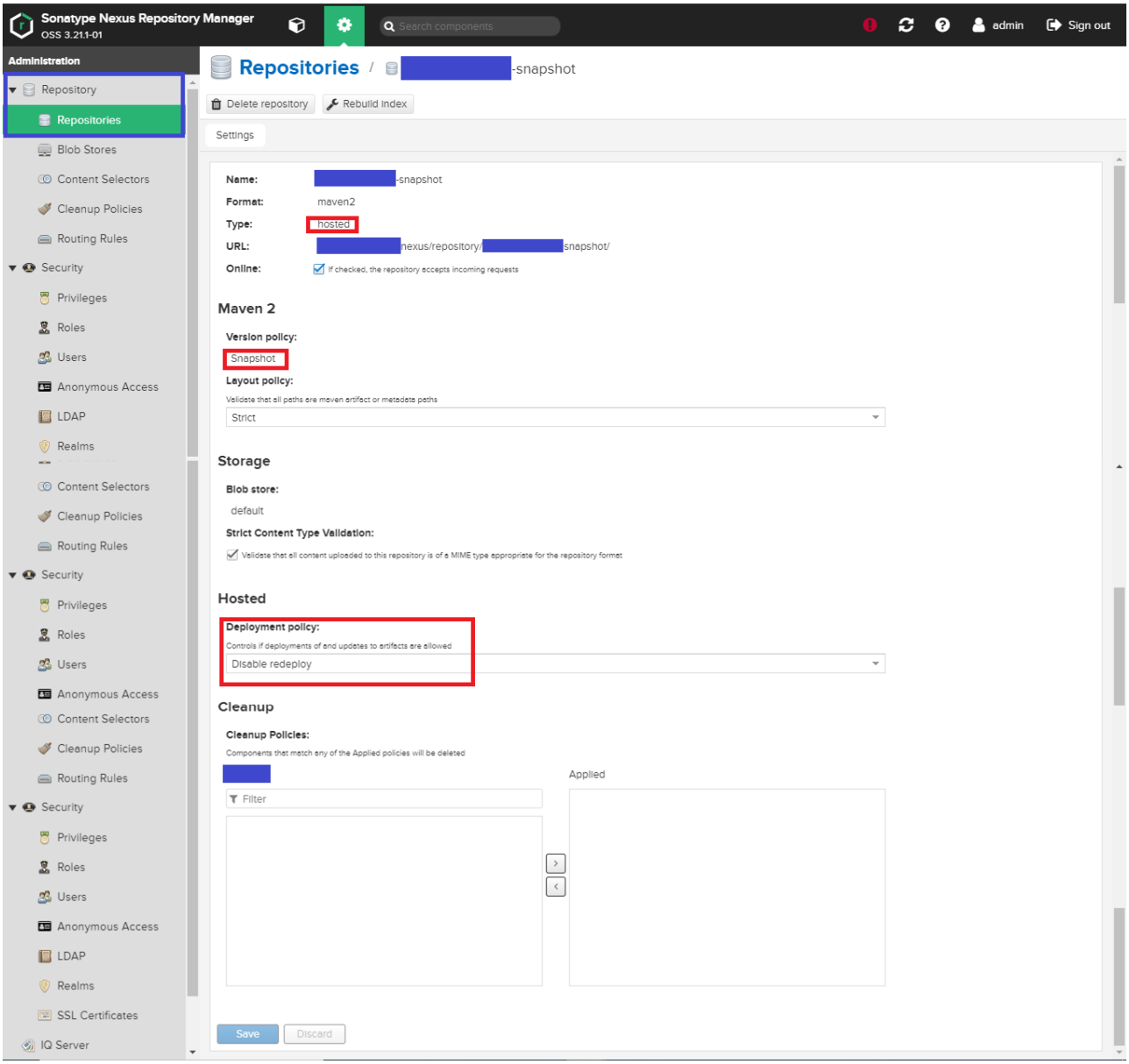
各リポジトリは、以下の設定となっています。
その他の項目はデフォルト値です。

表 11. リポジトリ設定

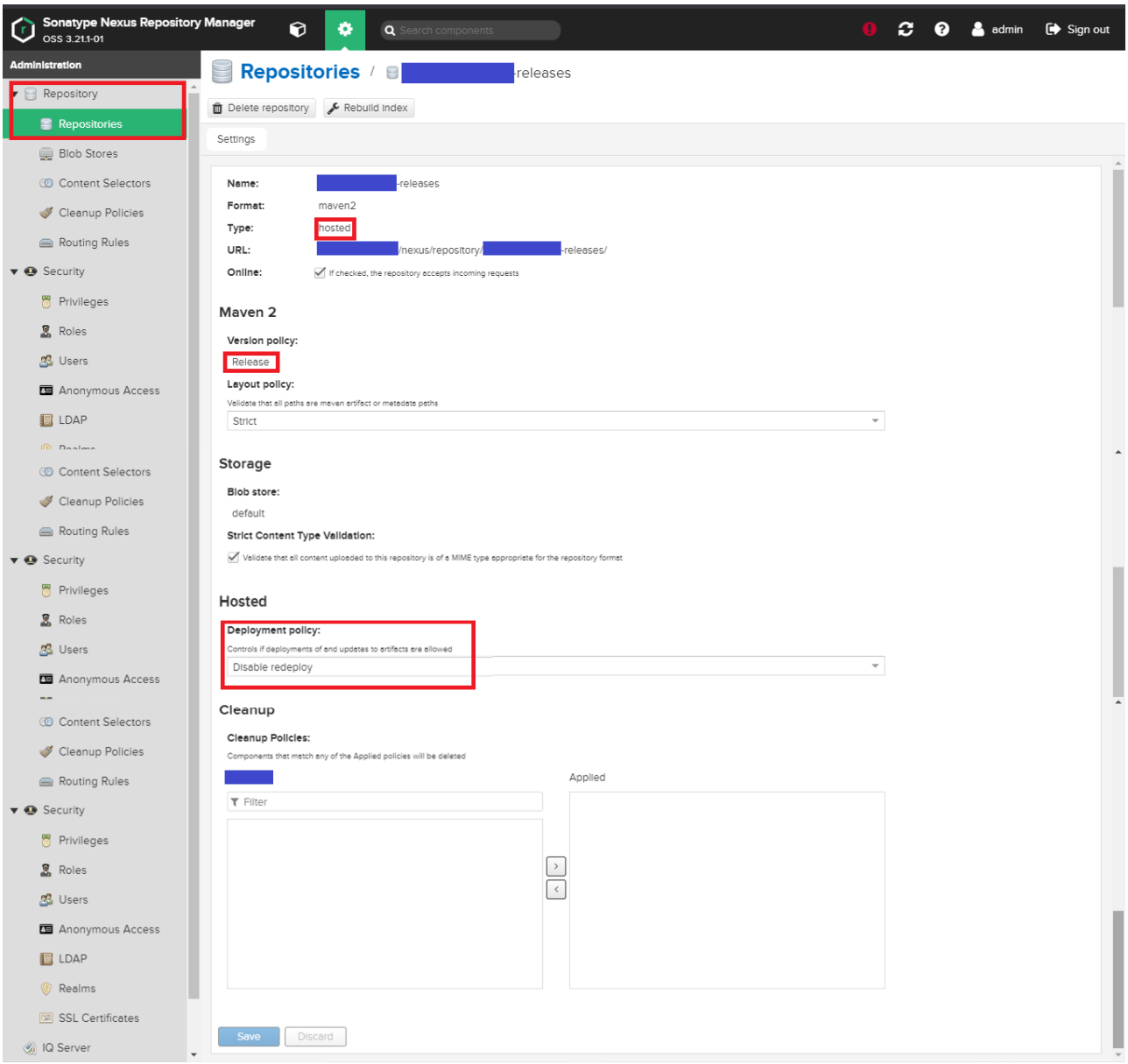
リポジトリ	リポジトリタイプ	バージョンポリシー	同バージョンのデプロイ可否（Deployment policy）
SNAPSHOTリポジトリ	hosted ※独自のリポジトリを作成する	Snapshot ※スナップショット版のライブラリだけを含むリポジトリ	Disable redploy

リポジトリ	リポジトリタイプ	バージョンポリシー	同バージョンのデプロイ可否（Deployment policy）
RELEASEリポジトリ	hosted ※独自のリポジトリを作成する	Release ※リリース版のライブラリだけを含むリポジトリ	Disable redploy

- SNAPSHOTリポジトリ



- RELEASEリポジトリ



リポジトリ設定については、下記がとても参考になります。

- ＜Sonatype Nexus で Maven リポジトリを構築しよう＞
  https://weblabo.oscasierra.net/sonatype-nexus-1/

