トラディショナルな Java 開発者も クラウドネイティブへ - Cloud Run でアプリ開発 -

Google Cloud Japan
Application Modernization Specialist
柳原 伸弥

本日のセッションのポイント

- Cloud Run がどのようなサービスか分かって頂く
- Cloud Run で動かすアプリケーション開発の流れを分かって頂く
- Cloud Run を利用したモダナイゼーションのサンプル開発をご覧頂く

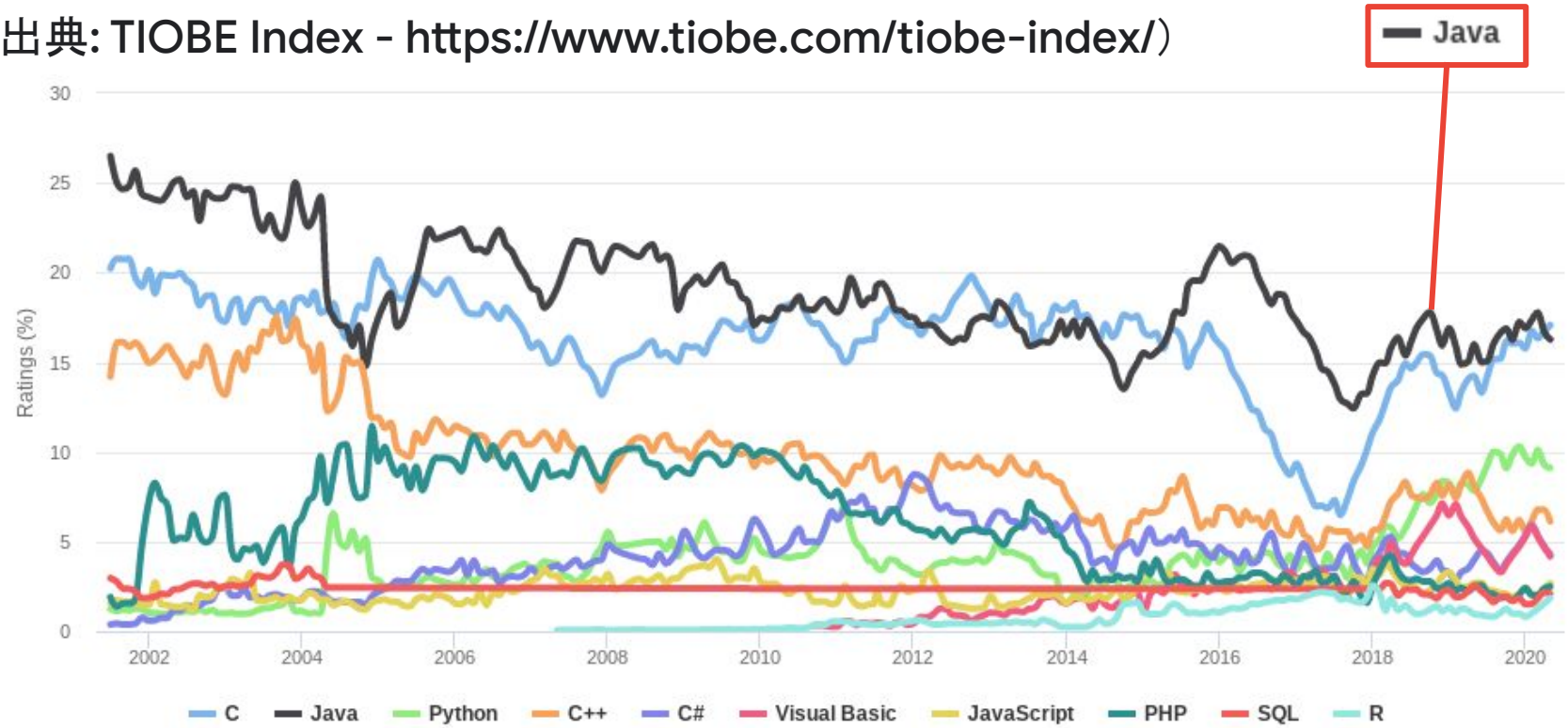
- Cloud Run を使ってみたいと思って頂く

本日のセッションの前提

- 対象とするアプリケーションは **Java 言語**によるもの
- 対象とするプラットフォームは Cloud Run（フルマネージド）

プログラム言語 人気ランキング 2020 年 5 月版

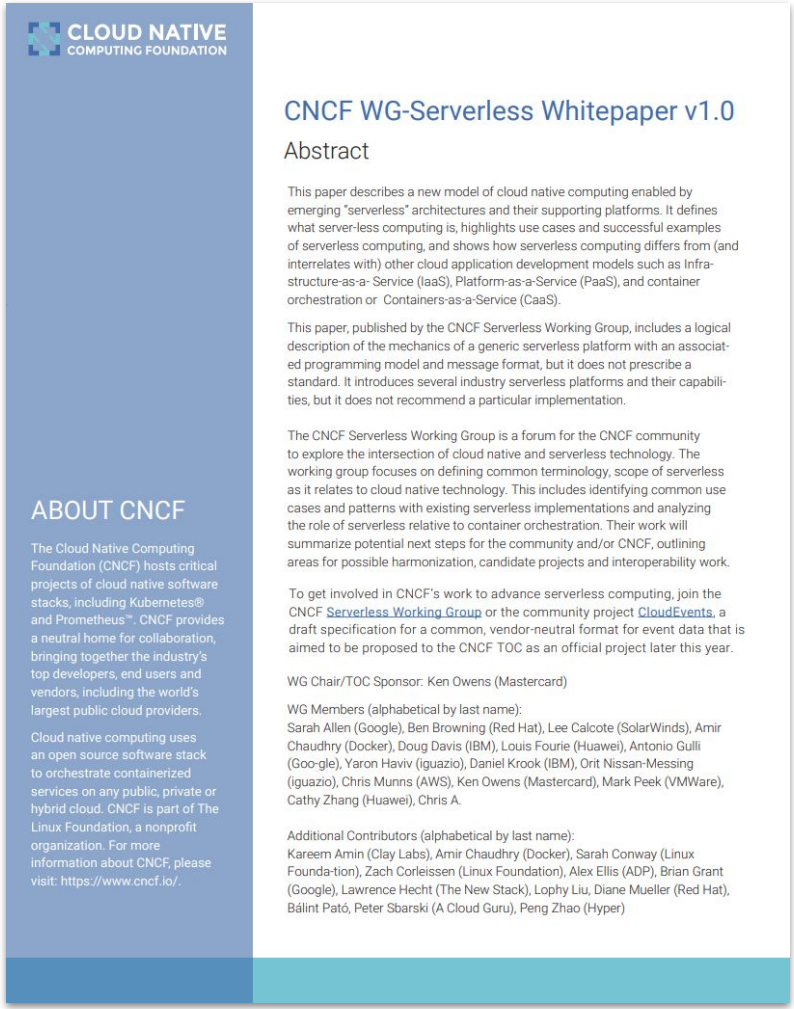
（出典: TIOBE Index - https://www.tiobe.com/tiobe-index/）



Google Cloud

What is
サーバーレス？

Google Cloud



CLOUD NATIVE COMPUTING FOUNDATION

CNCF WG-Serverless Whitepaper v1.0

Abstract

This paper describes a new model of cloud native computing enabled by emerging "serverless" architectures and their supporting platforms. It defines what server-less computing is, highlights use cases and successful examples of serverless computing, and shows how serverless computing differs from (and interrelates with) other cloud application development models such as Infra-structure-as-a- Service (IaaS), Platform-as-a-Service (PaaS), and container orchestration or Containers-as-a-Service (CaaS).

This paper, published by the CNCF Serverless Working Group, includes a logical description of the mechanics of a generic serverless platform with an associat-ed programming model and message format, but it does not prescribe a standard. It introduces several industry serverless platforms and their capabili-ties, but it does not recommend a particular implementation.

The CNCF Serverless Working Group is a forum for the CNCF community to explore the intersection of cloud native and serverless technology. The working group focuses on defining common terminology, scope of serverless as it relates to cloud native technology. This includes identifying common use cases and patterns with existing serverless implementations and analyzing the role of serverless relative to container orchestration. Their work will summarize potential next steps for the community and/or CNCF, outlining areas for possible harmonization, candidate projects and interoperability work.

To get involved in CNCF's work to advance serverless computing, join the CNCF Serverless Working Group or the community project CloudEvents, a draft specification for a common, vendor-neutral format for event data that is aimed to be proposed to the CNCF TOC as an official project later this year.

WG Chair/TOC Sponsor: Ken Owens (Mastercard)

WG Members (alphabetical by last name):
Sarah Allen (Google), Ben Browning (Red Hat), Lee Calcote (SolarWinds), Amir Chaudhry (Docker), Doug Davis (IBM), Louis Fourie (Huawei), Antonio Gulli (Goo-gle), Yaron Haviv (iguazio), Daniel Krook (IBM), Orit Nissan-Messing (iguazio), Chris Munns (AWS), Ken Owens (Mastercard), Mark Peek (VMWare), Cathy Zhang (Huawei), Chris A.

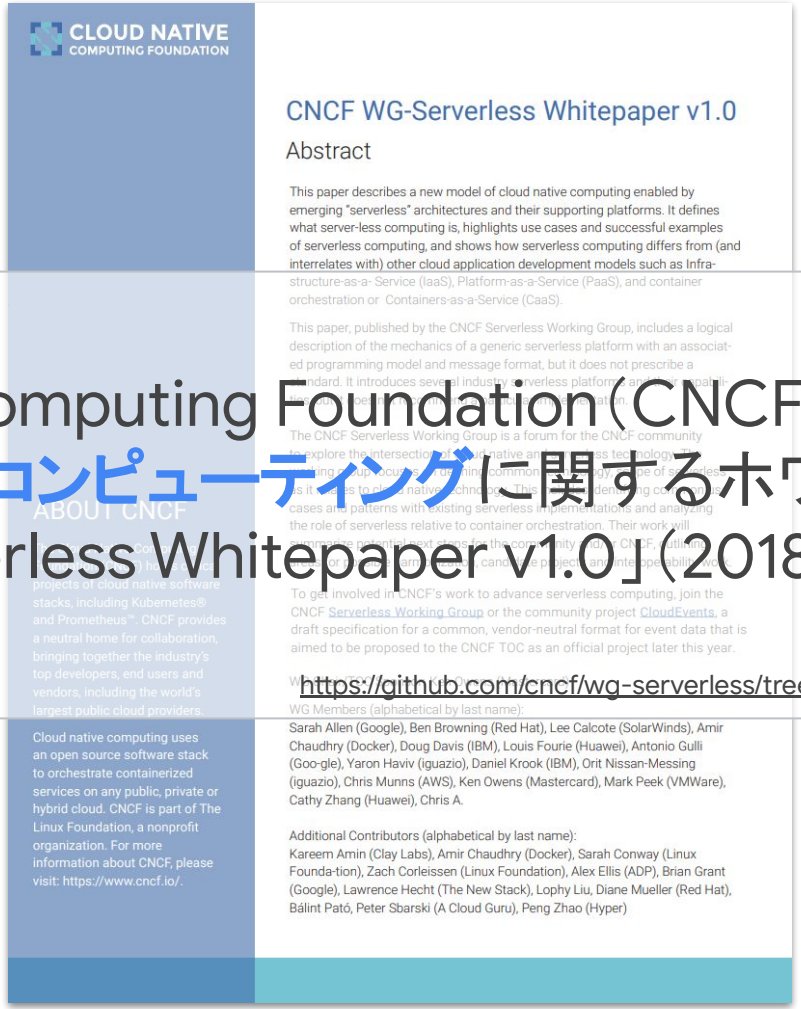
Additional Contributors (alphabetical by last name):
Kareem Amin (Clay Labs), Amir Chaudhry (Docker), Sarah Conway (Linux Founda-tion), Zach Corleissen (Linux Foundation), Alex Ellis (ADP), Brian Grant (Google), Lawrence Hecht (The New Stack), Lophy Liu, Diane Mueller (Red Hat), Bálint Pató, Peter Sbarski (A Cloud Guru), Peng Zhao (Hyper)

ABOUT CNCF

The Cloud Native Computing Foundation (CNCF) hosts critical projects of cloud native software stacks, including Kubernetes® and Prometheus™. CNCF provides a neutral home for collaboration, bringing together the industry's top developers, end users and vendors, including the world's largest public cloud providers.

Cloud native computing uses an open source software stack to orchestrate containerized services on any public, private or hybrid cloud. CNCF is part of The Linux Foundation, a nonprofit organization. For more information about CNCF, please visit: https://www.cncf.io/.

Google Cloud



Cloud Native Computing Foundation（CNCF）が公開している **サーバーレスコンピューティング**に関するホワイトペーパー「CNCF Serverless Whitepaper v1.0」（2018 年 2 月公開）

https://github.com/cncf/wg-serverless/tree/master/whitepapers/serverless-overview

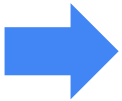
Google Cloud

サーバーレス コンピューティングの定義

サーバーレスコンピューティングとは、
サーバー管理を必要としないアプリケーションの構築と実行の概念

(出典) CNCF Serverless Whitepaper v1.0
https://github.com/cncf/wg-serverless/tree/master/whitepapers/serverless-overview

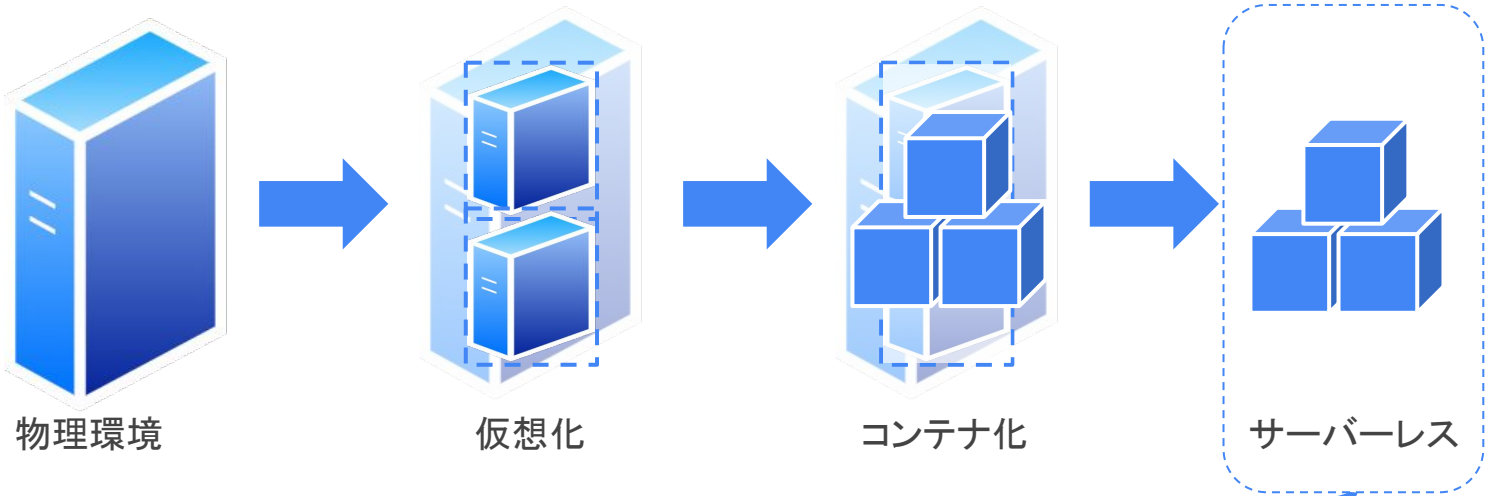
- コードをホストして実行するためにサーバーを使用しない
- 運用エンジニアがもはや必要とされない



利用者が、サーバのプロビジョニング、メンテナンス、アップデート、スケーリング、キャパシティプランニングに時間とリソースを費やす必要ない

Google Cloud

サーバーレス コンピューティングの定義

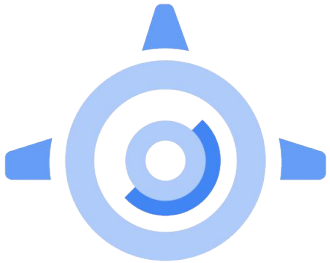


特長

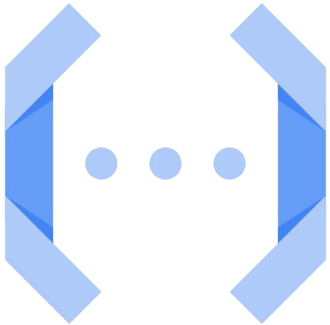
- インフラの運用管理不要
- クラウド管理セキュリティ
- 使用分のみの課金

Google Cloud

Google Cloud のサーバーレス コンピューティング
（アプリケーション実行環境）



App Engine



Cloud Function

Cloud Run
Cloud Run for Anthos

3 種類のサーバーレスなアプリケーションプラットフォーム
何に対する "サーバーレス" ？

アプリケーション

関数

コンテナ

Google Cloud

Cloud Run

Knative ベースの
サーバーレスプラットフォーム

本セッションでは、サーバーレスコンピュートの中から
Cloud Run について説明を行います

Google Cloud

What is Cloud Run ?

