Azure Kubernetes Service (AKS) によるアプリケーション開発

発行日 2018 年 5 月 15 日

更新履歴

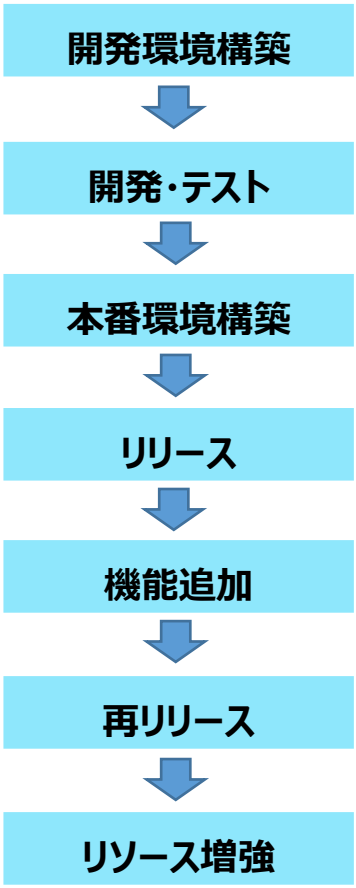
版数	発行日	更新履歴
第 1 版	2018 年 4 月 1 日	初版発行
第 2 版	2018 年 5 月 15 日	第 2 版発行。Azure Kubernetes Service の製品名変更

はじめに

本自習書をご利用いただきありがとうございます。

本自習書では、Azure Kubernetes Service (AKS) の技術的な説明に終始せず、Azure Kubernetes Service (AKS) がどうして必要なのかをイメージしやすいよう、開発エンジニアがよく経験する実際の開発業務の中で Azure Kubernetes Service (AKS) の使い方を説明することで、Azure Kubernetes Service (AKS) を使う人の理解の一助となることを狙いとしています。

具体的には実際の下記アプリケーション開発フローの中にて、Azure Kubernetes Service (AKS)がどのように使われるのかを、実際に構築や管理をしながら、学習体験をします。



また、本自習書ではコンテナホストが不要でコンパクトなリリース、運用ができる Azure Container Instances (ACI) の利用法についても解説いたします。

目次

1　Azure Kubernetes Service (AKS) の概要

1.1 Azure Kubernetes Service (AKS) とは

AKS とは、マネージド型の Kubernetes です。Kubernetes については後述します。

AKS は、Kubernetes を用いたアプリケーションのスケーリング及びオーケストレーションが可能で、Kubernetes クラスタの管理の複雑さと運用上のオーバーヘッドを軽減します。
料金については、マスターノードへの課金は無く、利用したリソース(エージェントノードの VM やストレージ等）に対しての課金のみです。

AKS は、2018/05/15 現在、プレビュー版のみの提供で、対応リージョンについては、日本は無く、以下のみになります。

- 米国東部
- 西ヨーロッパ
- 米国中央部
- カナダ中部
- カナダ東部

AKS は次の機能を提供しています。

- 自動的な Kubernetes のバージョンのアップグレードと修正プログラムの適用
- 簡単なクラスタースケーリング
- ホストされているコントロール プレーン (マスター) の自己修復
- 実行しているエージェント プール ノードのみの課金によるコストの削減
- 標準の Kubernetes API エンドポイントの公開で、kubectl、helm、draft などの Kubernetes クラスタと通信できる任意のソフトウェアで通信可能
- AKS の使用には、Azure CLI、または、ポータル(Marketplace で Azure Kubernetes Service を検索)を経由することで、AKS クラスタをデプロイ可能

1.2 AKS のメリット

AKS は、Kurbenetes で必要であった、以下の運用管理作業の負荷を軽減します。

- Kubernetes クラスタの管理(アップグレード、修正プログラムの適用など)
- クラスタ内のノードの管理
- AKS サービスの正常性監視

そのため、AKS のクラスタへの直接的なアクセス(SSH など)は提供されていません。

1.3 Azure Container Instances (ACI) との違い

ACI は次の特徴を持ちます。

- コマンド 1 つでコンテナの起動からアプリケーションの公開までを実施
- サーバー（コンテナホスト）を意識する必要がなく、安価

AKS と異なりクラスタースケーリングができませんが、反対にクラスターに関する細かい設定が不要ということです。
とにかく早く簡単に環境を公開したい場合に向いていきます。本書では最後に ACI の活用法を示します。