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

❶ build.gradleファイルに下記の記載を行います

❷ config.ymlファイルに下記の記載を行います

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"
    }
}
```

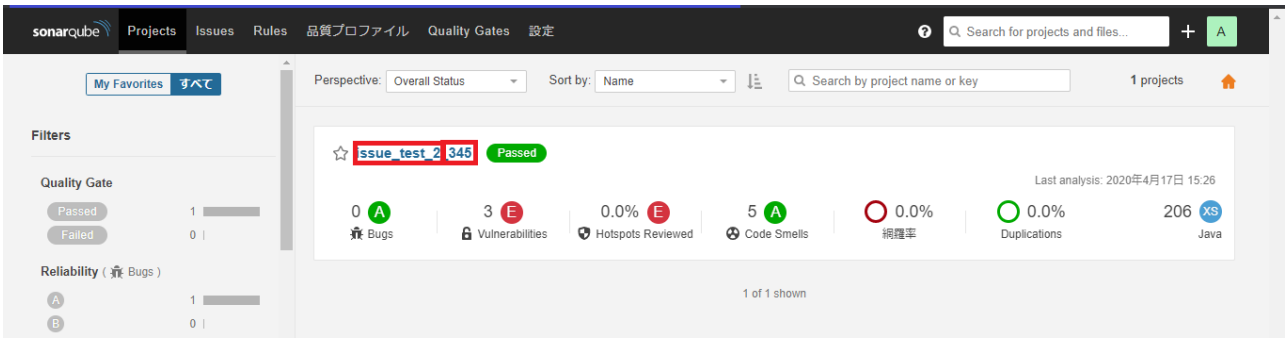
config.yml：sonarqubeタスクの呼び出し

```
##"./gradlew sonarqube"コマンドによるSonerQubeサイトで設定した検証の実施と結果連携を行います。
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 12. 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