Google Cloud

What is Knative ?

Knative

Kubernetes 上でサーバーレス環境を提供

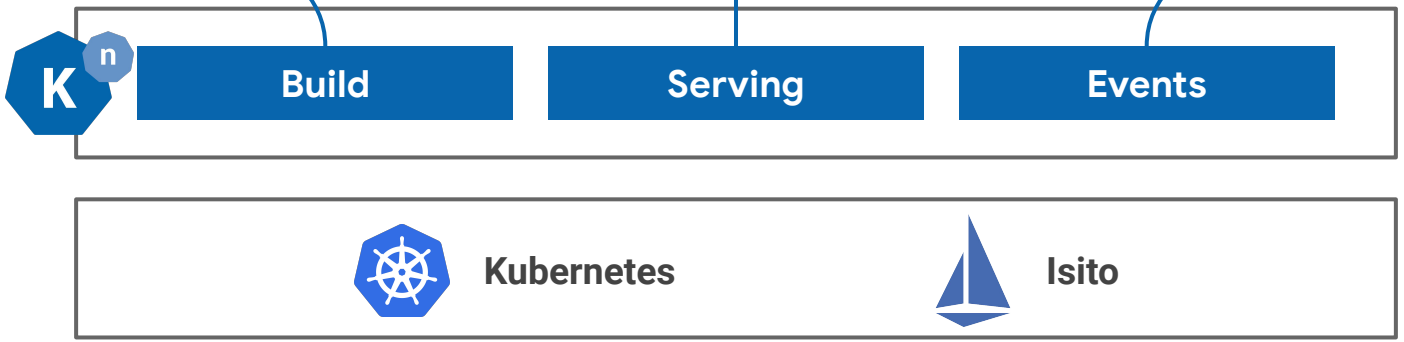
- オープンソース ソフトウェア
  - Google が 50 社以上の企業と共に開発
- ビルド・デプロイ・管理
  - オンプレミス/クラウドに関係なくクラウドネイティブアプリケーションの実行や管理
- Kubernetes のリソースを抽象化
  - よりシンプルに開発
  - よりコーディングに集中

Knative 概要

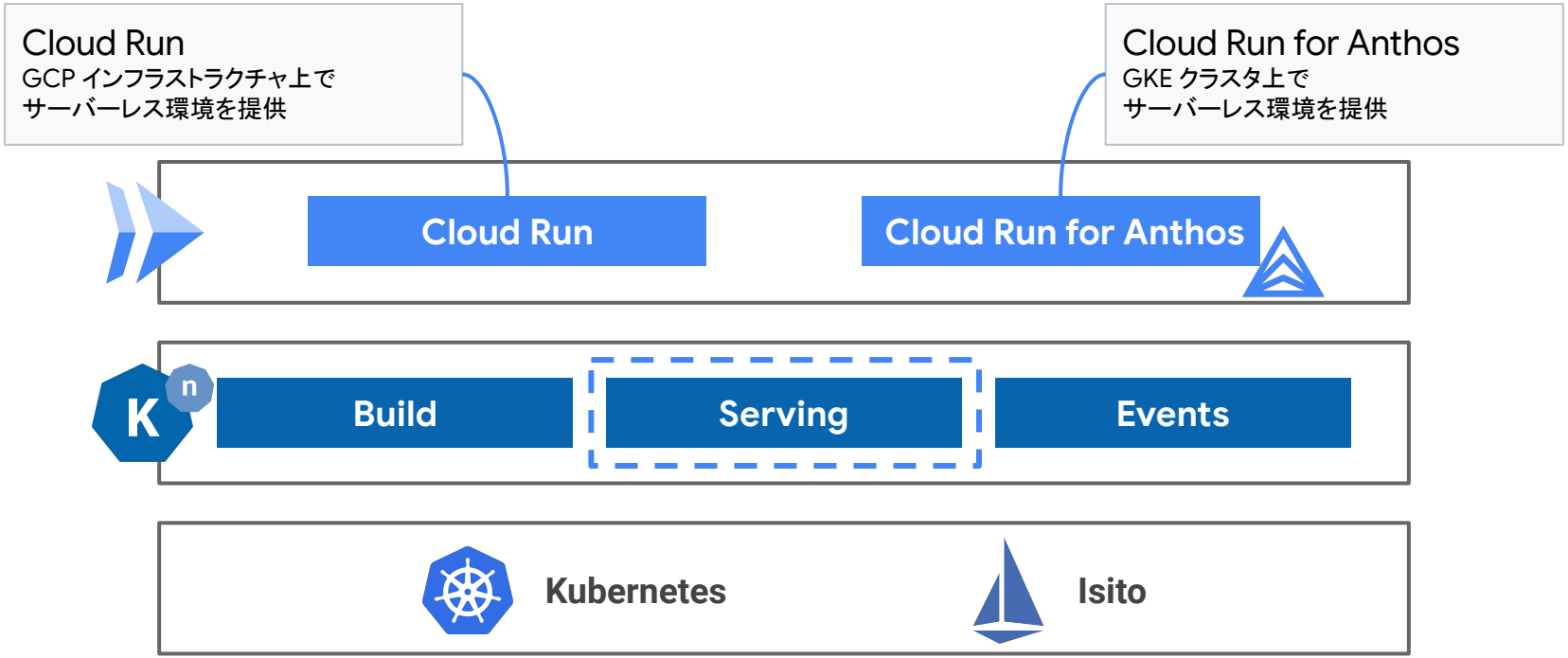
Build
ソースコードをコンテナへビルド
テンプレート化による再利用

Serving
コンテナのデプロイ
自動スケーリング (0 to N)
ネットワーク設定

Event
イベントの発行元・受け手の抽象化
イベントドリブンアーキテクチャのサポート



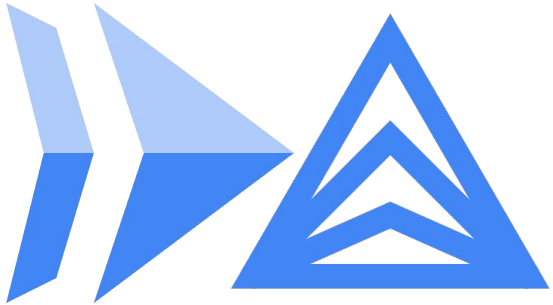
Knative と Cloud Run



2 つの Cloud Run プラットフォーム

Cloud Run（フルマネージド）

基盤を完全に隠蔽



Cloud Run for Anthos

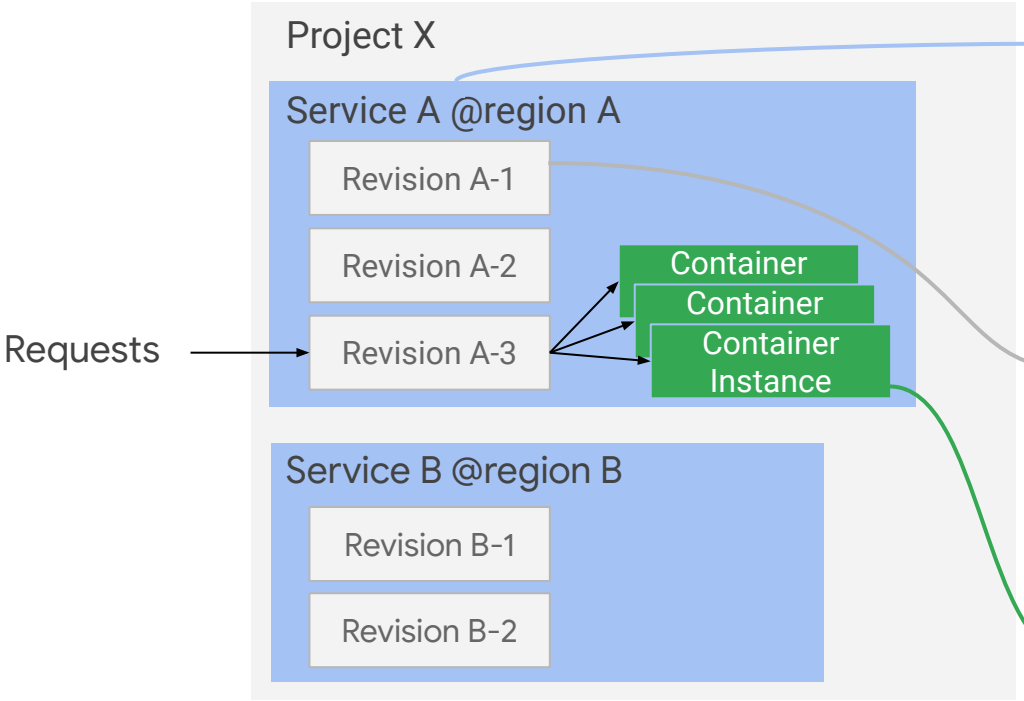
Kubernetes とサーバーレスの利点を享受

従量課金	**課金**	Anthos ライセンス費用 + GKE ノード費用
CPU: 1 vCPU ～ 2 vCPU メモリ: 128 MB ～ 2 GB	**マシンタイプ**	GKEのノードで利用可能なCPU/Memory サイズを指定出来る
デフォルト 1000インスタンス までのスケール(追加可能)	**オートスケール**	Anthos GKE クラスタの キャパシティに依存

Google Cloud

本日のセッションはこちらを説明

Cloud Run (Knative) のリソースモデル



Service

Cloud Run の主リソース
Service 毎に Endpoint を提供
自動で設定される a.run.app ドメイン、
もしくはカスタム ドメインが選択可能

Revision

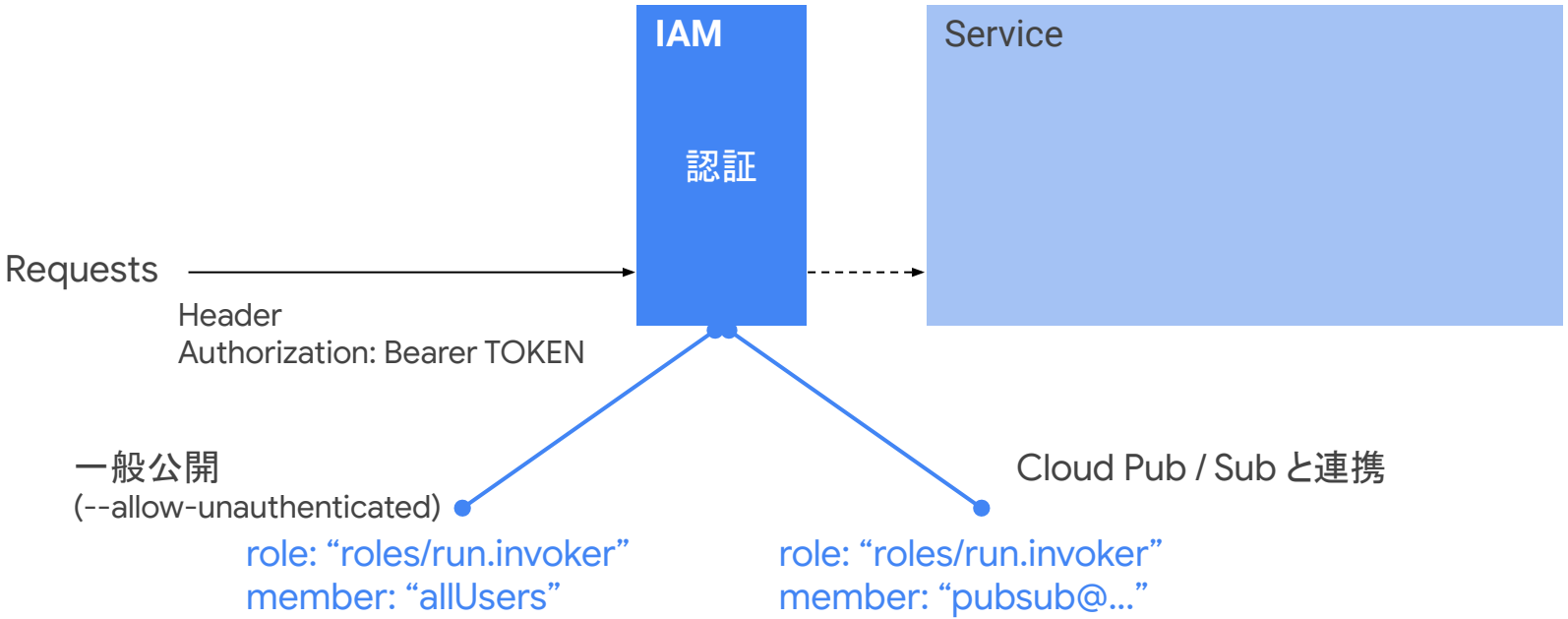
デプロイするごとに生成される
コンテナ イメージとデプロイ時に指定される
環境変数やパラメーターから構成される