1.4 Kubernetes について

1.4.1 Kubernetes とは

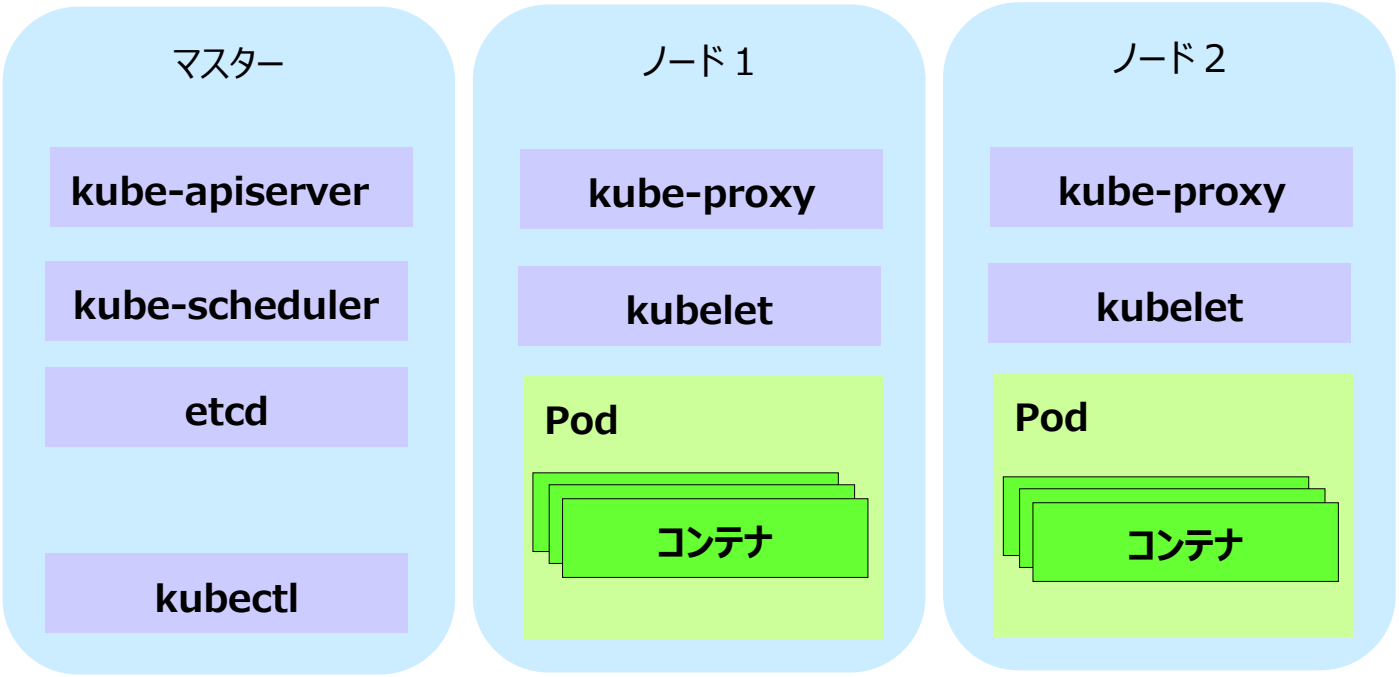
Kubernetes とは、Docker を始めとするコンテナを管理するオープンソース・ソフトウェアです。

以下の機能を実現可能です。

- 簡単なコマンドだけでノード数の増減ができるので、急なリソース増強にも容易に対応できる。
- Rolling Update(無停止更新)により、ユーザー影響を無し、もしくは最小限にしてサービスをリリースできる。
- Self-Healing(自己回復)により、障害などで、クラスタの状態に変化(コンテナが落ちるなど)があっても、元の状態を維持することができる。

1.4.2 Kubernetes の構成

Kubernetes は Master と Node の複数台の構成になります。Node は実際にコンテナが動作するマシンで、その Node たちの状態を Master が管理しています。さらに kube-proxy や API Server という複数のプロセスが連携して動作する複雑な構成となっています。
以下に、Kubernetes の構成するコンポーネントの図を示します。