❶ build.gradleファイルに下記の記載を行います

❷ config.ymlファイルに下記の記載を行います

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

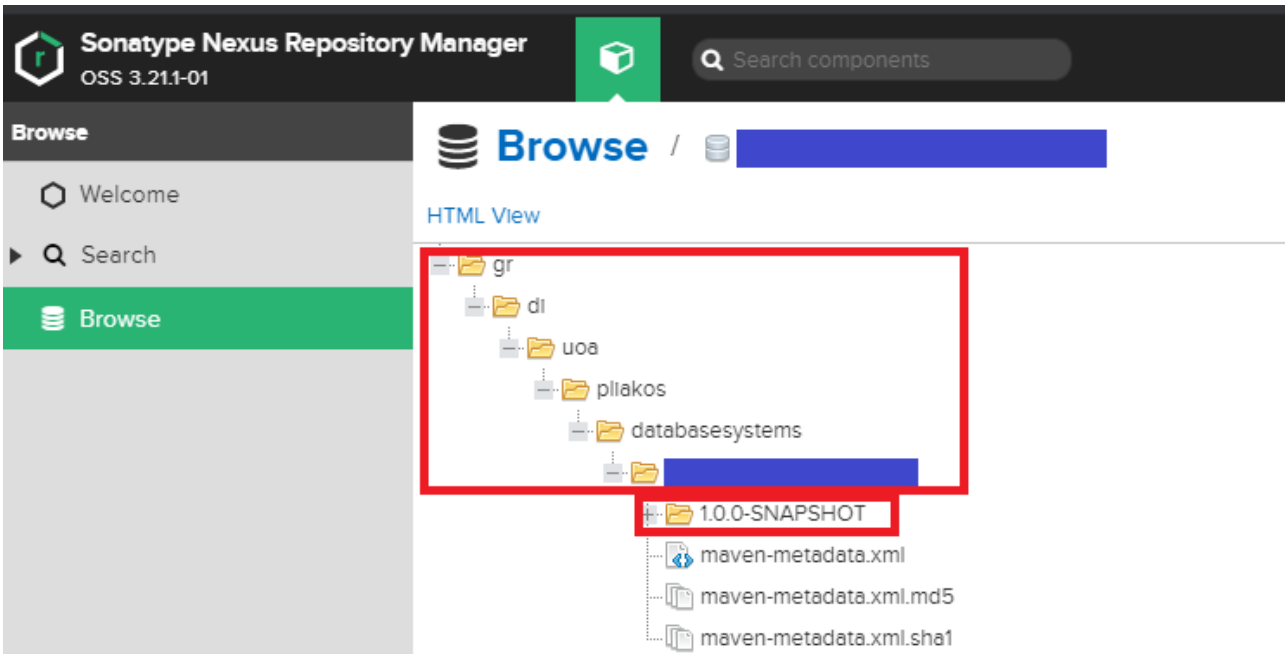
config.yml：publishingタスクの呼び出し

```
##"./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 13. publishingタスクのプロパティ

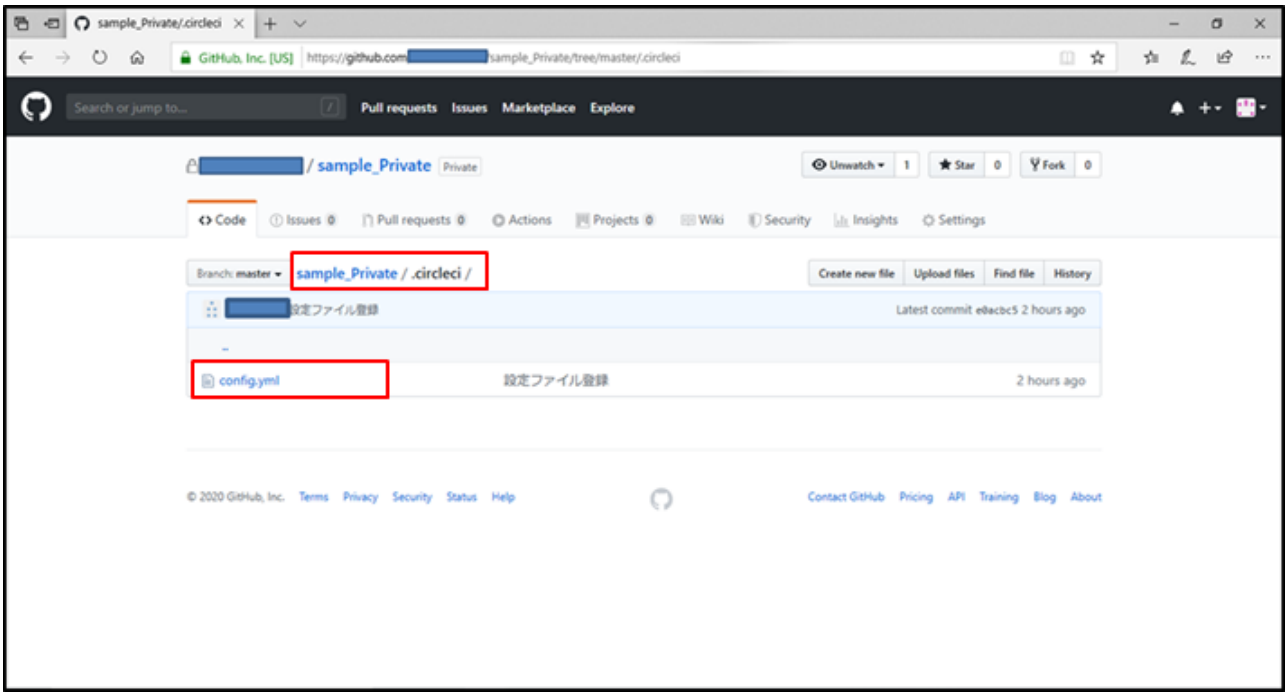
プロパティ	説明	変更要否
groupId	Nexusに登録する際のパッケージ名	PJの設定に併せて変更する ※このガイドラインでは、「group」の値を設定しています
artifactId	Nexusに登録する際のファイル名	PJの設定に併せて変更する ※このガイドラインでは、「archivesBaseName」の値を設定しています
url	接続先SonarQubのURL	PJの設定に併せて変更する ※環境変数の設定については後述します
username	Nexusのログインユーザ名	PJの設定に併せて変更する ※環境変数の設定については後述します
password	Nexusのログインパス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