Container Instance

実際にリクエストを受けるコンテナ、
リクエスト の数に応じて自動的にスケール

Google Cloud

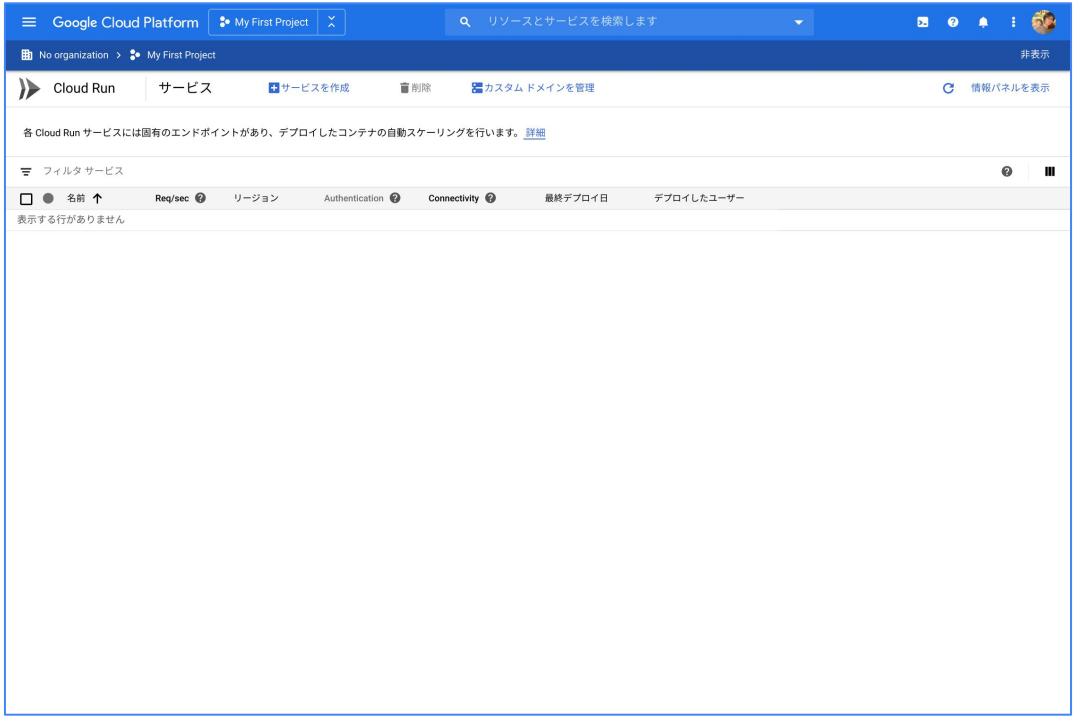
Cloud Run のアクセス認証



Google Cloud

サンプル コンテナの デプロイ

手順 1. Cloud Run に移動

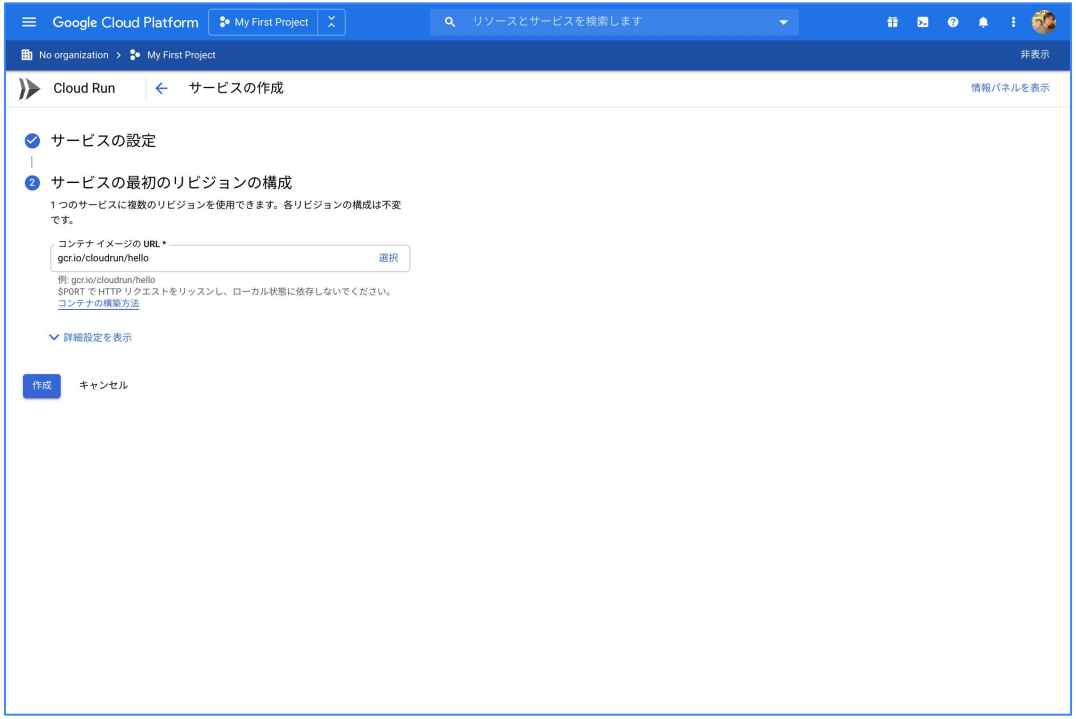


Google Cloud

手順 2. サービスの設定

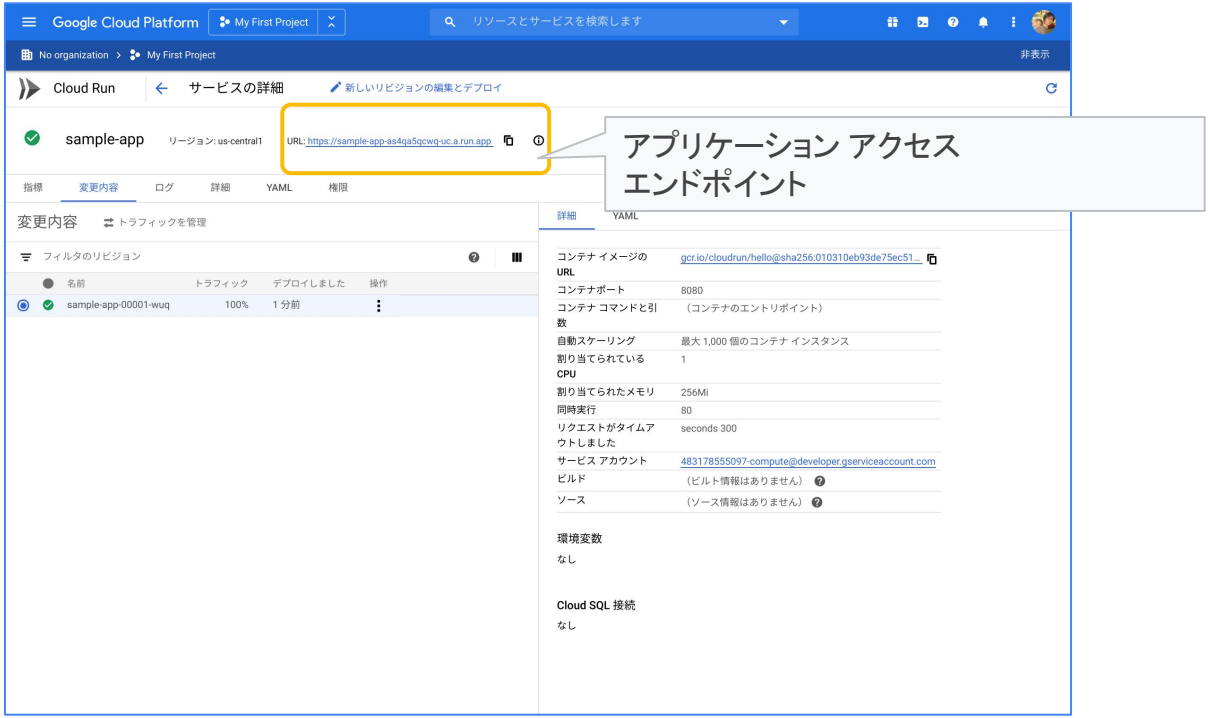
Google Cloud

手順 3. 最初のリビジョンの構成

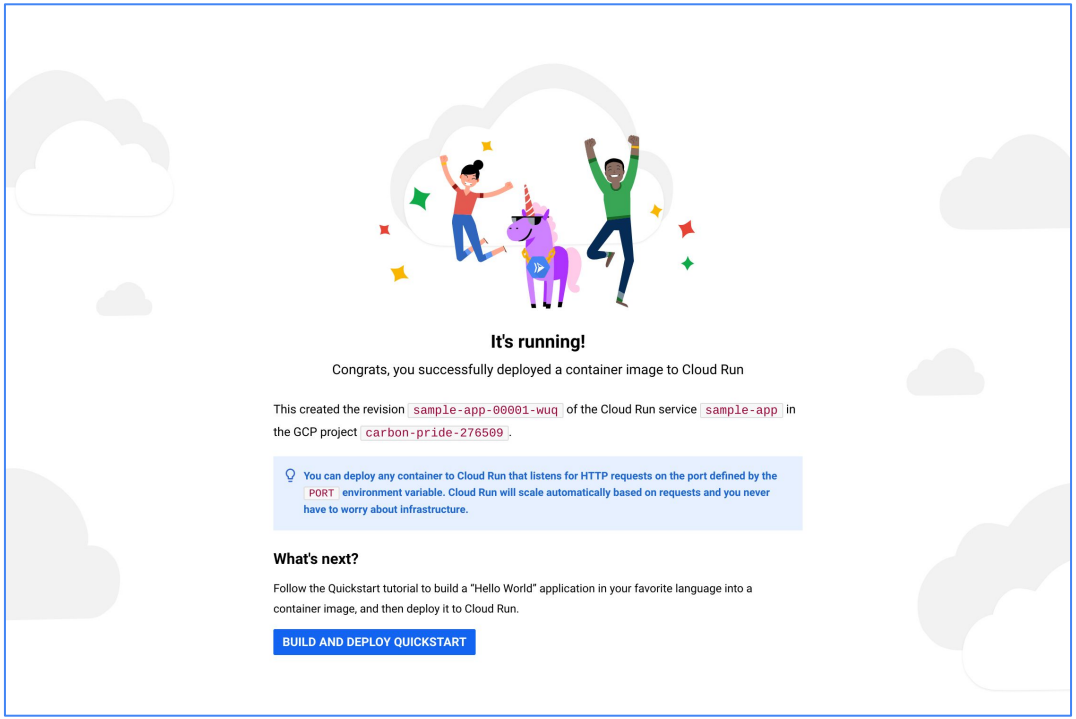


Google Cloud

手順 4. アプリケーションにアクセス



手順 5. アプリケーション表示



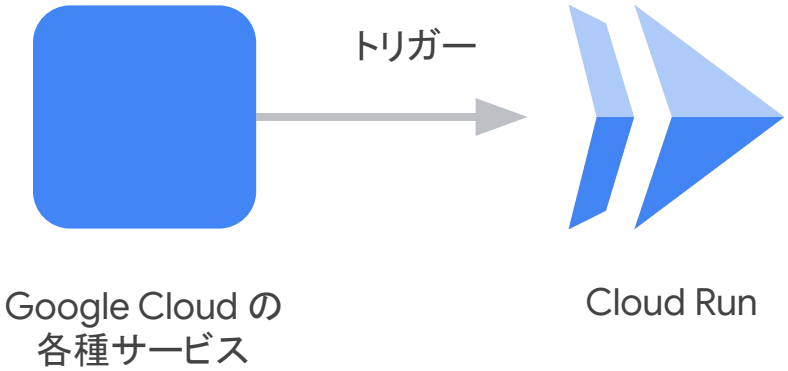
Google Cloud

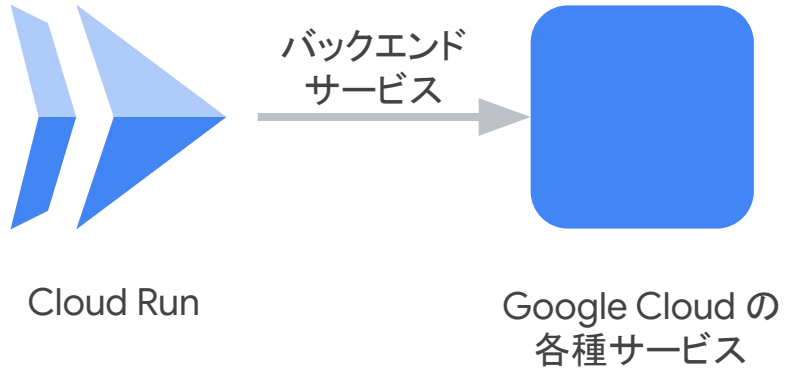
動作フロー

Cloud Run 外部のサービスを使用して
アプリケーションを実行

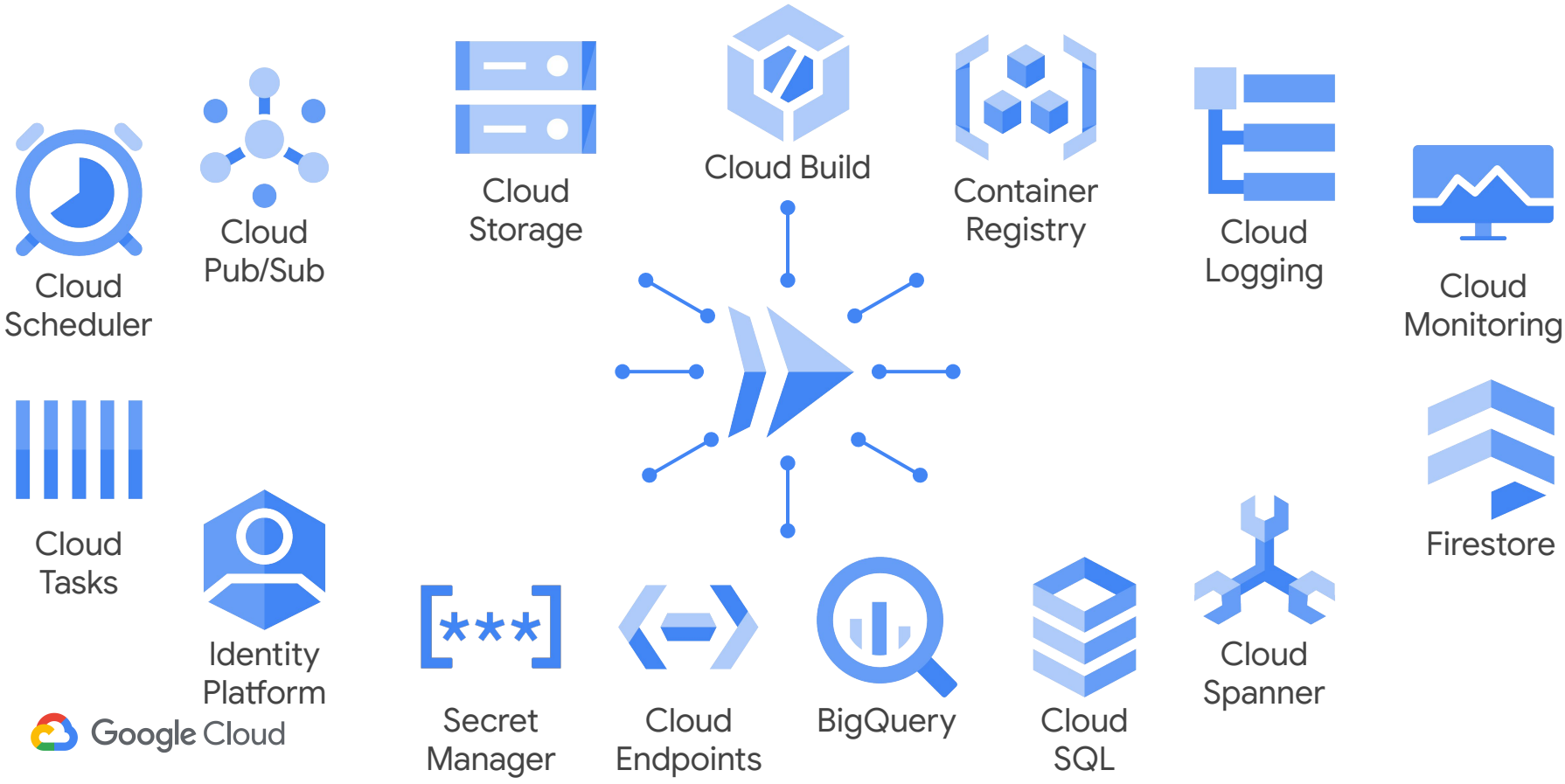
Google Cloud サービスとの接続

Cloud Run と Google Cloud サービスの接続





Cloud Run と Google Cloud サービスの接続



Pub / Sub からのイベント プッシュ

非同期タスク


Cloud Tasks の
HTTP Target を使って
Cloud Run の URL へ
認証トークンを付けてプッシュ
Cloud Tasks
Cloud Run