構成するコンポーネントについて、次の表にまとめます。

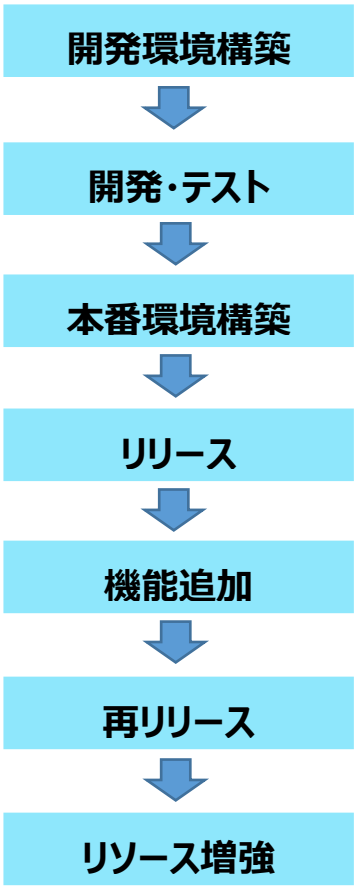
構成コンポーネント名	説明
kube-apiserver	以下に説明する各種プロセス(kube-proxy や kube-schedular)からの、Rest による要求により、Kurbernetes のリソース情報(Pod に関する情報や Cluster IP など)を返します。各種プロセスはこの kube-apiserver を通じて、リソース情報のやり取りを行います。
kube-scheduler	各 Node の空き状況や状態を API Server を通じて kubelet に伝える役割を持っています。この情報をもとに kubelet はコンテナを生成します。
etcd	kubernetes のリソース情報を保存する高信頼分散 KVS です。etcd は、kube-apiserver のデータストアであり、各種プロセスから受け取ったリソース情報を etcd に情報を保存します。また、flannel(Pod 間の通信を実現する VXLAN 管理サービス)も自身のデータストアとして、etcd を利用します。
kubectl	Kubernetes を管理するためのコマンドラインツールです。クラスタの作成や構成の変更など、Kubernetes の管理に必要な操作は全てこの kubectl を用いて行います。内部的には、kube-apiserver に Rest でリクエストを行っています。
kube-proxy	Cluster IP のルーティングとロードバランシングを行います。Cluster IP とは、サービスにアクセスするための代表的な IP アドレスになります。定期的に API Server 経由で Kubernetes のリソース情報にアクセスし、必要に応じて iptables のルールを作成します。この IP アドレスにアクセスすると、そのリクエストは適切な Pod に接続することができます。
kubelet	Pod の生成や停止を行います。定期的に API Server に問い合わせて、生成や停止の数やタイミングを決めています。
Pod	Kubernetes のデプロイの単位になります。Pod は複数のコンテナをまとめる論理的な単位です。

2　本自習書の進め方

本自習書では、実際のアプリケーション開発フローの中で、Kubernetes がどのように使われるのか、実際に Kubernetes の構築や管理をしながら、説明します。

2.1 開発フロー

Kubernetes がかかわってくるのは設計フェーズの後のため、ここでは要件定義、基本設計、詳細設計などの設計フェーズは終了している前提とします。以下の開発フローに基づき、各フェーズにおける AKS の利用方法を記載します。



本開発フローで登場する人物は、アプリケーションアーキテクトとプログラマの 2 人になります。

本開発フローでのそれぞれの役割は以下とします。

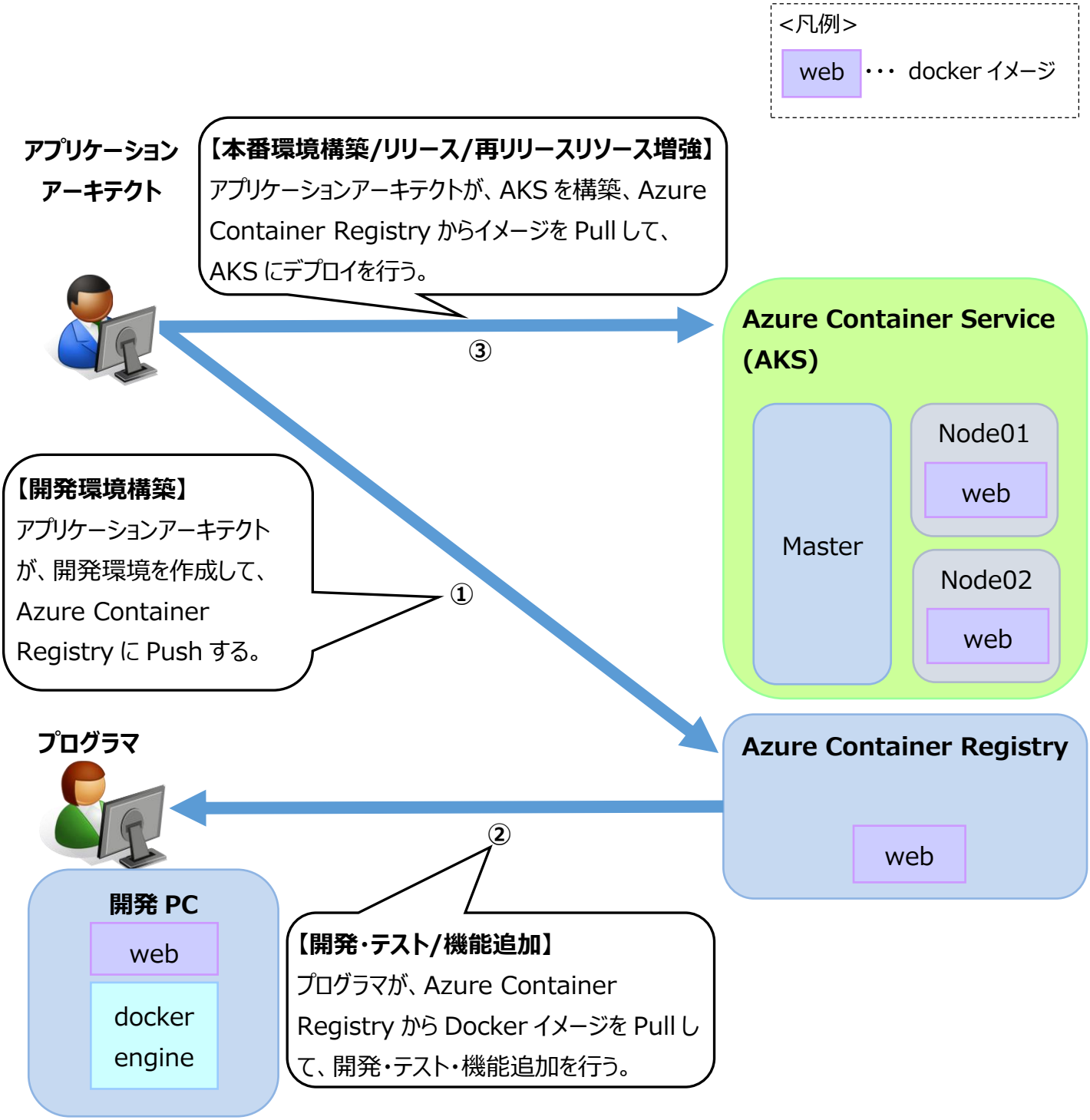
- アプリケーションアーキテクト
  リリースできる品質のインフラとアプリケーションの原型の作成、本番リリース、本番環境構築とリソースの増強を行います。
- プログラマ
  詳細設計書に従ってコードの実装とテストをいます。