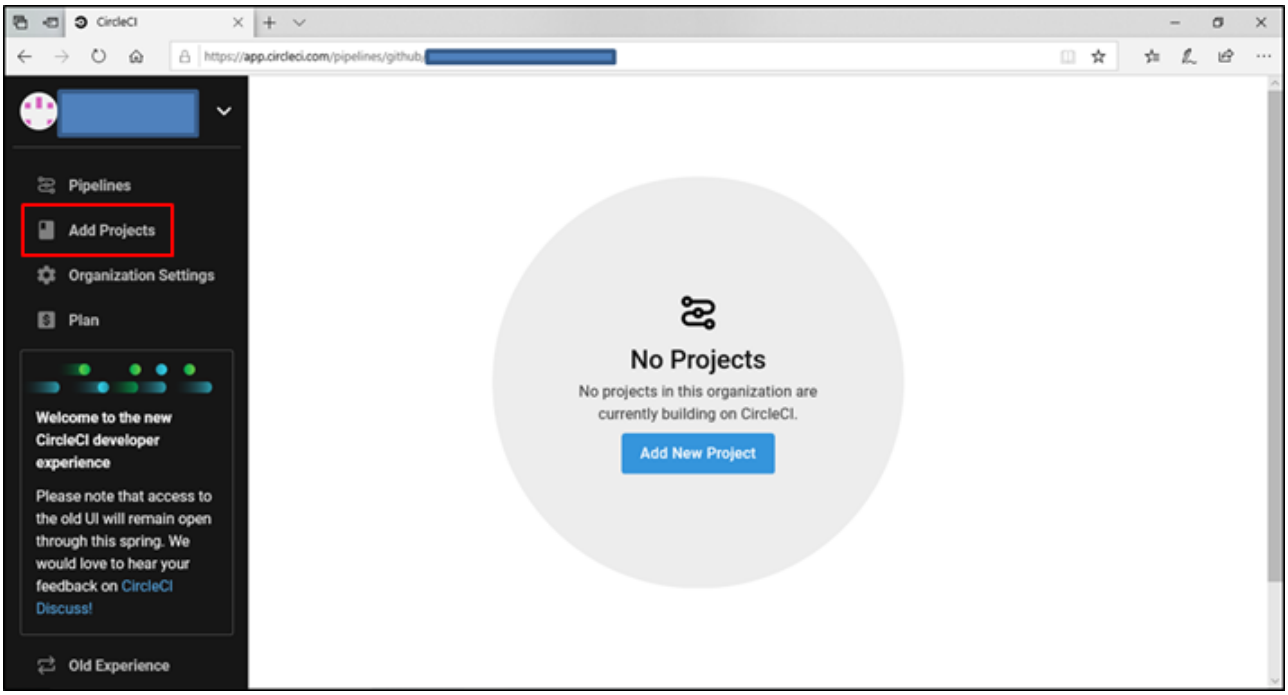
ログイン方法等については、公式ドキュメントを参照

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

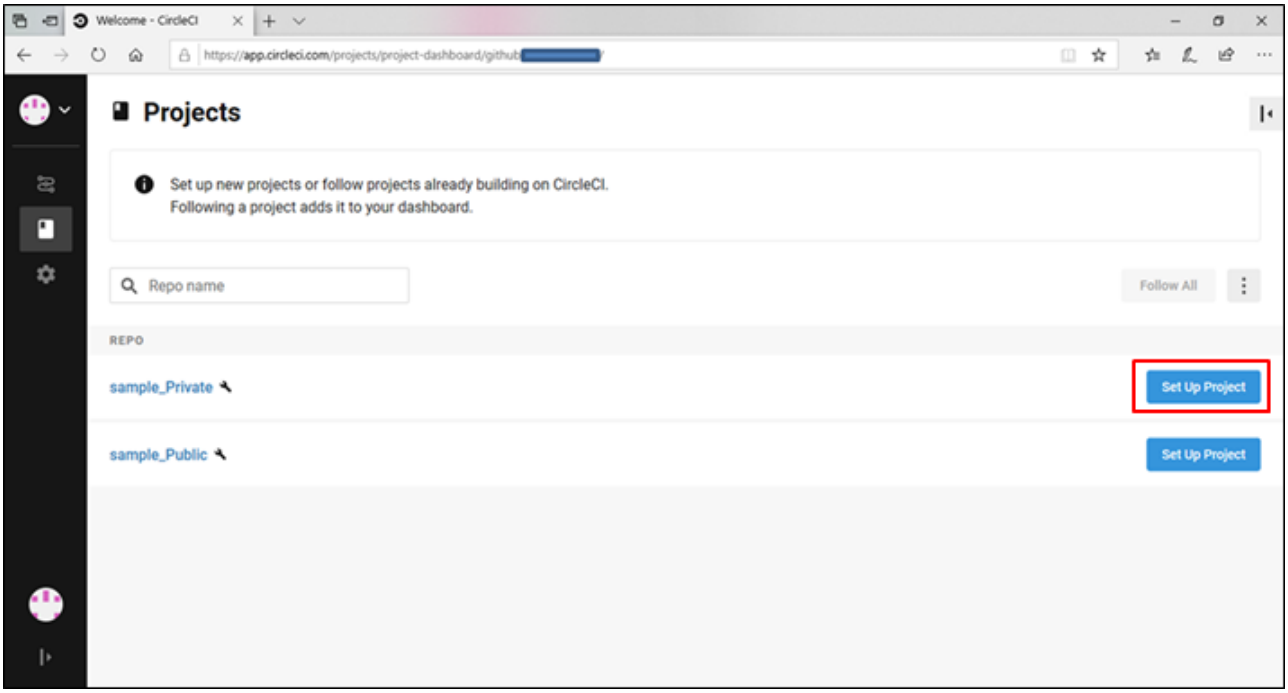
3.5.1. 対象リポジトリの登録

「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。

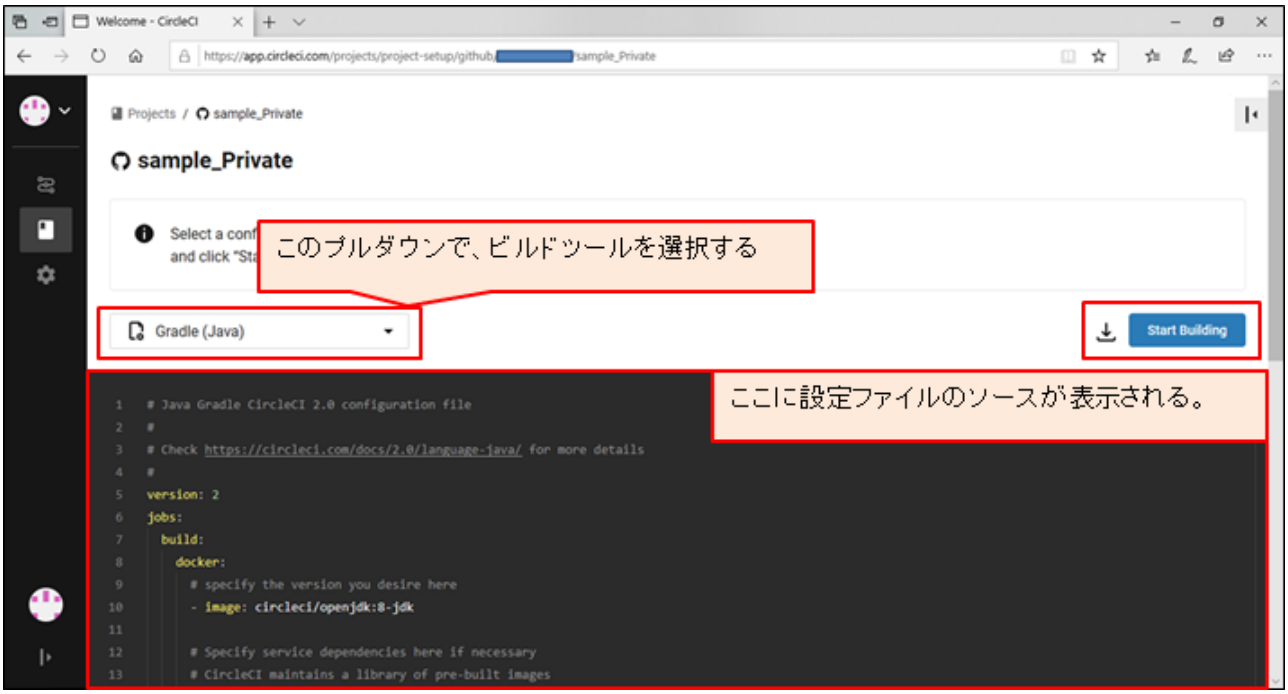
1 ダッシュボード画面より「Add Projects」を選択します。



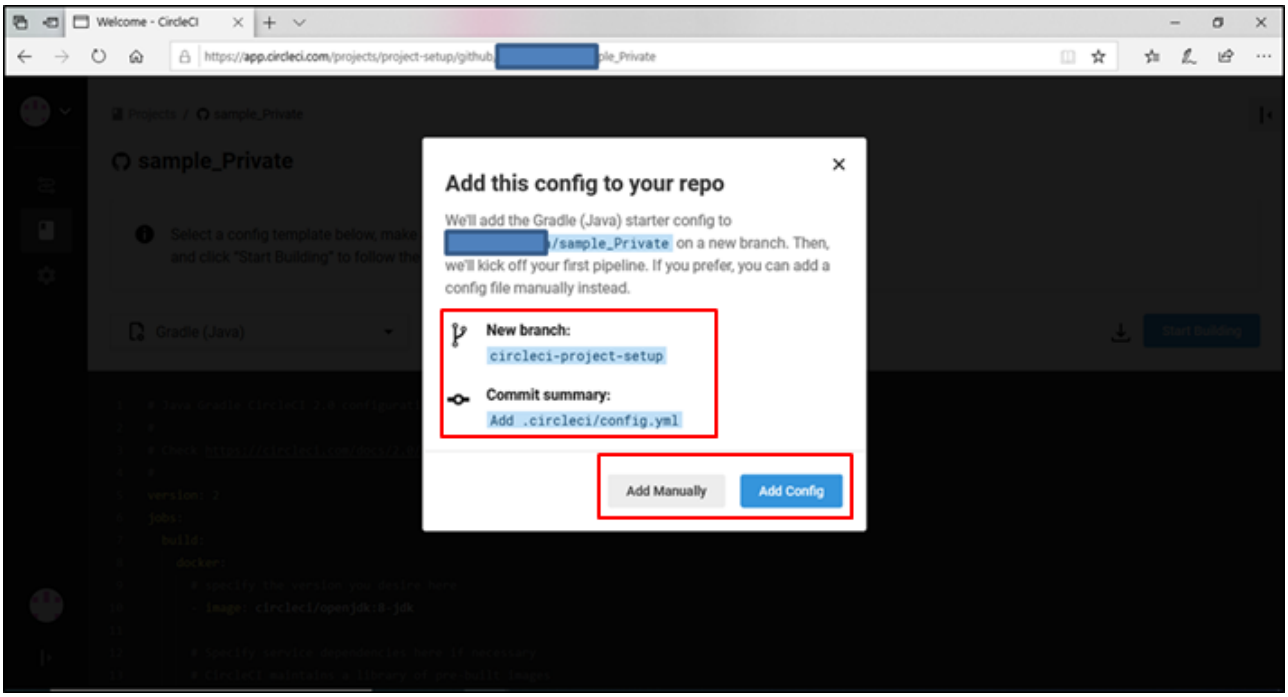
2 GitHub側の公開設定に関わらず、全リポジトリが表示されるので対象とするリポジトリを選択します。



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します。

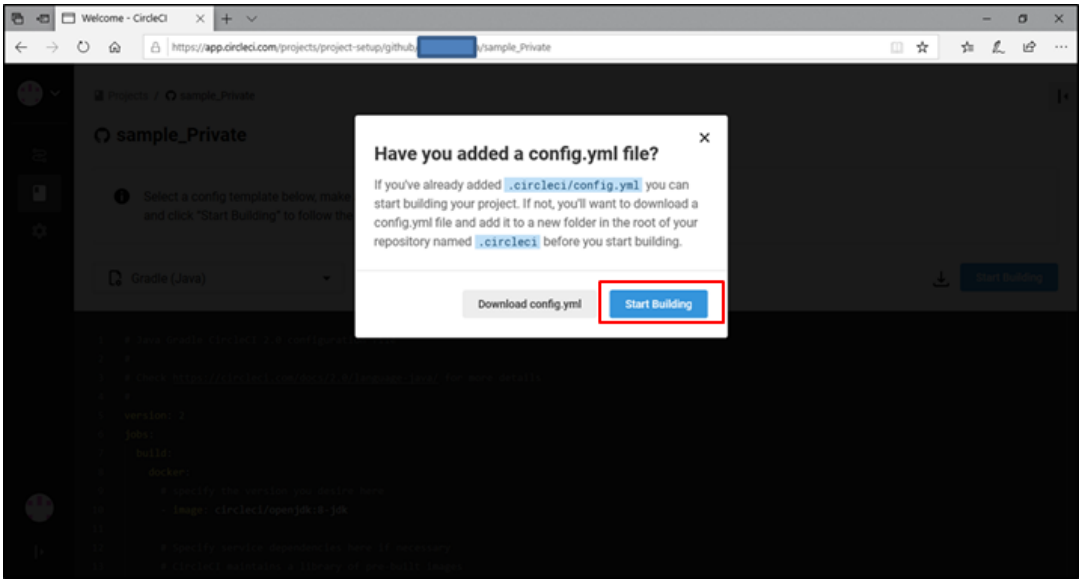


4 「Start Building」を押すと下記のポップアップが表示されるので適宜「Add Manually」か「Add Config」を選択します。

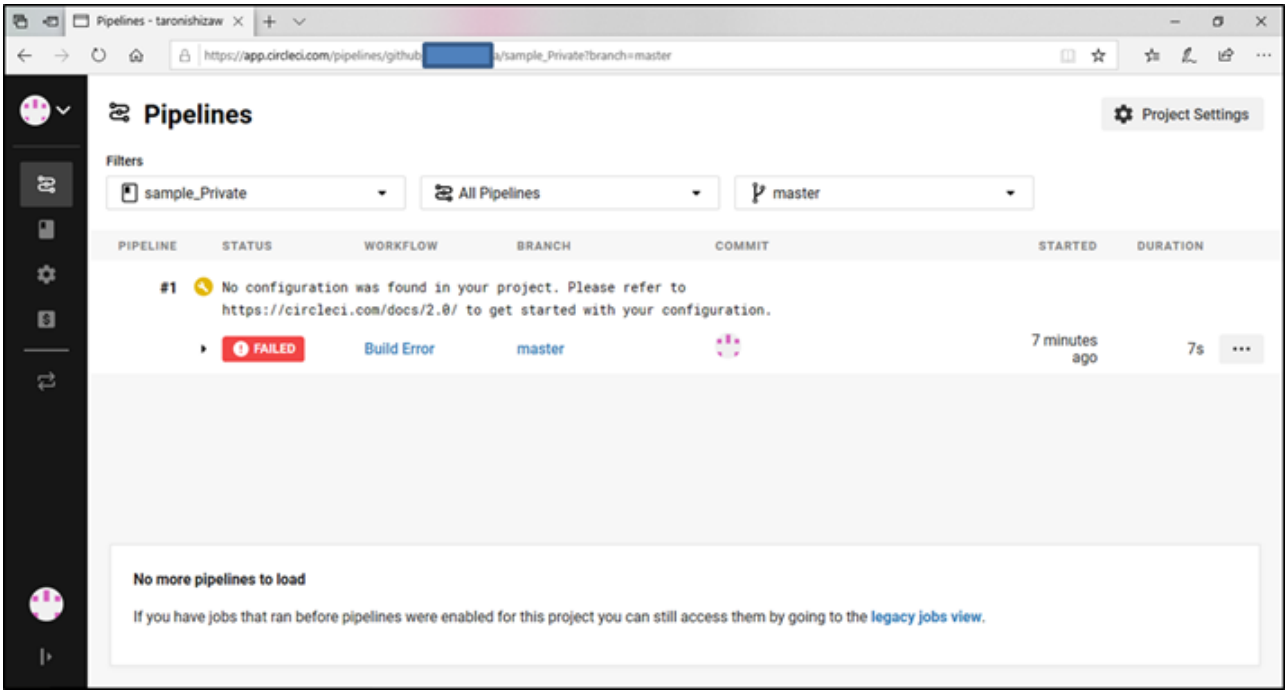


「Add Config」を選択した場合、図に表示されているようにブランチが作成され、.circleci配下に設定ファイル（config.yml）が作成されます。