スケジュール実行

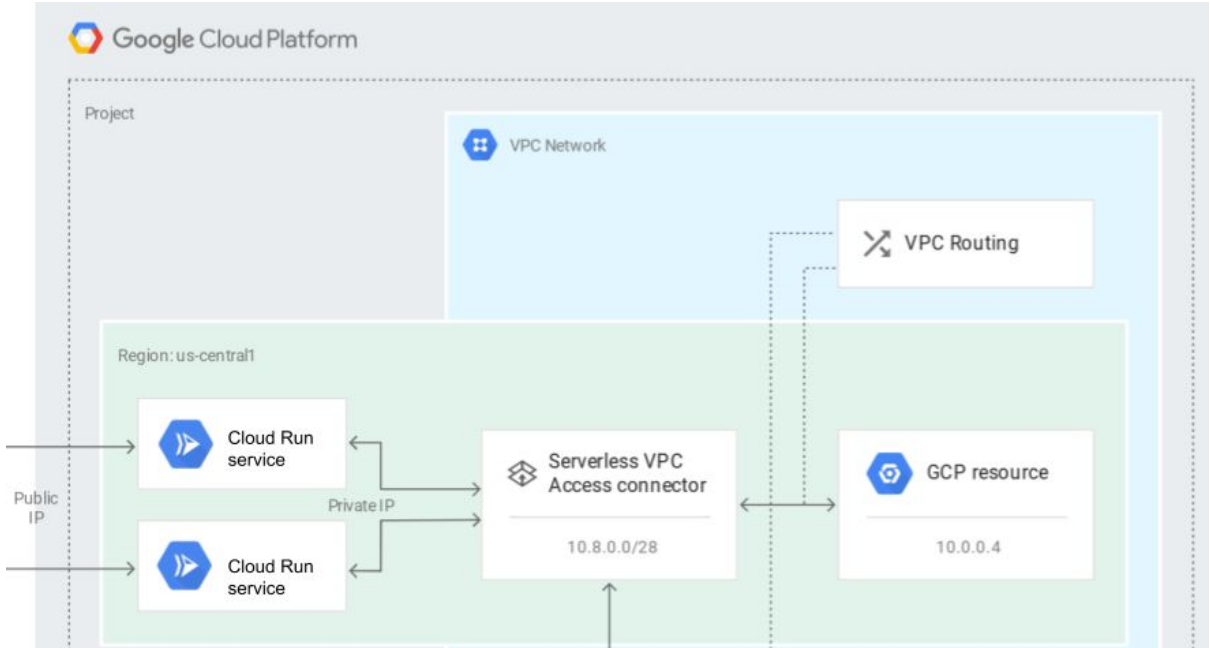

Cloud Scheduler から
認証トークンを付けて呼
び出す
Cloud
Scheduler
Cloud Run

データベース接続

- Cloud SQL Proxy
- Serverless VPC Access

[β 版機能] VPC ネットワーク接続 (サーバーレス VPC アクセス)

サーバーレスVPC アクセスを
使用しVPC ネットワークへ
直接接続



Google Cloud

Cloud Run と Cloud SQL の接続

（事前準備）アプリケーション側で接続情報を定義

（Java の場合）
Cloud SQL 用 JDBC ドライバを使用（*）

Cloud SQL Socket Factory for JDBC drivers を追加

```
<dependency>
  <groupId>com.google.cloud.sql</groupId>
  <artifactId>mysql-socket-factory-connector-j-8</artifactId>
  <version>1.0.16</version>
</dependency>
```

JDBC URL を作成

```
jdbc:mysql:///<DATABASE_NAME>?
cloudSqlInstance=<INSTANCE_CONNECTION_NAME>&
socketFactory=com.google.cloud.sql.mysql.SocketFactory&
user=<MYSQL_USER_NAME>&
password=<MYSQL_USER_PASSWORD>
```

Cloud SQL 接続を取得

```
Connection conn = DriverManager.getConnection (JDBC_URL)
```

Cloud SQL へ クエリーを発行

```
ResultSet rs = conn.prepareStatement (SQL).executeQuery ()
```

Google Cloud

(*) https://github.com/GoogleCloudPlatform/cloud-sql-jdbc-socket-factory

（事前準備）アプリケーション側で接続情報を定義

（Python の場合）
Cloud SQL Proxy を使用

Cloud SQL Proxy を呼び出し

```
./cloud_sql_proxy
-instances=<INSTANCE_CONNECTION_NAME>=tcp:3306
```

Cloud SQL 接続を取得

```
import pymysql
connection = pymysql.connect (host='127.0.0.1',
                              user='DATABASE_USER',
                              password='PASSWORD',
                              db='DATABASE_NAME')
```

（PHP の場合）
Cloud SQL Proxy を使用

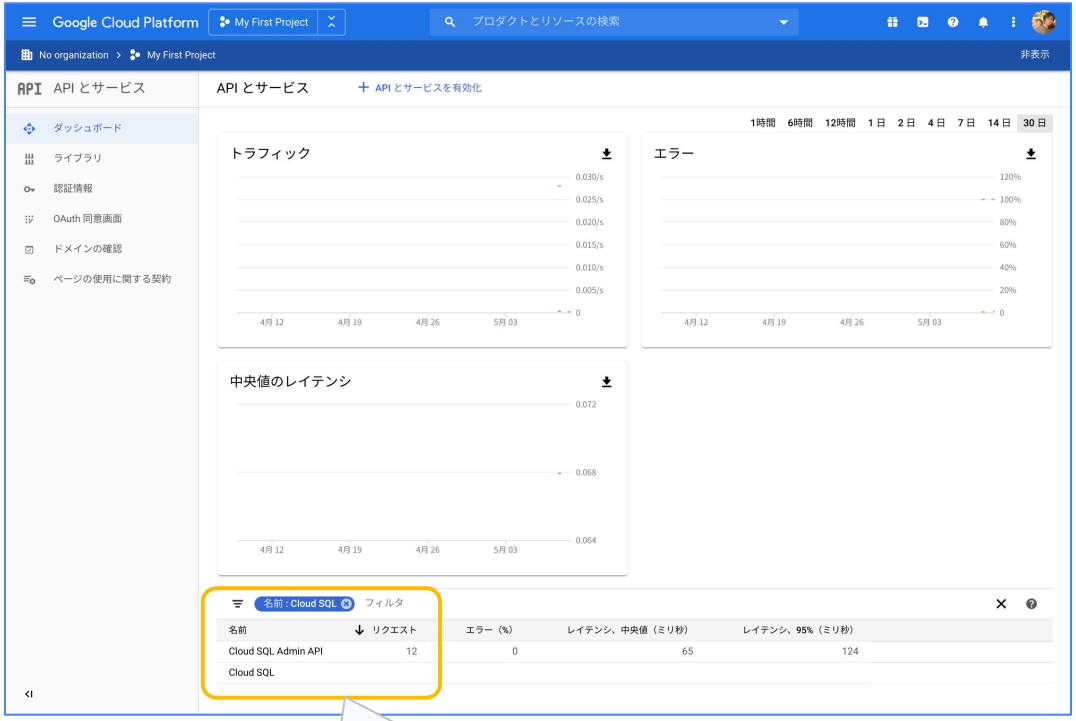
Cloud SQL Proxy を呼び出し

```
./cloud_sql_proxy
-instances=<INSTANCE_CONNECTION_NAME>=tcp:3306
```

Cloud SQL 接続を取得

```
$dbName = 'DATABASE_NAME';
$dbUser = 'DATABASE_USER';
$dbPass = 'PASSWORD';
$mysqli = new mysqli ('127.0.0.1', $dbUser, $dbPass, $dbName, 3306);
```