その後、設定ファイルに基づいて初回のパイプライン実行が行われます。
※設定ファイルの内容は前画面で表示されたデフォルトの内容になるので、初回のパイプライン実行は上手くいかないことが多いです。

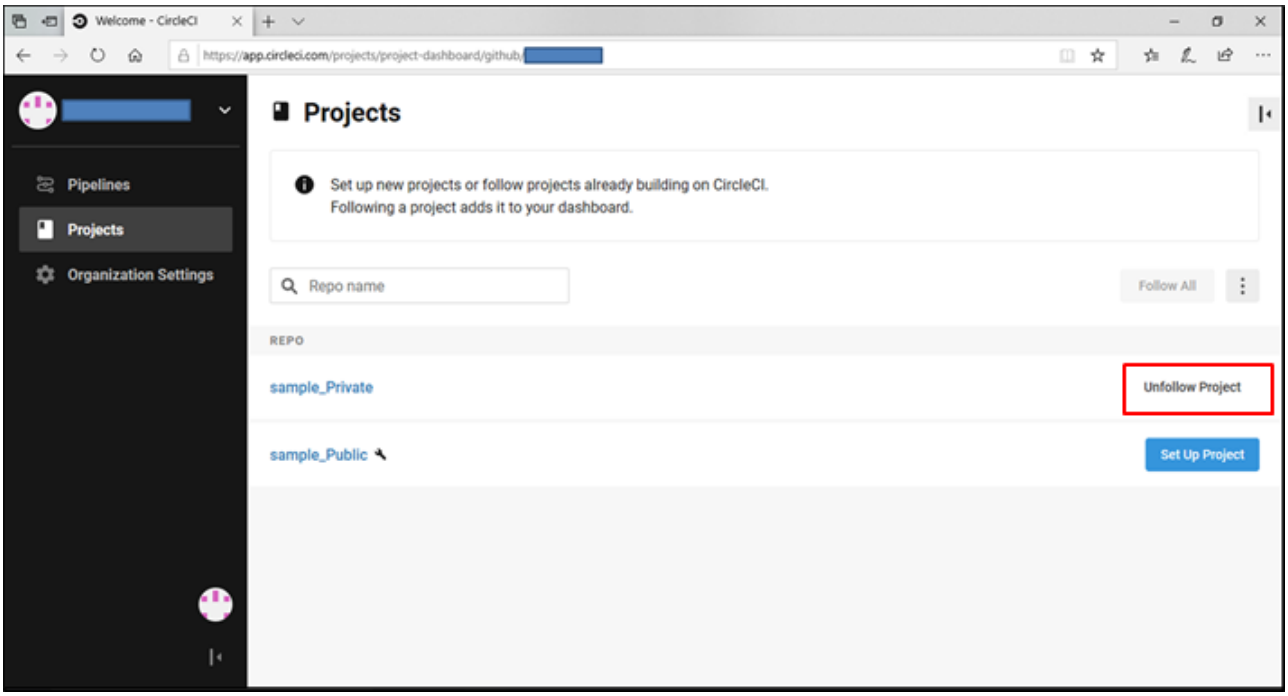
[Add
Manually」を選択した場合、下記のポップアップが表示されるので適宜設定ファイルを.
circleci配下に準備してから「Start Building」をクリックします。
※下図にあるように「Download
config.yml」で設定ファイルをダウンロードすることも可能です。
※設定ファイルの内容は、前画面で表示された内容のものになります。



4 初期パイプライン実行が行われると、下図のように実行結果が表示されます。



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ます。

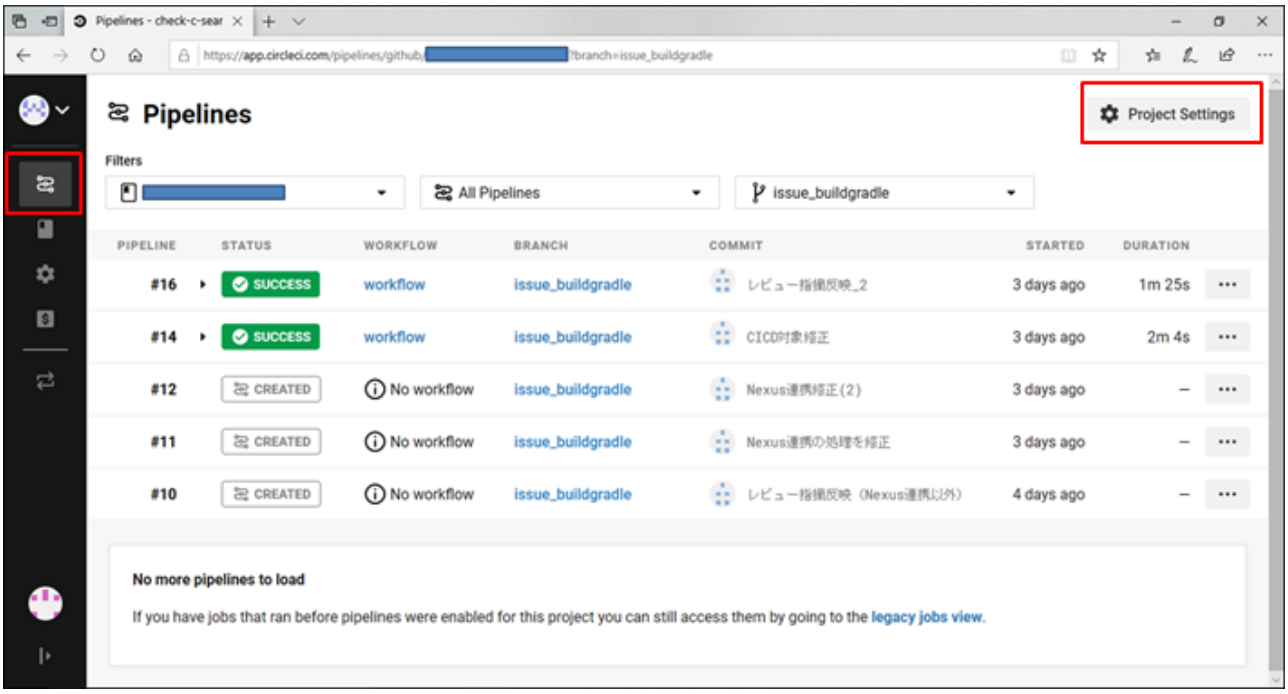


3.5.2. 環境変数の登録

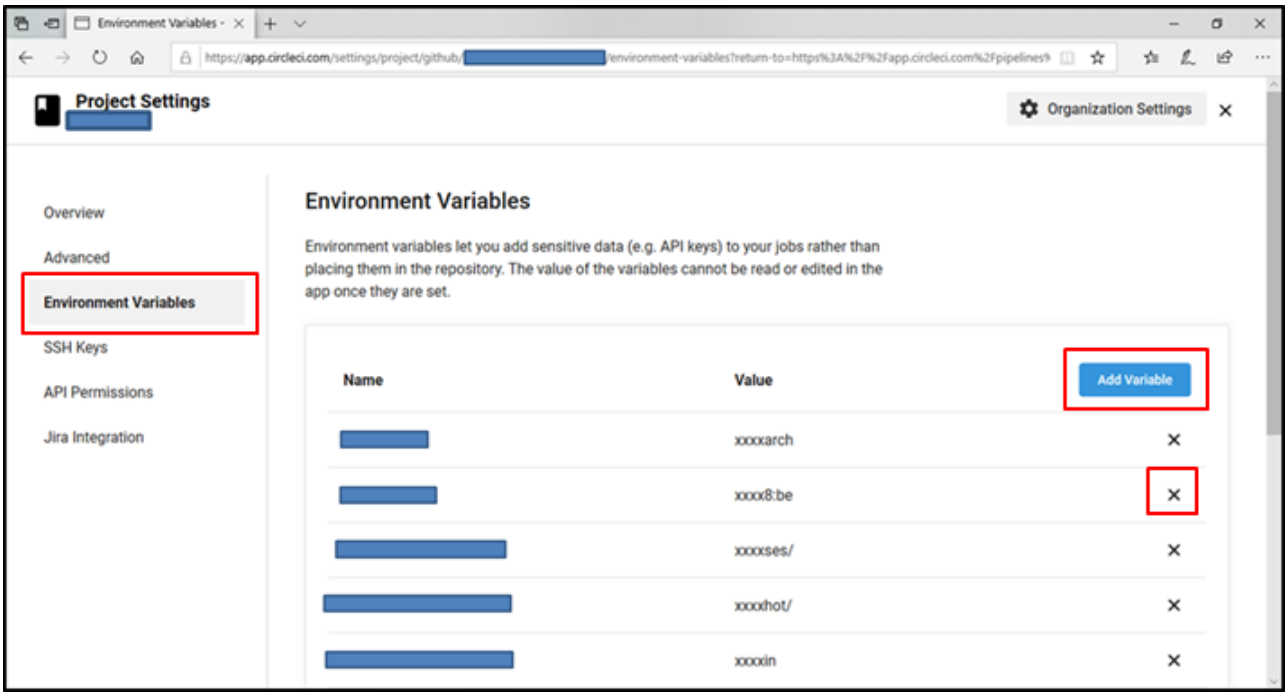
「config.yml」や「build.gradle」に記載した環境変数はパイプライン起動時にCircleCI画面で設定した値に自動的に置き換えられます。

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。

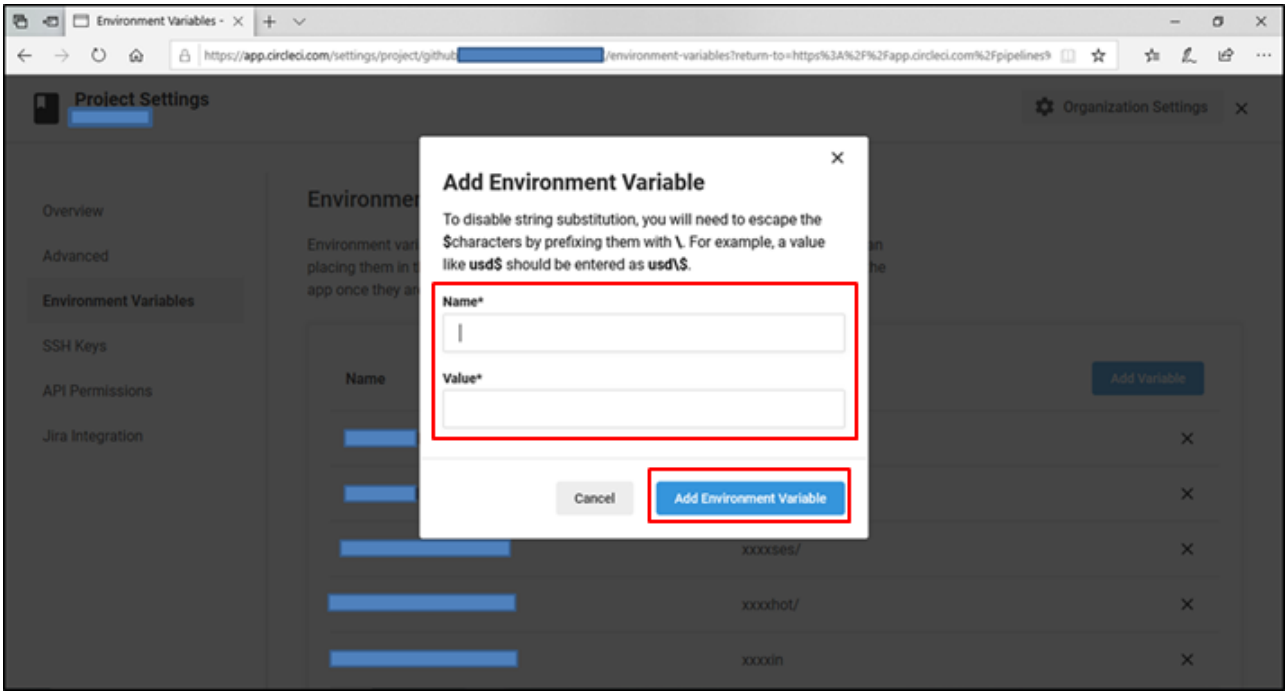
1 対象リポジトリのパイプライン画面より「Project Settings」を選択します。



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ます。
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ます。

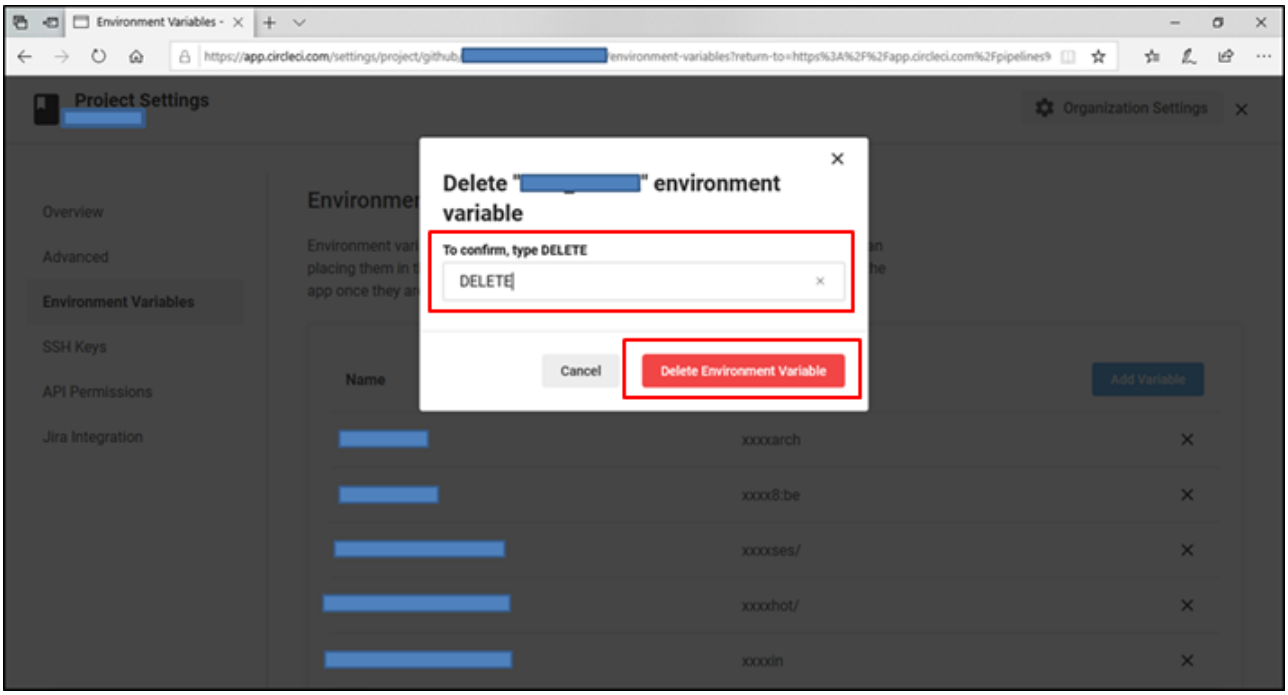


3 「Add Config Variable」を押すと下記のポップアップが表示されるので、任意の環境変数と変数値を入力して「Add Environment Variable」を押します。
　既存の環境変数を入力すると変数値を置き換えることができます。



4 「☒」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」を押します。



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します。
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。

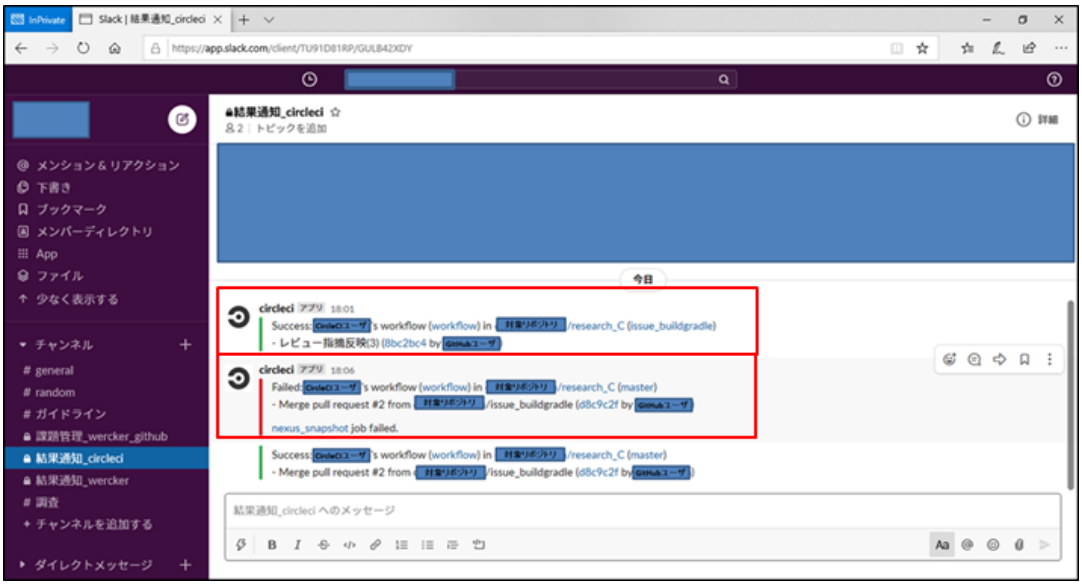
「CircleCIサイト＞環境変数の使い方」
https://circleci.com/docs/2.0/env-vars/#setting-an-environment-variable-in-a-step