手順 1. Cloud SQL の有効化確認

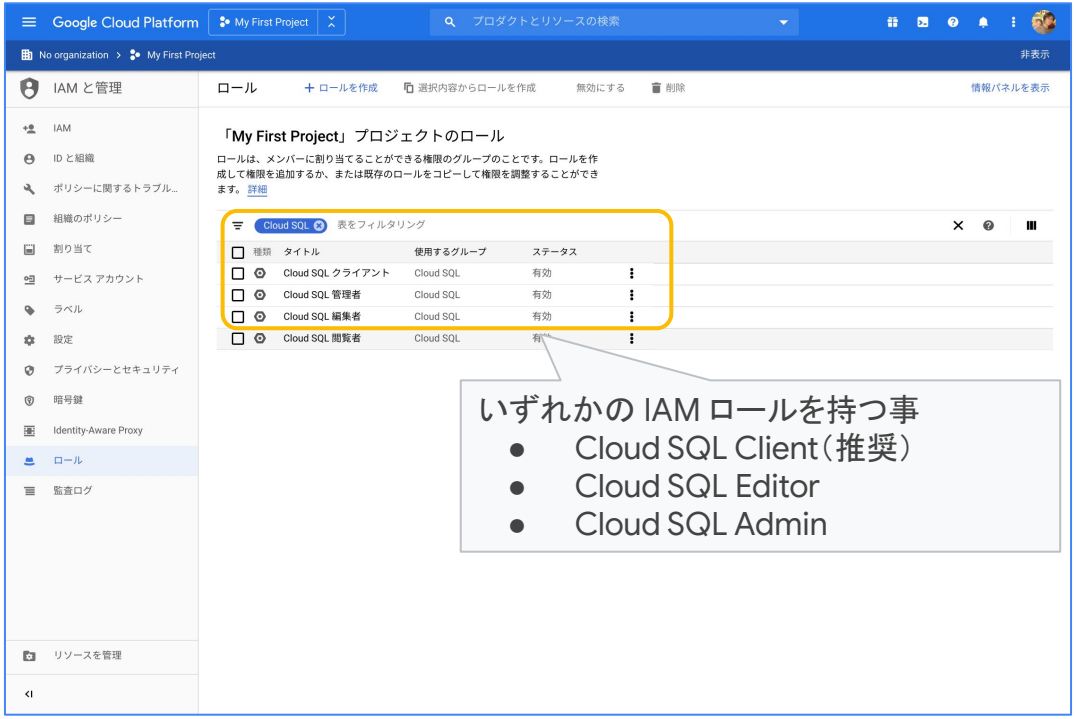


以下が有効であること

- Cloud SQL
- Cloud SQL Admin

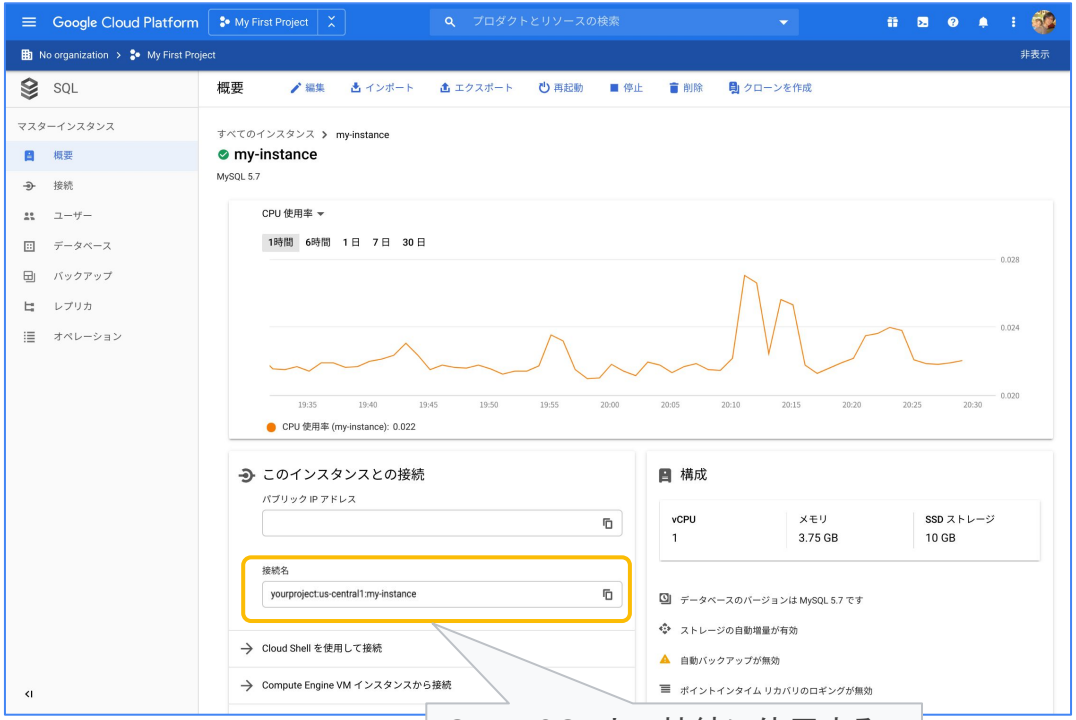
Google Cloud

手順 2. Cloud SQL 実行可能な IAM ロールの設定



Google Cloud

手順 3. Cloud SQL インスタンス作成

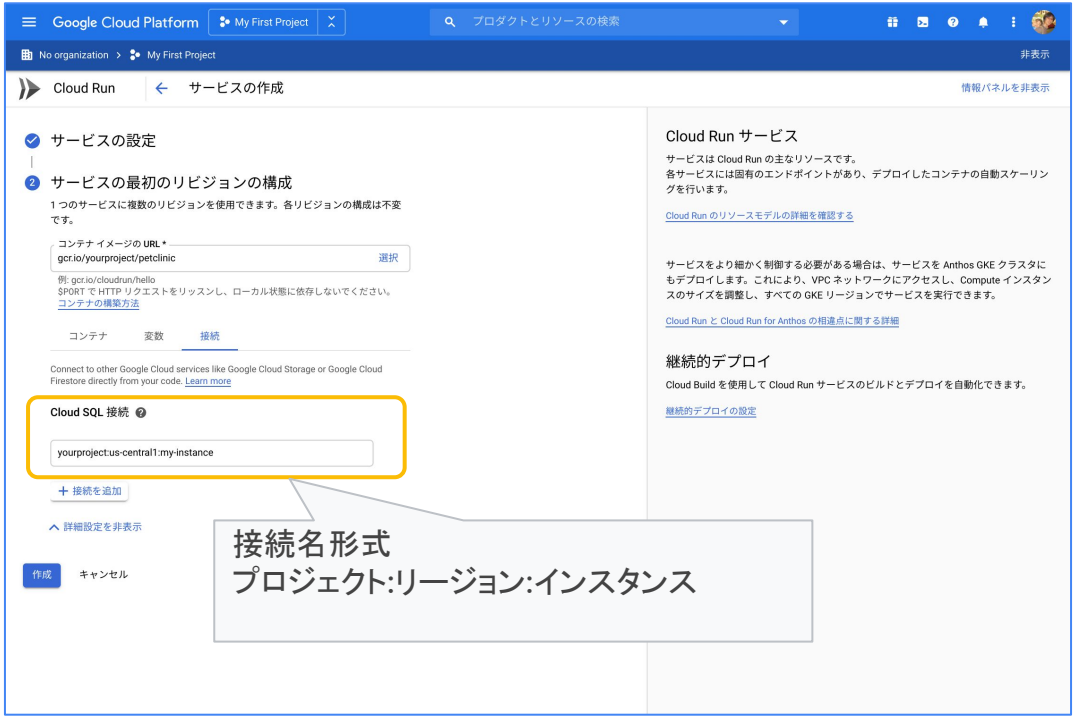


Google Cloud

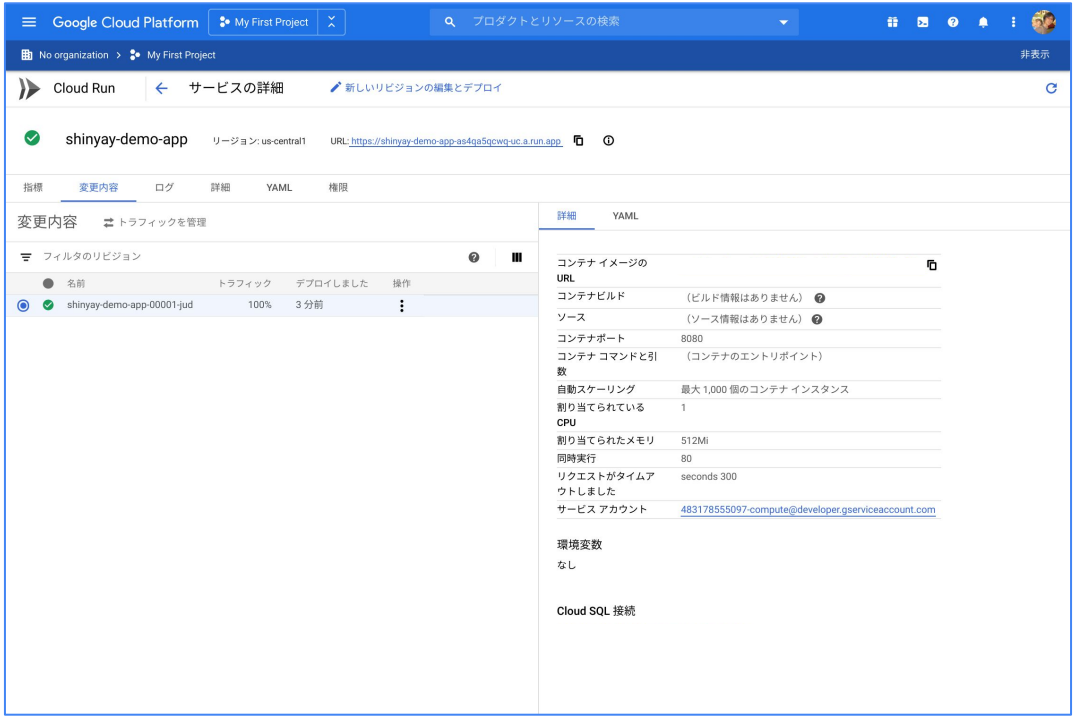
手順 4. リビジョンの構成から詳細設定を表示

Google Cloud

手順 4. Cloud SQL 接続を追加



手順 5. アプリケーション起動



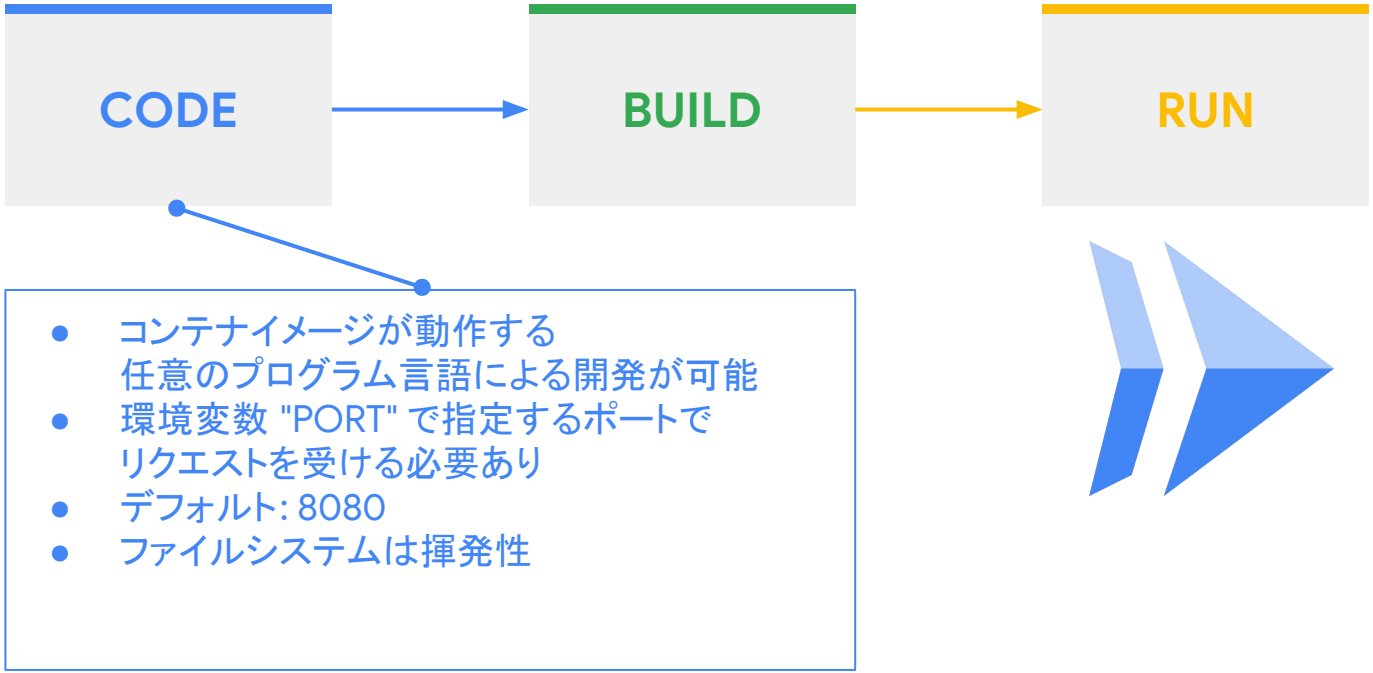
この章の振り返り

- Cloud Run は **Knative** ベースのサーバーレス環境
- **コンテナをデプロイ**するだけで簡単にサービス開始
- Google Cloud の**様々なサービス**に接続することが可能

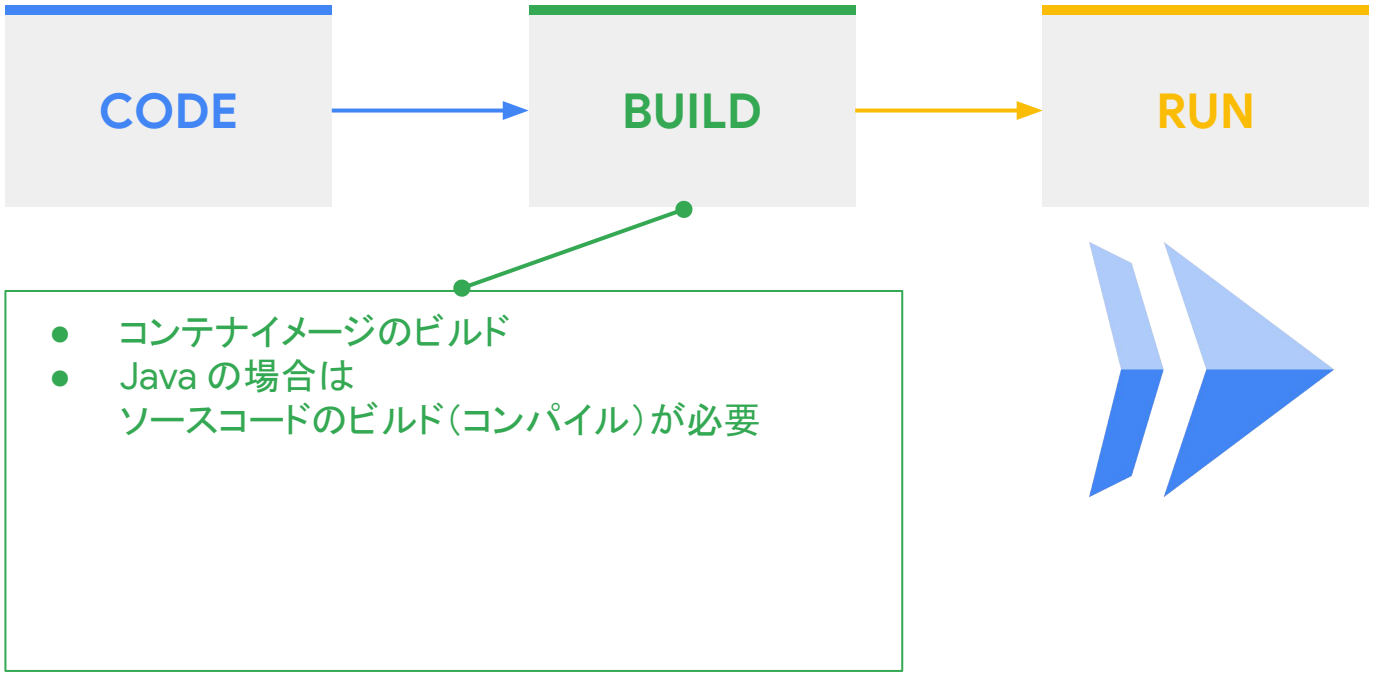
Cloud Run
入門編

Google Cloud

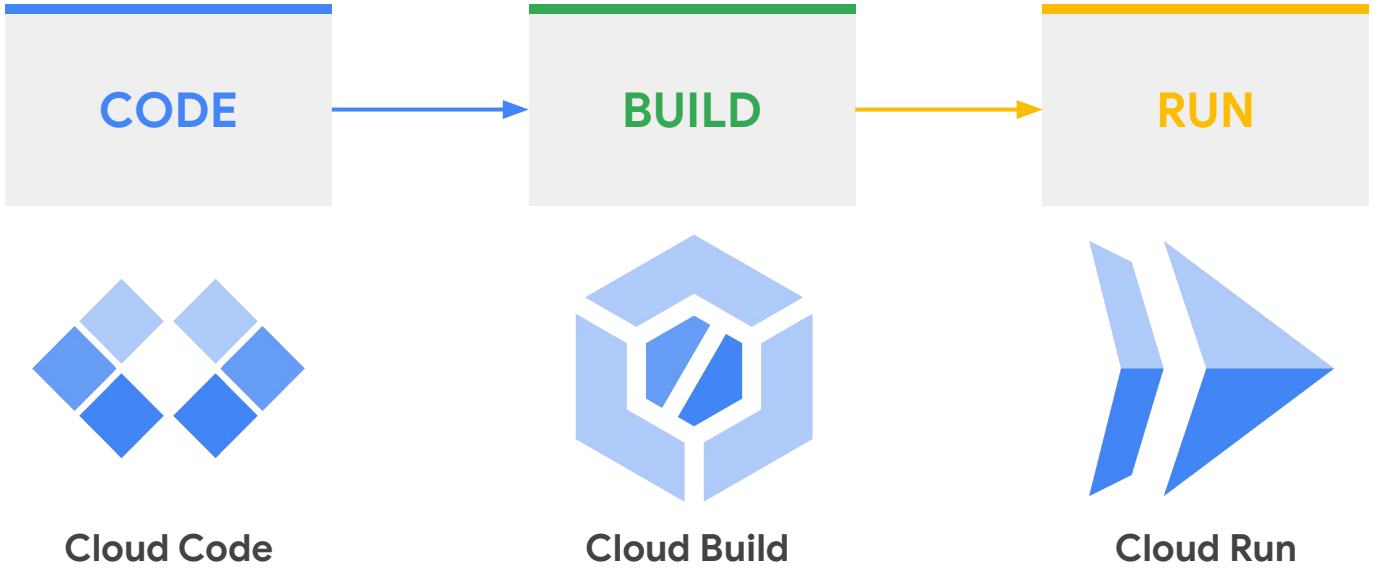
アプリケーション開発フロー



アプリケーション開発フロー



Google が提供するアプリケーション開発ツール



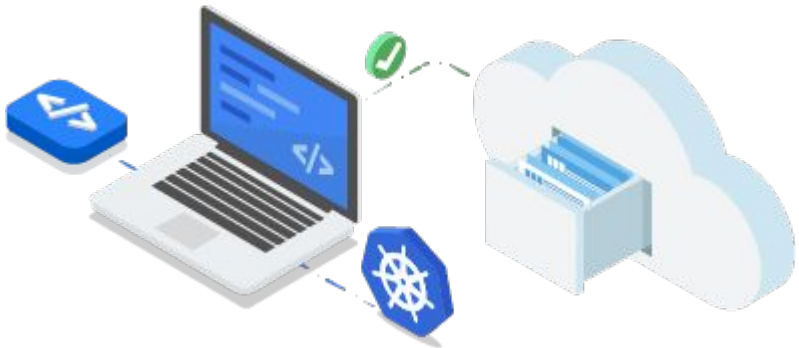
Cloud Code を使用した開発環境

CODE | BUILD

Cloud Code

Google が提供するアプリケーション開発支援ツール

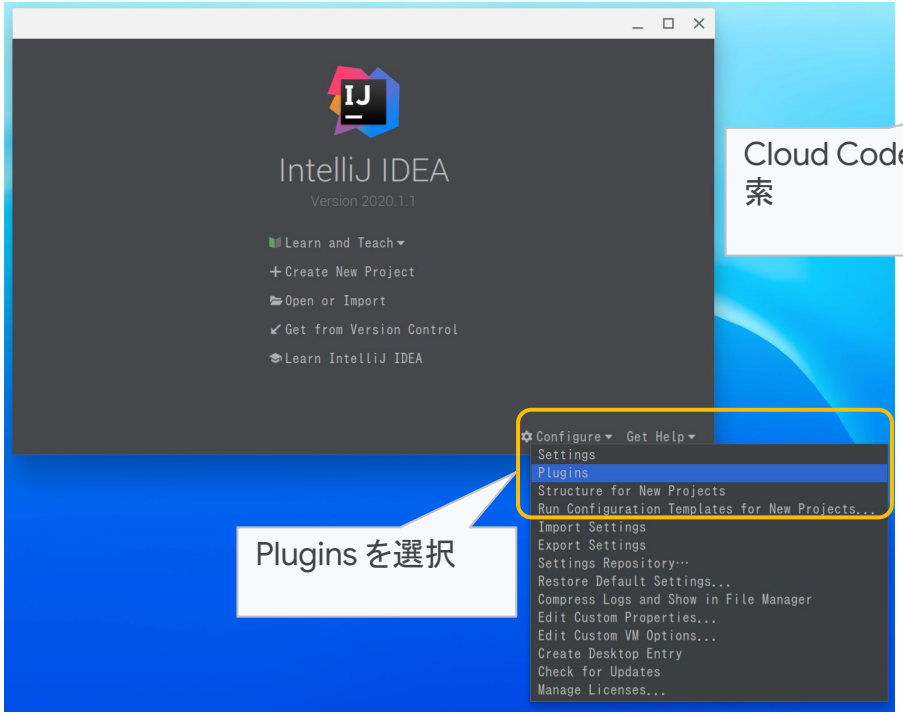
Visual Studio Code や JetBrains IDE (IntelliJ など)の機能を拡張