3.5.3. 実行結果の発報

パイプライン事項結果をPJのチャットツール（このガイドラインではSlack）に連携します。

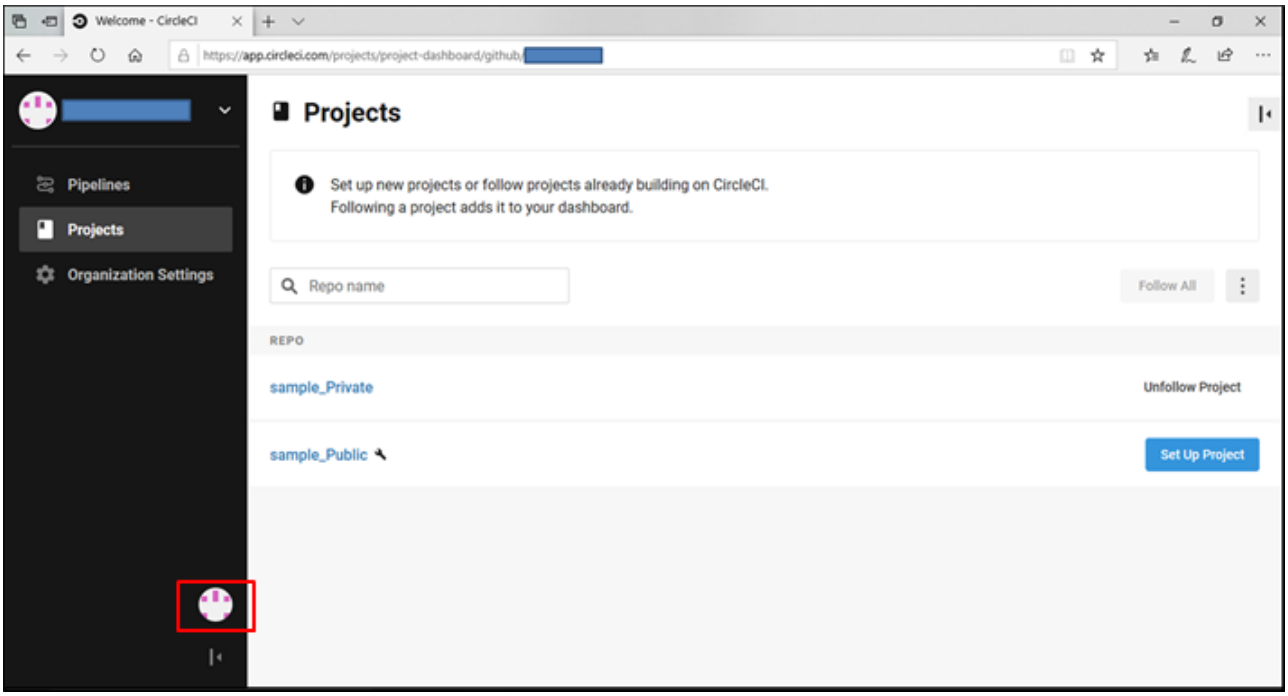
1 Slackの「App」よりCircleCIアプリを追加して、「セットアップの手順」に従ってCircleCIに設定します。

実行結果は、下記の方式でCircleC及びGitHubへのリンク付きで連携されます。
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」は誰かを通知します。
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」は誰かを通知します。

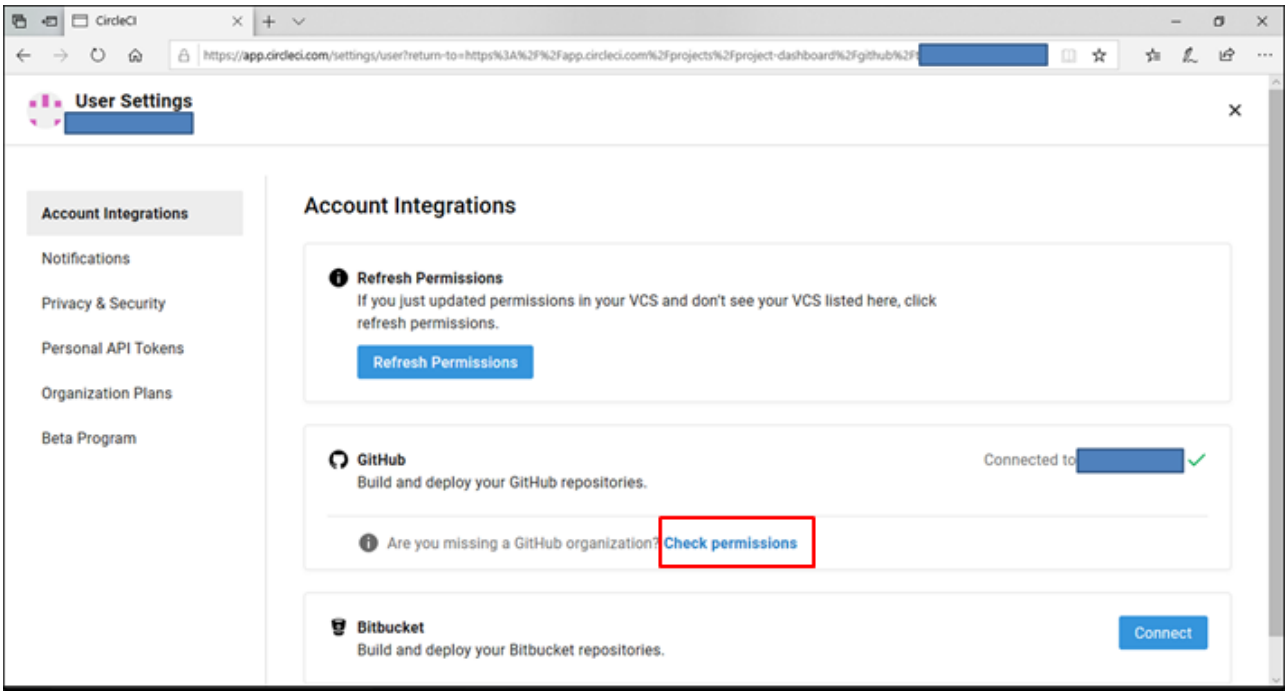


3.5.4. Organizationアカウントのrepositoryを参照するための設定

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」画面より設定を行います。

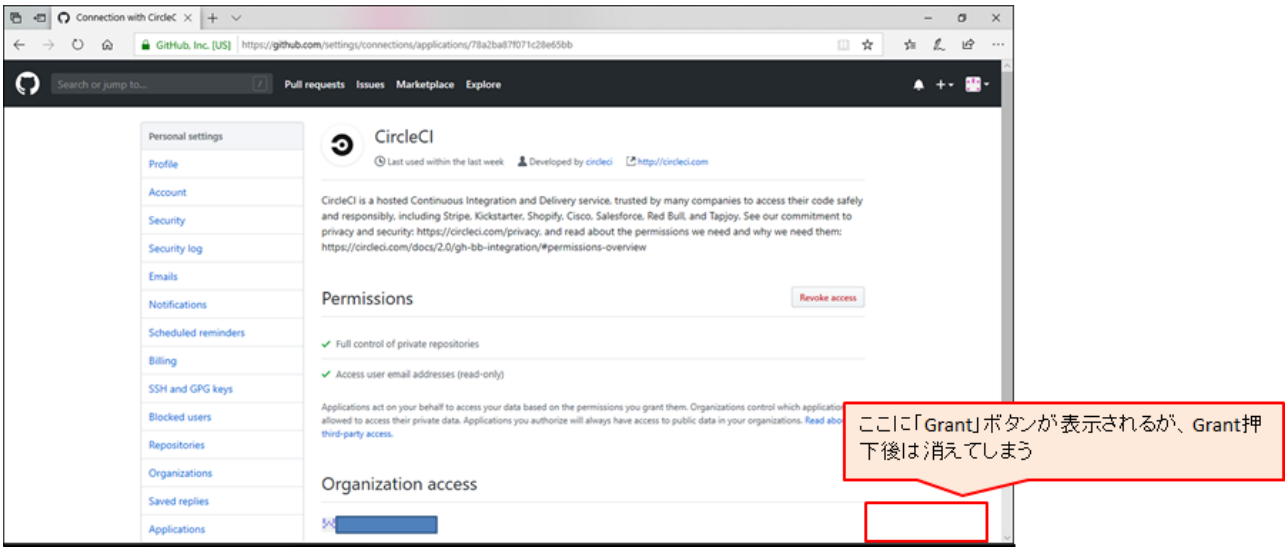


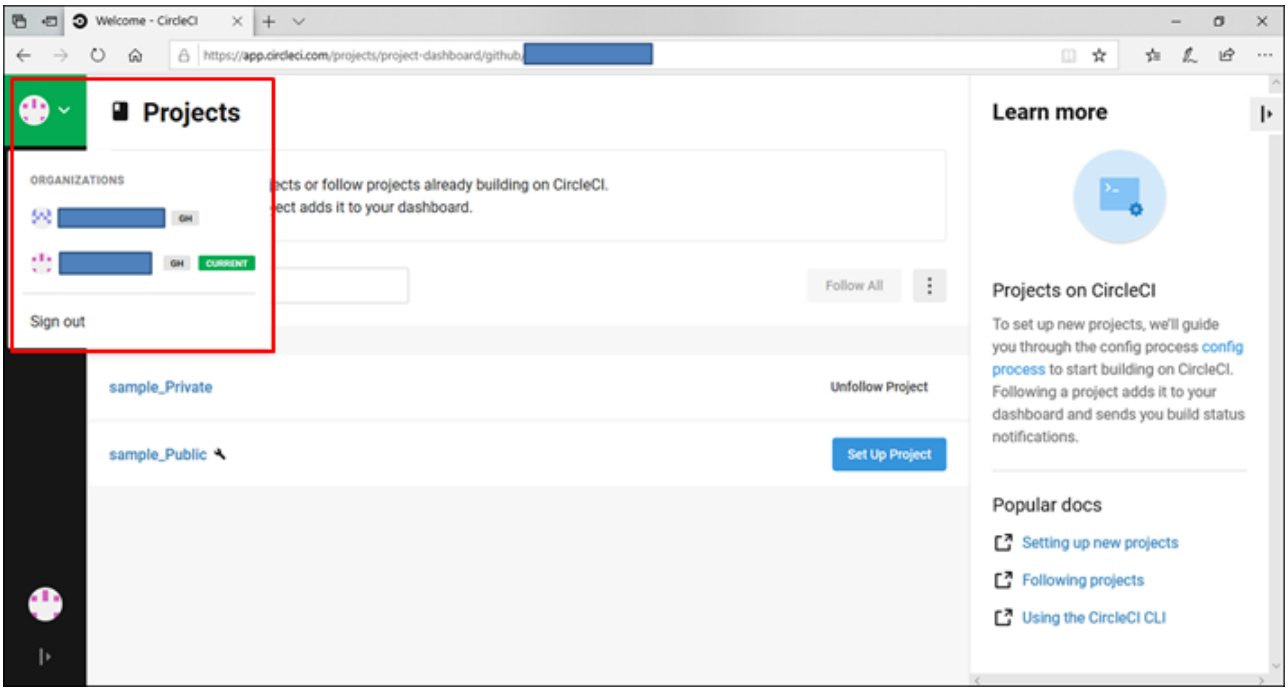
2 「User Setting」画面で「Account Integrations」→GitHubの「Check permissions」を選択して、GitHub画面を開きます。



3 画面下の「Organization access」に自身が所属しているOrganizationが表示されるので、その内容を確認して「Grant」ボタンを押します。

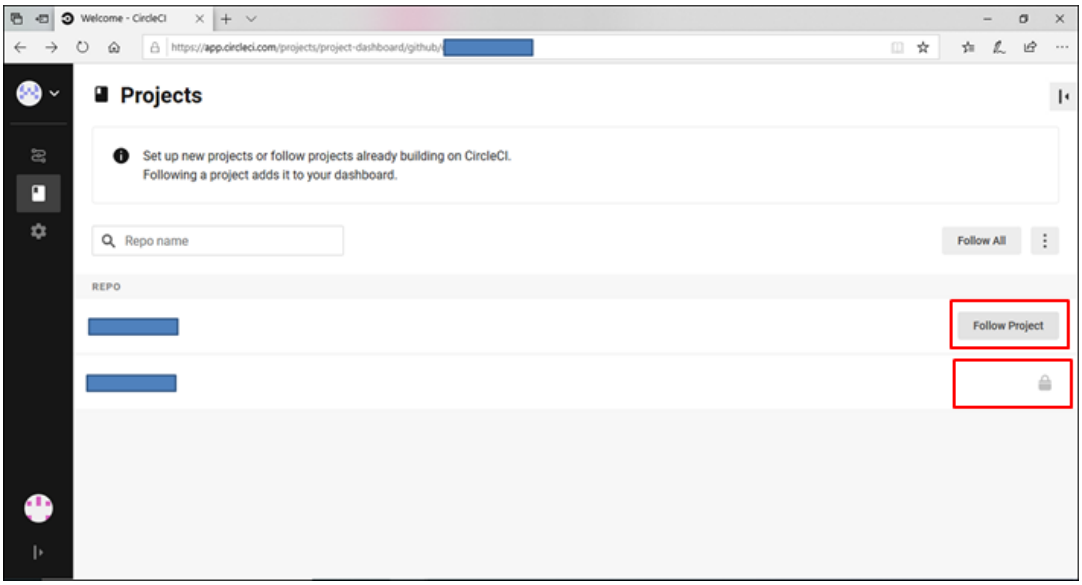
　この操作でOrganization所有のリポジトリを参照できるようになります。

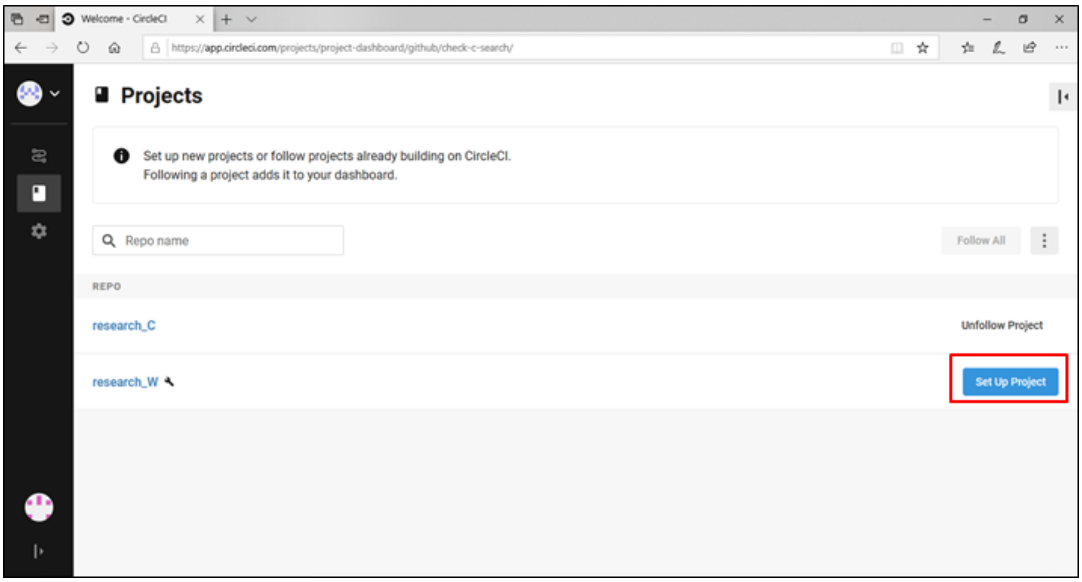




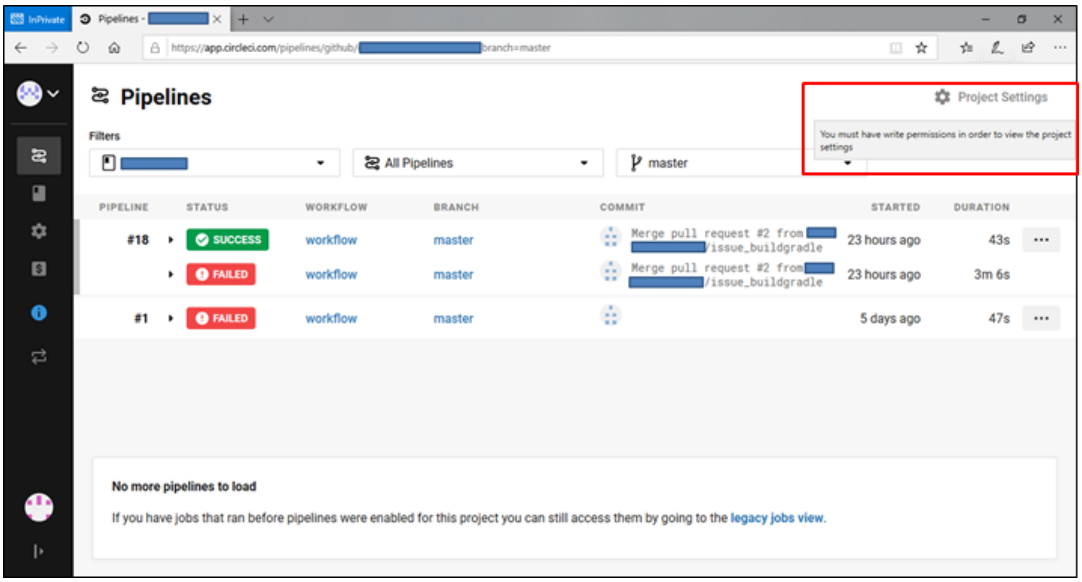
Organization所有のリポジトリについては、Organization内での自アカウントの権限によって操作が制限されます。

- 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」を行うことができます。
  ※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ます。
  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」を行うことができます。
  ※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）です
  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します。





- 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可能です。