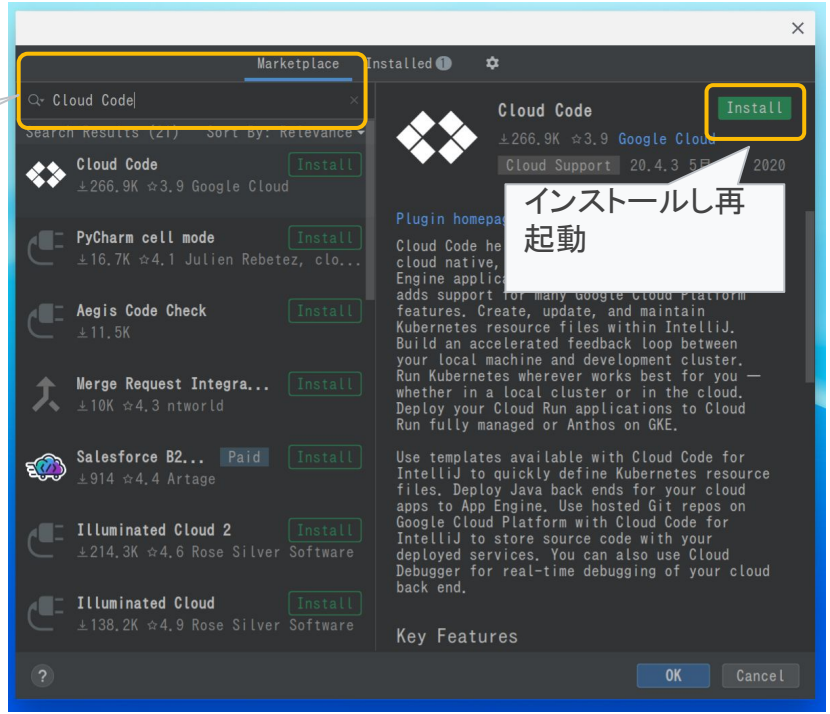


Cloud Code の代表的な機能

- Java のサポート
- Node.js、Go、Python、.NET Core のサポート
- スターター テンプレート
- 複数のデプロイ プロファイル / ターゲットのサポート
- ログのストリーミングと表示
- Cloud Run サポート
- Cloud Source Repositories のサポート
- Cloud Build のサポート
- Kubernetes YAML のサポート

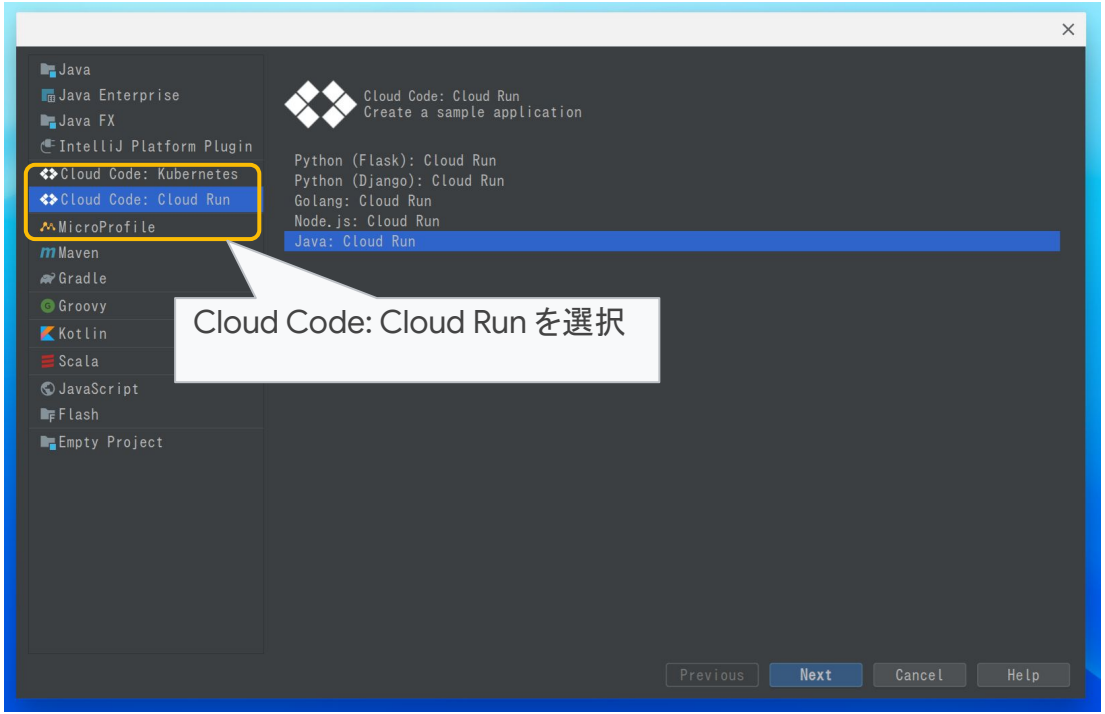
手順 1. Cloud Code プラグインのインストール (Cloud Code for IntelliJ の場合)

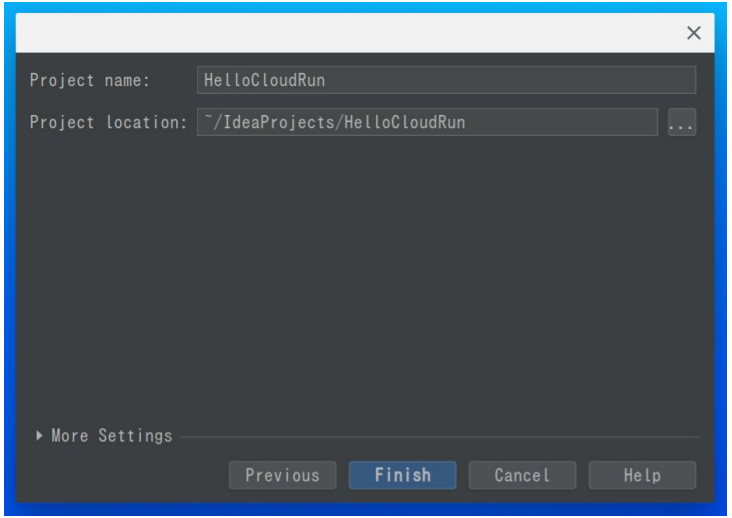




Google Cloud

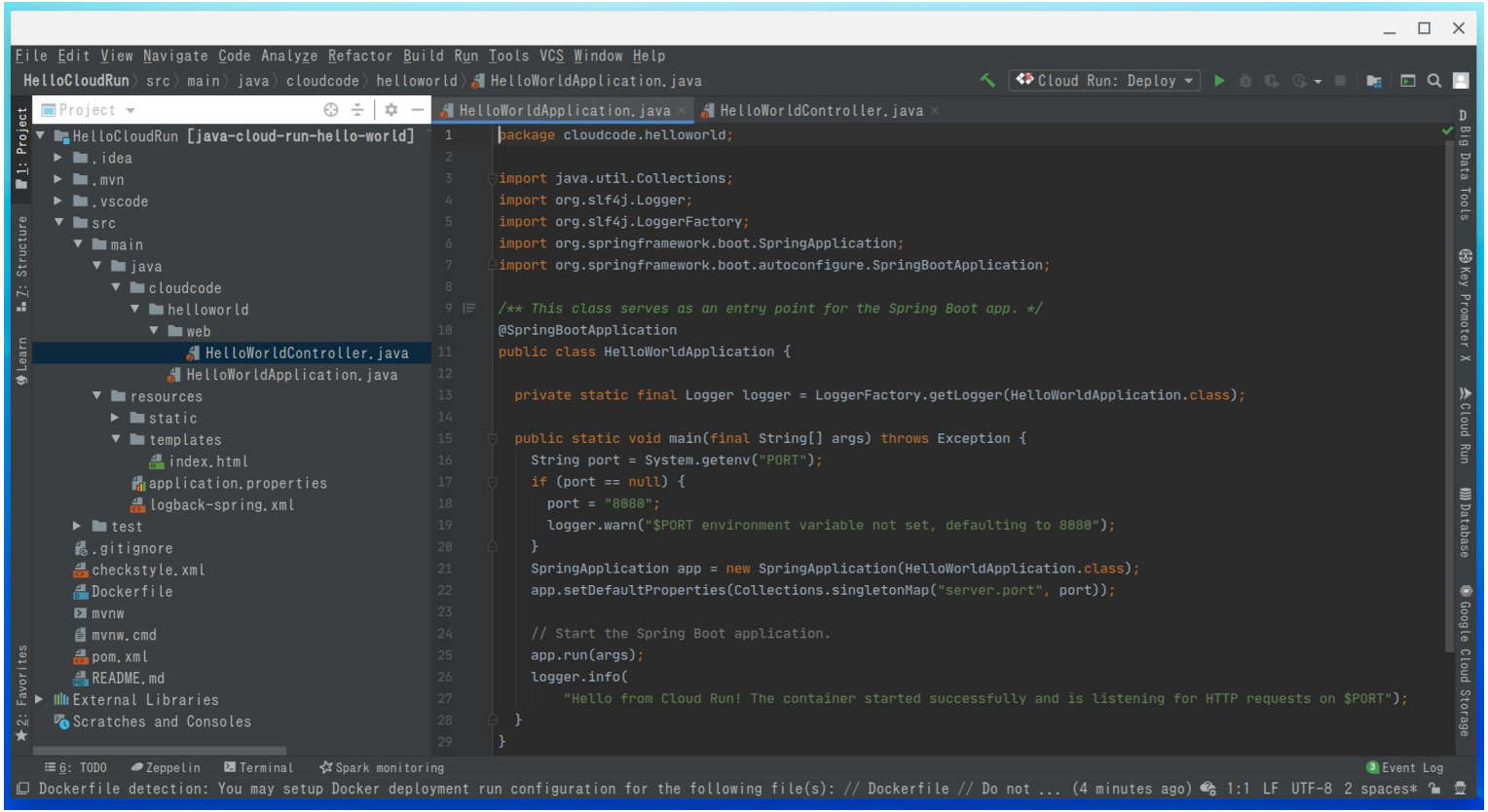
手順 2. プロジェクト作成





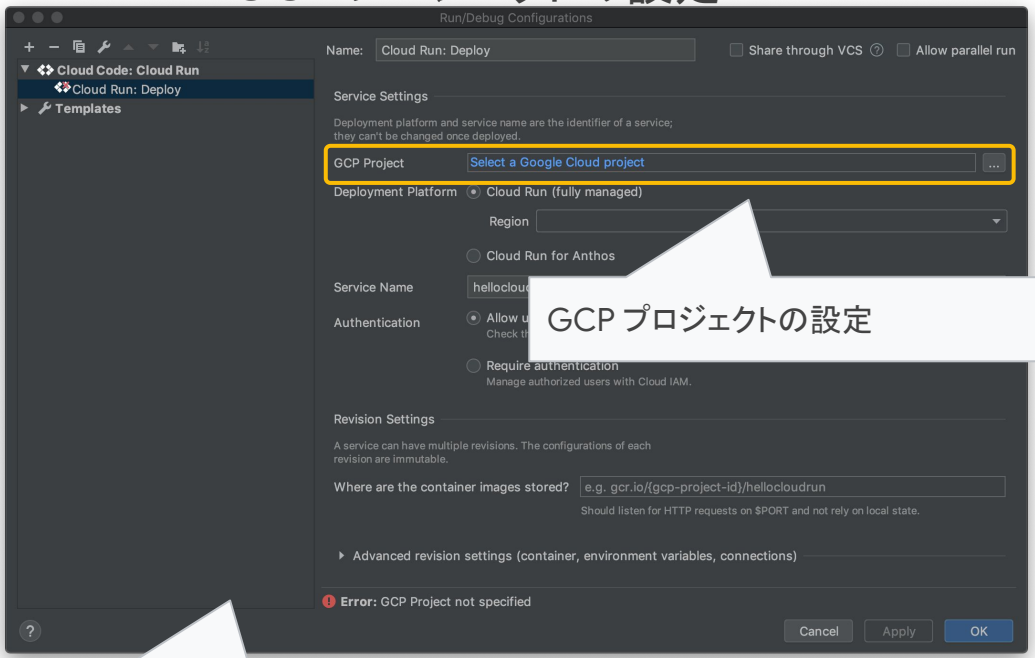
Google Cloud

手順 3. サンプルコードの自動生成



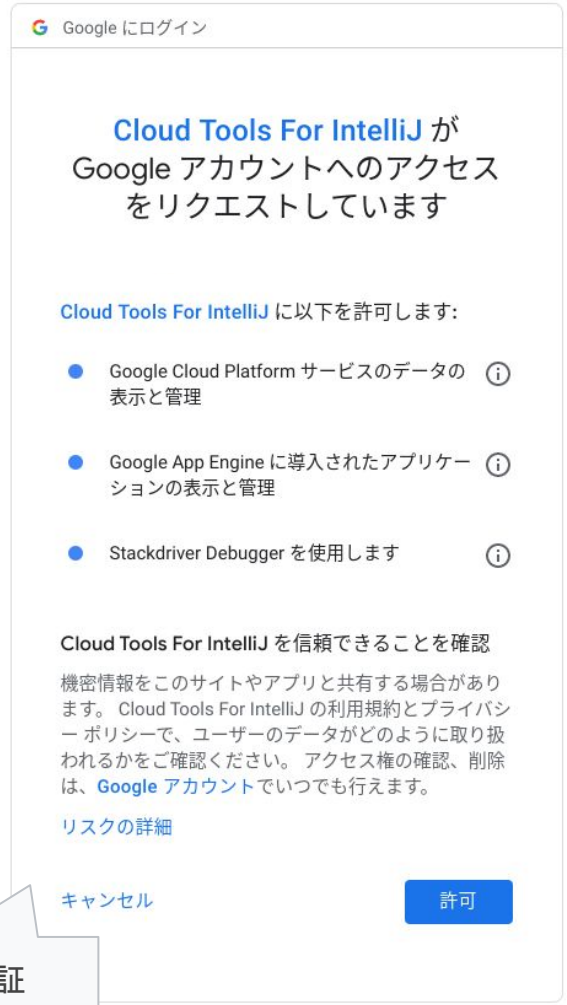
Google Cloud

手順 4. Cloud Run 実行構成
- GCP プロジェクトの設定



GCP プロジェクトの設定

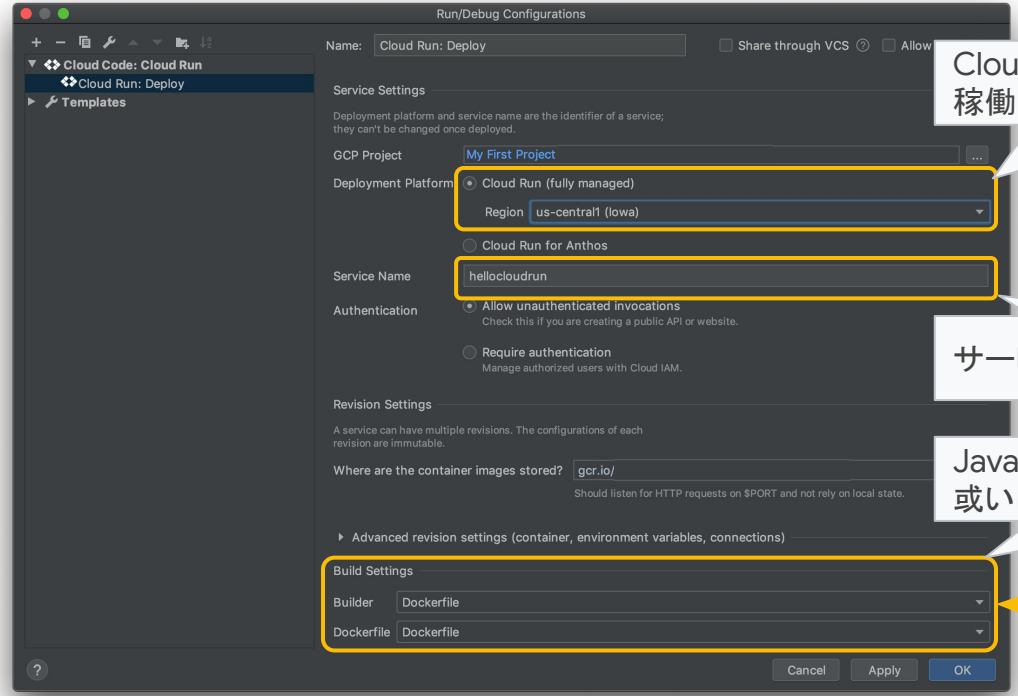
[Run] > [Edit Configurations] で表示



ブラウザ経由で認証

Google Cloud

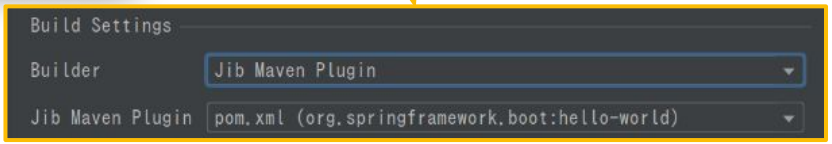
手順 5. Cloud Run 実行構成 - ビルドおよびデプロイ設定



Cloud Run (フルマネージド)を選択し
稼働させるリージョンを設定

サービス名を設定

Java のビルド設定に Dockerfile
或いは Jib を設定



Google Cloud

（参考）Jib

https://github.com/GoogleContainerTools/jib

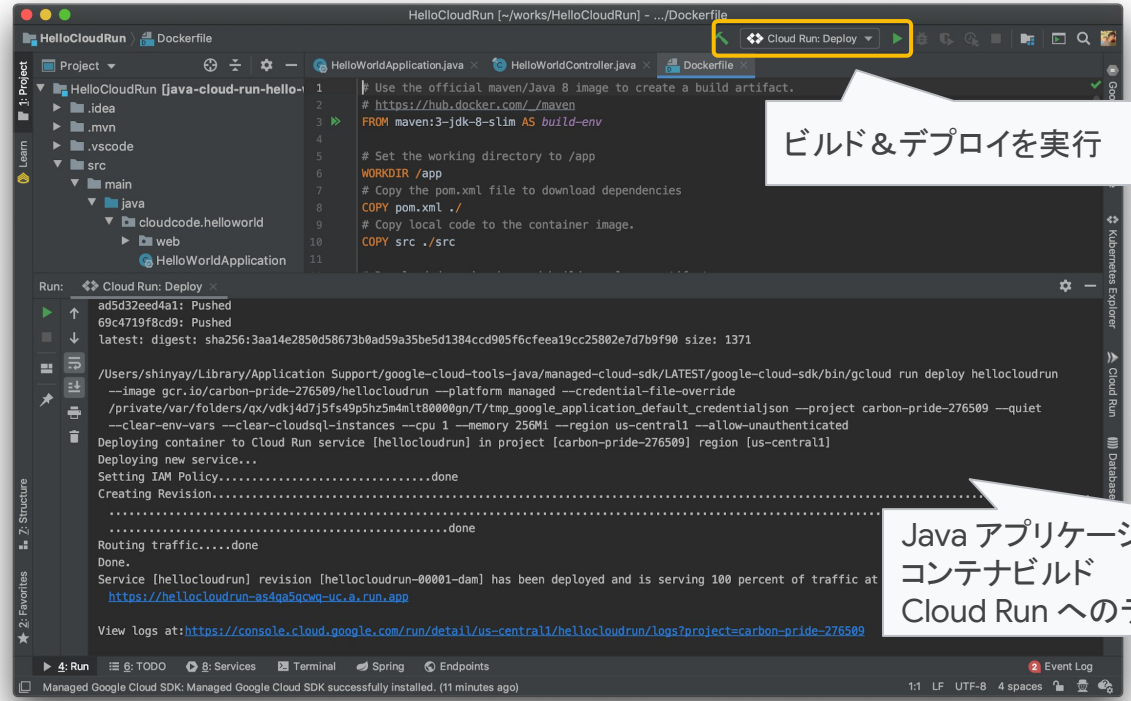
Jib

Google がオープンソースとして公開している
Java アプリケーション の コンテナイメージを作成するツール

Java のビルドツールである "Maven" や "Gradle" のプラグインとして動作する

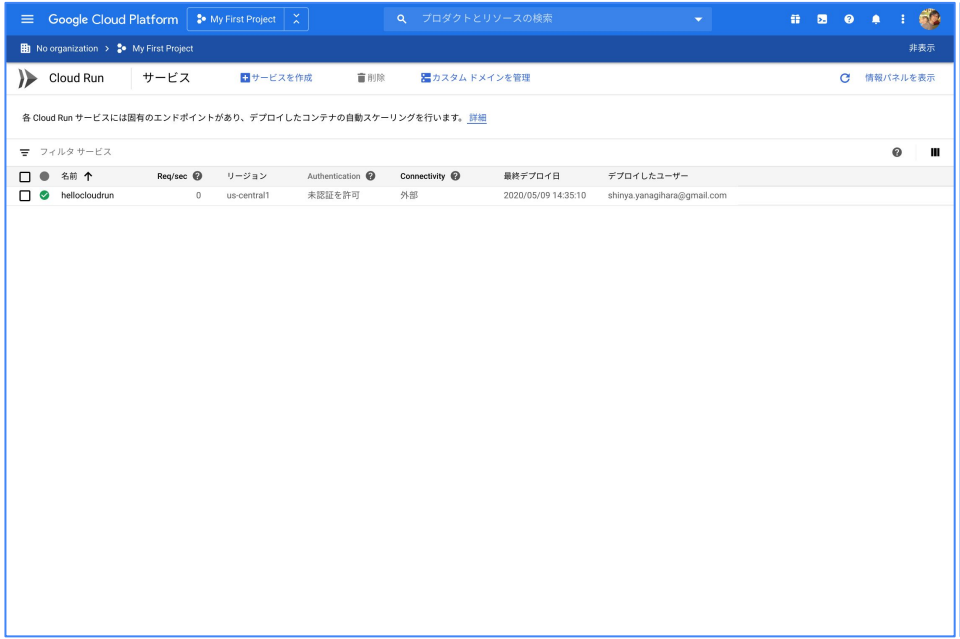
コンテナイメージは Google Container Registry や Docker Hub などのコンテナレジストリへ Jib による格納が可能

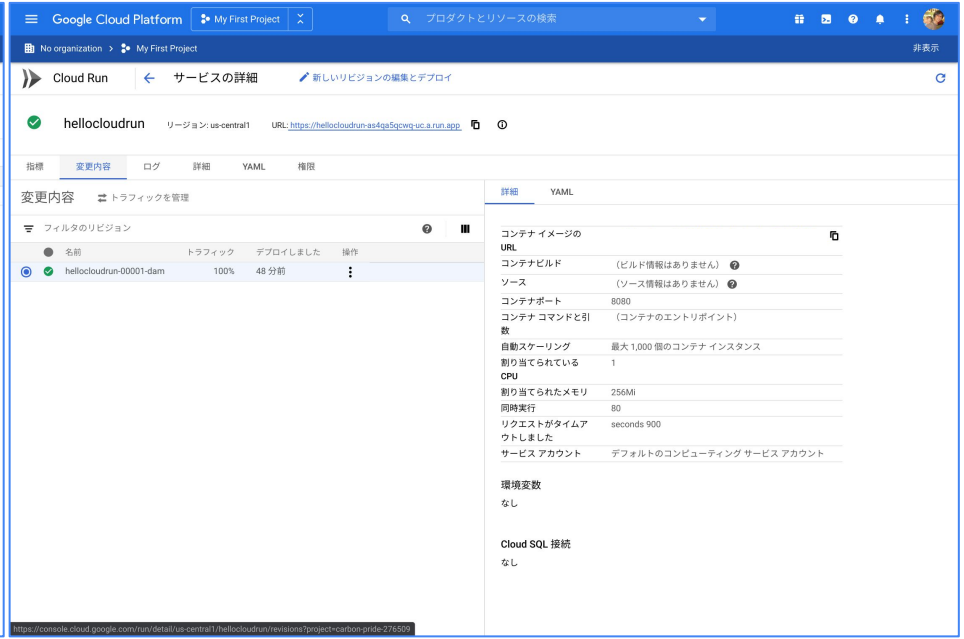
手順 6. ビルド＆デプロイ実施



Google Cloud

手順 7. Cloud Run デプロイ確認





Google Cloud

この章の振り返り

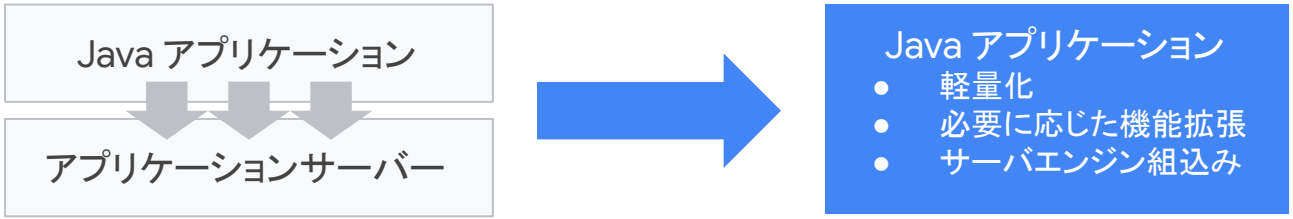
- Cloud Run で動かすアプリケーションの**開発フローはシンプル**
- Cloud Code を利用すると**統合開発環境上からのデプロイ**もシンプルに実施可能

Cloud Run
実践編

04

Google Cloud

Java アプリケーションのトレンド



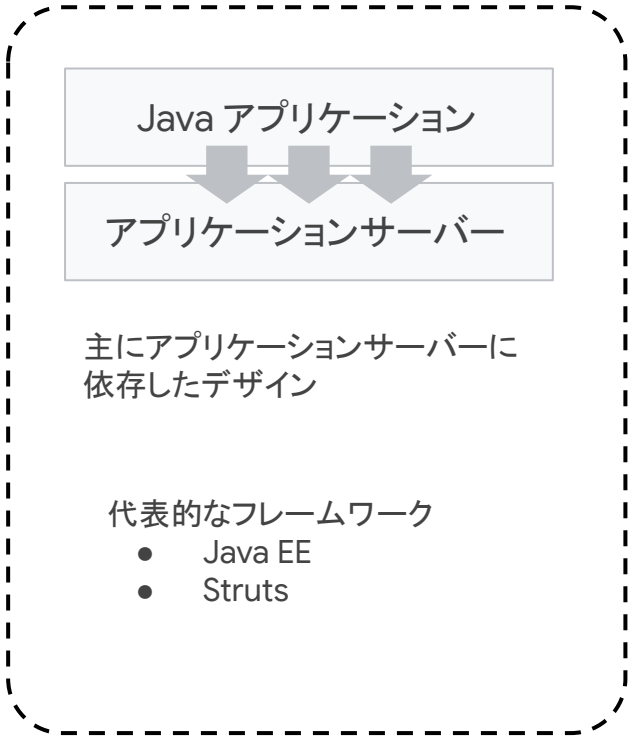
主にアプリケーションサーバーに
依存したデザイン

代表的なフレームワーク
- Java EE
- Struts

Kubernetes やサーバーレスを
効果的に利用するデザイン

代表的なフレームワーク
- Eclipse MicroProfile
- Spring Boot
- Micronaut
- Quarkus
- Helidon

一般的なアプリケーションポートフォリオの割合



主にアプリケーションサーバーに
依存したデザイン

代表的なフレームワーク
- Java EE
- Struts

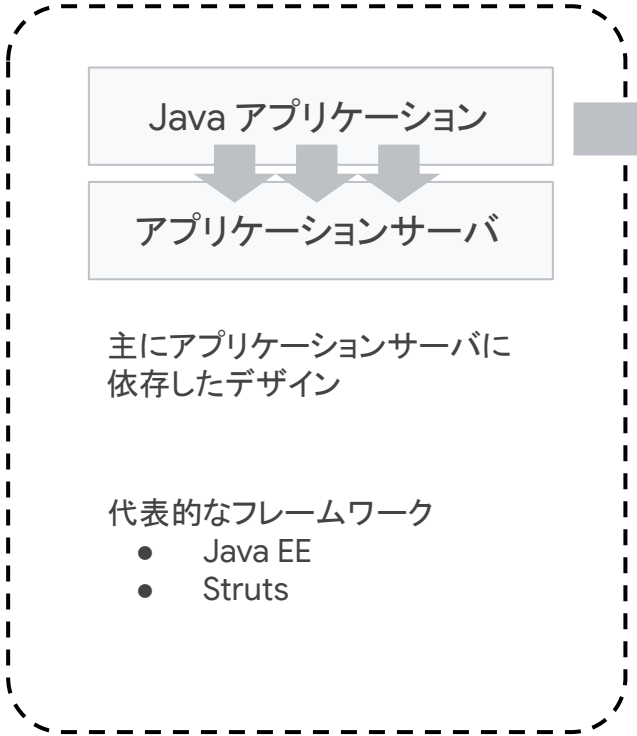
>

Java アプリケーション
- 軽量化
- 必要に応じた機能拡張
- サーバエンジン組込み

Kubernetes やサーバーレスを
効果的に利用するデザイン

代表的なフレームワーク
- Eclipse MicroProfile
- Spring Boot
- Micronaut
- Quarkus
- Helidon

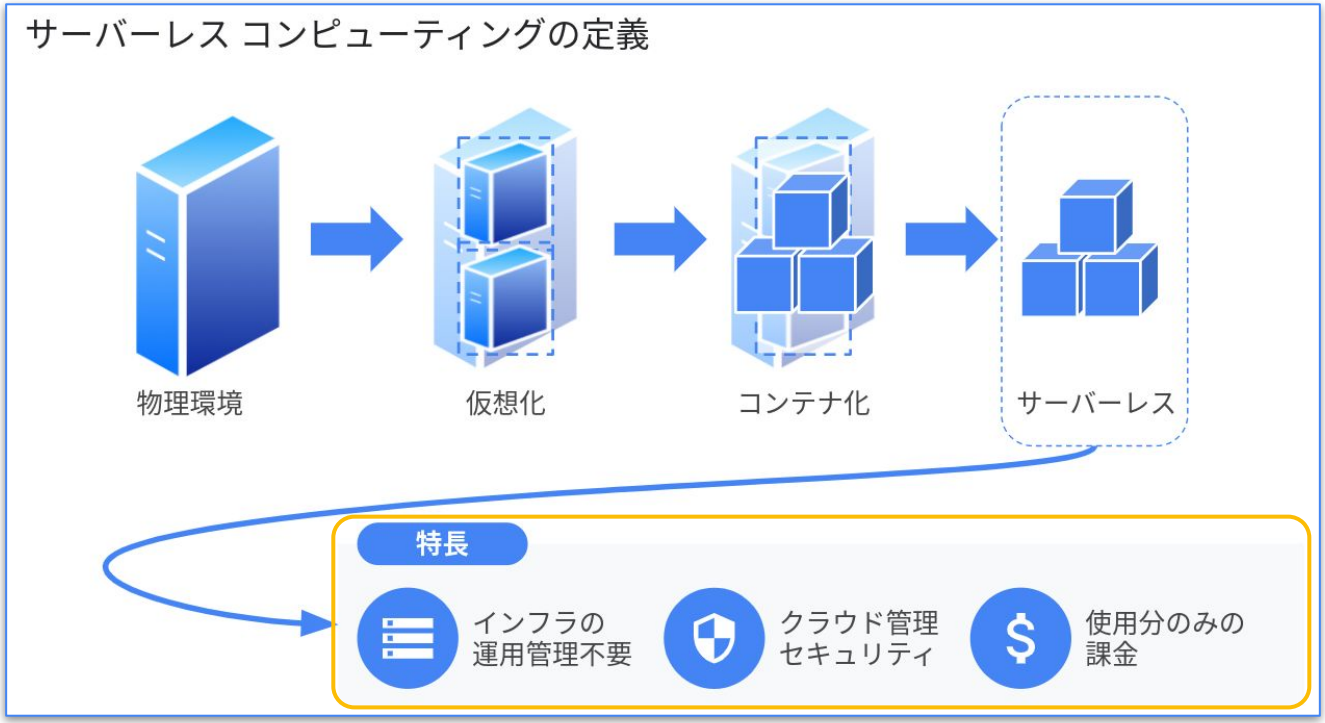
休眠アプリケーション資産は多くないでしょうか？



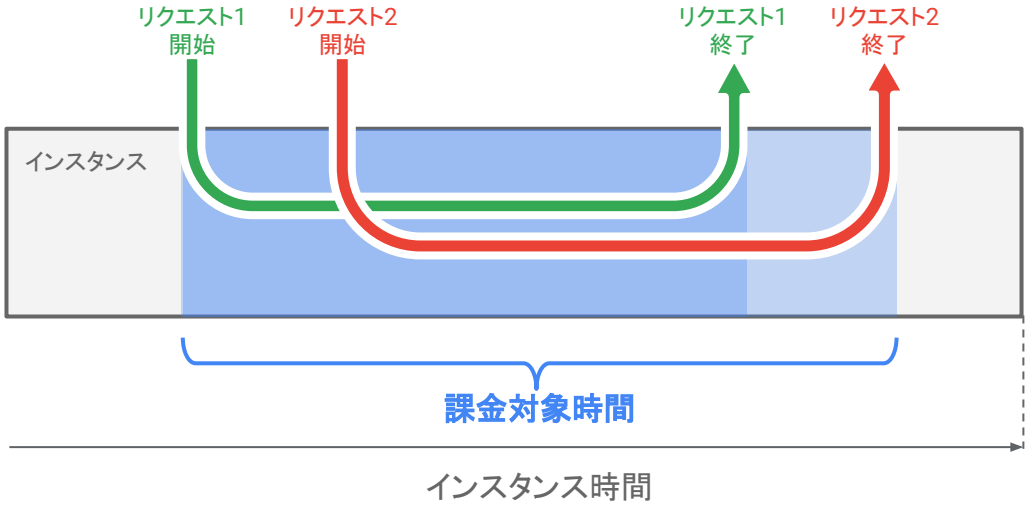
休眠アプリケーションの課題

- 機能変更や拡張はないけれど
  広く利用され続けているため
  運用し続けている
- セキュリティ対応のみの変更を実施
- 季節的に一時的なトランザクション量増加があるために運用チームも外せない
- アプリケーションサーバの運用保守は継続的に実施
- ハードウェアの EOL に伴う移行作業が求められる

休眠アプリケーションにサーバーレスは使えないでしょうか？



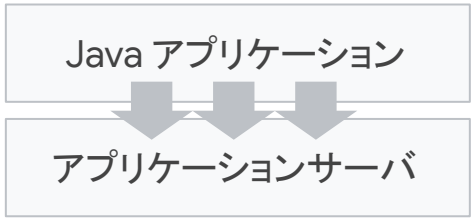
課金時間の考え方



課金
非課金

- CPU 時間
- メモリ時間
- ネットワーク転送料金
- リクエスト数

Struts アプリケーションをサーバーレス化



使用フレームワーク
- Struts

サーバーレス化

- アプリケーション サーバの依存を排除
- 軽量フレームワークに移行
- コンテナ化

休眠アプリケーション資産を安価かつ安心に運用へ

（前提）サンプル Struts アプリケーション構成

モダナイズ 1. pom.xml に Spring の依存関係を追加

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-gcp-starter-sql-mysql</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.struts</groupId>
  <artifactId>struts2-spring-plugin</artifactId>
</dependency>
```

- GCP で提供する Cloud SQL の利用
- Spring Web の利用
- Spring Data JPA の利用
- 組込み Tomcat の利用
- JSP の利用
- Struts の DI を Spring の DI に変更

モダナイズ 2. Spring Auto Configuration クラスの追加

```
@SpringBootApplication
public class Application {

    public static void main (String... args) {
        SpringApplication.run (Application.class, args);
    }
}
```

@SpringBootApplication アノテーションをつけた
エントリポイントのクラスを作成

モダナイズ 3. サーブレットの移行

```
@Bean
public FilterRegistrationBean filterDispatch () {
    return buildFilterRegistration (2, new StrutsPrepareAndExecuteFilter (),
        asList ("/", "/addUserForm.action", "/addUser.action",
            "/findUserForm.action", "/findUser.action",
            "/listAllUsers.action"));
}

private FilterRegistrationBean buildFilterRegistration (int order, Filter
filter, List<String> urlPatterns) {
    FilterRegistrationBean registration = new FilterRegistrationBean ();
    registration.setFilter (filter);
    registration.setUrlPatterns (urlPatterns);
    registration.setOrder (order);
    return registration;
}
```

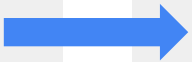
FilterRegistrationBean を使用して
Struts で使用しているアクションを登録

モダナイズ 4. Action の Bean 登録

```
public class AddUser {...}

public class FindUser {...}

public class ListAllUsers {...}
```



```
@Component
public class AddUser {...}

@Component
public class FindUser {...}

@Component
public class ListAllUsers {...}
```

Spring DI で使用できるように Bean 登録を実施

モダナイズ 5. Cloud SQL の接続設定

```
spring:
 jpa:
  generate-ddl: true
  properties.hibernate.dialect: org.hibernate.dialect.MySQL5Dialect
 cloud:
  gcp:
   sql:
    database-name: < 接続対象の Cloud SQL に作成した DB 名>
    instance-connection-name: < Cloud SQL インスタンスへの接続文字列>
 database: mysql
```

Cloud SQL への接続情報を
application.yml に記述

モダナイズ 6. コンテナ化設定（Jib）を pom.xml に追加

```
<build>
  <plugins>
    <plugin>
      <groupId>com.google.cloud.tools</groupId>
      <artifactId>jib-maven-plugin</artifactId>
      <version>2.2.0</version>
      <configuration>
        <to>
          <image>gcr.io/my-gcp-project/my-struts</image>
        </to>
      </configuration>
    </plugin>
  </plugins>
</build>
```

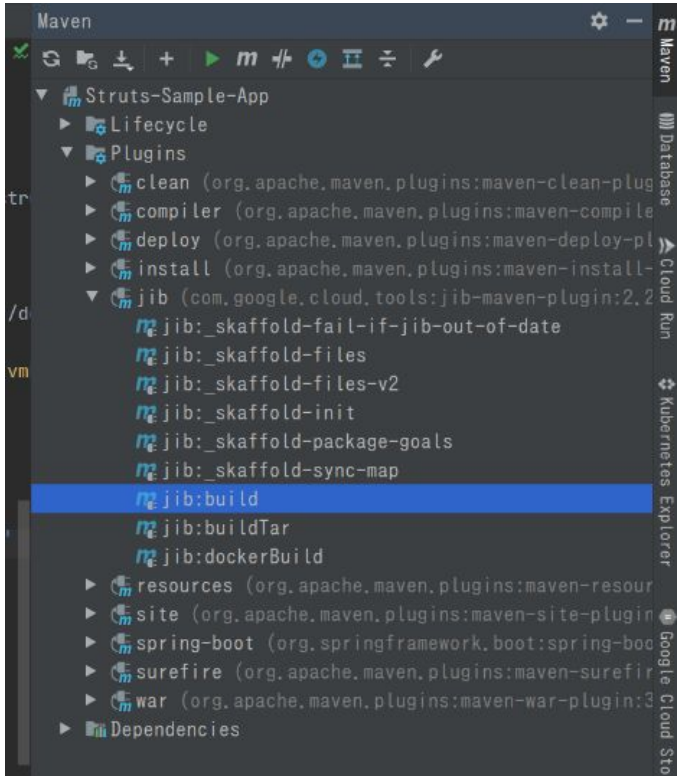
Jib のライブラリを追加

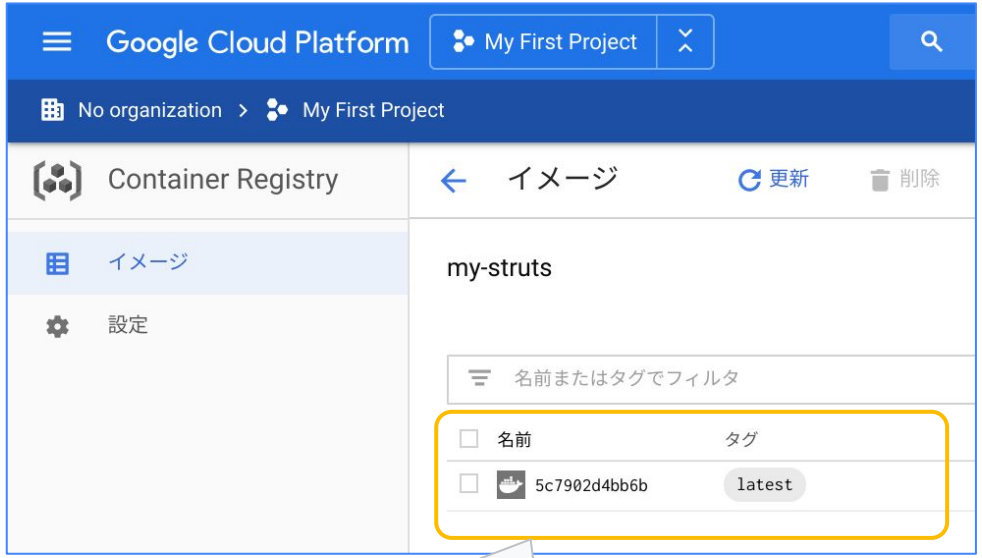
コンテナイメージの格納先になる
Google Container Registry の宛先とイメージ名を設定

```
<packaging>jar</packaging>
```

Java ビルドのパッケージングを WAR から JAR に変更

モダナイズ 7. Jib によるビルド

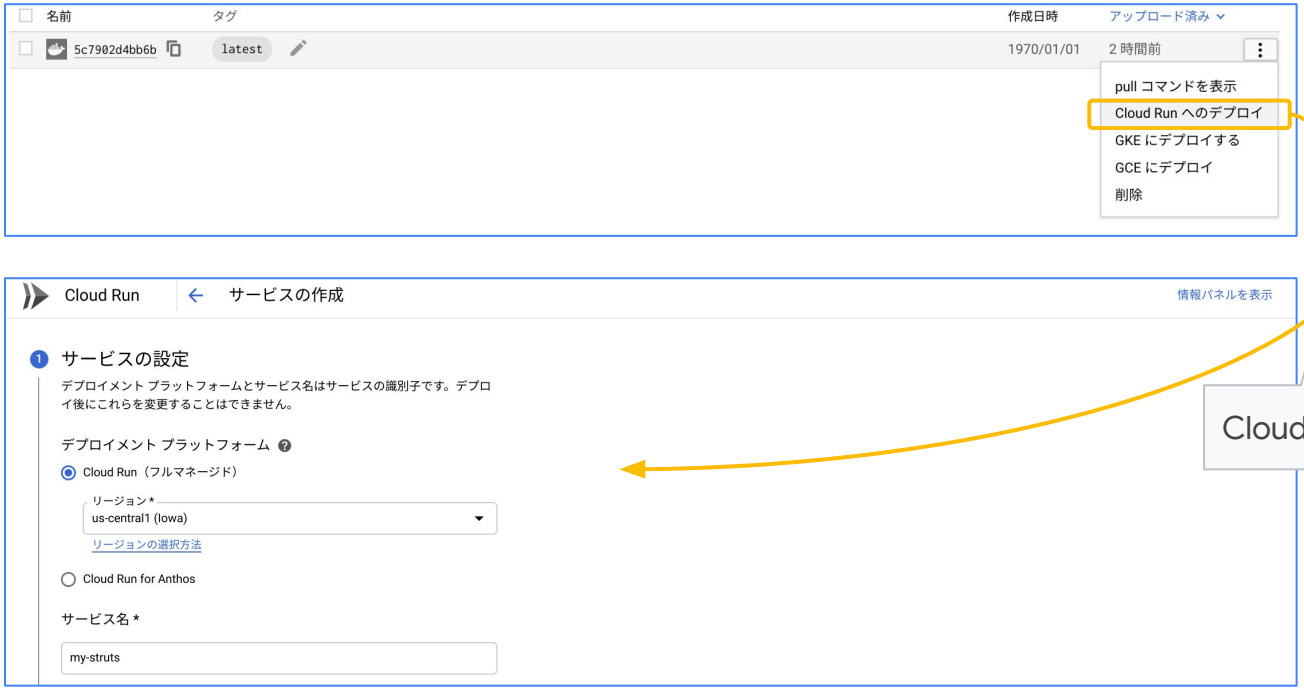




Jib でビルドしたコンテナイメージが
Container Registry に登録される

Google Cloud

モダナイズ 8. Container Registry から Cloud Run へデプロイ



Cloud Run のサービス作成画面へ遷移

Google Cloud

モダナイズ 9. Cloud Run デプロイ確認

Struts アプリケーションのサーバーレス化完了

Google Cloud

トラディショナルアプリケーションのサーバーレス化に関する注意点

1. Cloud Run で動作するコンテナのスペックによる制約

以下のスペックで十分な動作を満たすアプリケーションを選定

- CPU: 1〜2 vCPU
- メモリ: 128 MB〜2GB
- ファイルシステム: 揮発性

2. アプリケーションの挙動による制約

以下のような動作をするアプリケーションはアーキテクチャや
挙動の検討を要する場合がある

- Cloud Run の応答可能時間を越えるもの（5 分〜最大 15 分）
- グローバルトランザクションを使用している
- セッション共有している、等

この章の振り返り

- 新規アプリケーションだけでなく、**既存アプリケーションもサーバーレス化**可能

Key Takeaways

Google Cloud

Cloud Run

- コンテナを**シンプルにサーバーレス**化
- OSS の Knative をベースとした **ロックインフリー**
- Google Cloud の様々な**サービスと連携**
- 0 to N スケールし **未使用時は非課金**

Java と Cloud Run

本セッションでは、Java 開発を前提とした **Cloud Run** の利用について紹介しました。

以下のポイントを感じて頂けたなら嬉しく思います

- **既存資産の有効活用**ができる
- **従来の Java 知識**が役に立つ
- **コンテナに詳しくなくても** コンテナ化ができる
- **Kubernetes を知らなくても** コンテナを動作できる
- **今日から Cloud Run が始められそう**

Next steps

オンラインの **ハンズオン** があります。Anthos をはじめとした Google Cloud の製品群を体験いただけます

- Google Cloud Run Serverless Workshop on Qwiklabs (https://google.qwiklabs.com/quests/98?search_id=5454570)
- etc..

次回のセッションの紹介

- 次回のテーマは Anthos が提供する Kubernetes セキュリティ
- 日程:2020/06/19 20:00〜
- 今回触れなかった、Kubernetes を運用する上で必要となるセキュリティの実現について説明

Thank you