2.2 開発フローの各フェーズ

各フェーズの作業内容について、次の表にまとめます。

フェーズ	担当	説明
開発環境構築	アプリケーション アーキテクト	これから複数のプログラマにコーディングを行ってもらうために、プログラマ専用の開発環境を用意します。Java の開発で言えば、Eclipse で開発環境構築したり、Git リポジトリ作成したりです。一般的には、インフラとアプリケーションの両方に精通するエンジニア(以降、アプリケーションアーキテクトと呼称)が担当することが多いです。 この作業は、今回で言えば、開発環境専用の Docker イメージを作成、事前に構築した Azure Container Registry(後述)に Push するといった作業になります。
開発・テスト	プログラマ	プログラマが、事前に用意された開発環境を使ってコーディングとテストを繰り返す作業です。 この作業は、今回で言えば、先述の Azure Container Registry から開発環境専用の Docker イメージを Pull して、ローカル PC でコーディングをするといった作業になります。
本番環境構築	アプリケーション アーキテクト	開発したアプリケーションを動作させるための本番環境を構築します。 今回の要である Azure Kubernetes Service(AKS)の構築になります。
リリース	アプリケーション アーキテクト	「本番環境構築」のフェーズで構築した環境に、Azure Container Registry から Pull したイメージをデプロイするだけで、簡単に本番環境が出来上がります。ここでのポイントは、開発環境と本番環境で同じ Docker イメージを利用していることです。これにより、Docker を使う以前のレガシーな開発でよくあった「開発環境では動いたのに、本番環境では動かない」という問題は無くなります。
機能追加	プログラマ	エンドユーザーからの追加要望が発生したことに伴い、機能追加をするフェーズになります。
再リリース	アプリケーション アーキテクト	プログラマが機能追加したアプリケーションを Azure Container Registry に Push、Azure Kubernetes Service(AKS)にデプロイします。ここでのポイントは、Azure Kubernetes Service(AKS)は Rolling Update つまりアプリケーションを無停止で更新が可能ということです。エンドユーザーの業務に影響なく、アプリケーションをリリースできるというのは運用上非常に大きなメリットがあります。
リソース増強	アプリケーション アーキテクト	アプリケーションの利用者が増え、レスポンスが遅くなったことへの対応策として、リソースを増強するケースを想定しています。Azure Kubernetes Service(AKS)には、ReplicaSet の概念があり、設定ファイルで ReplicaSet を増やすだけで、簡単にスケールアウトが可能です。指定した ReplicaSet の数を維持するよう、Pod が各ノードに立ち上がります。

上記の開発フローのイメージを図にすると以下になります。

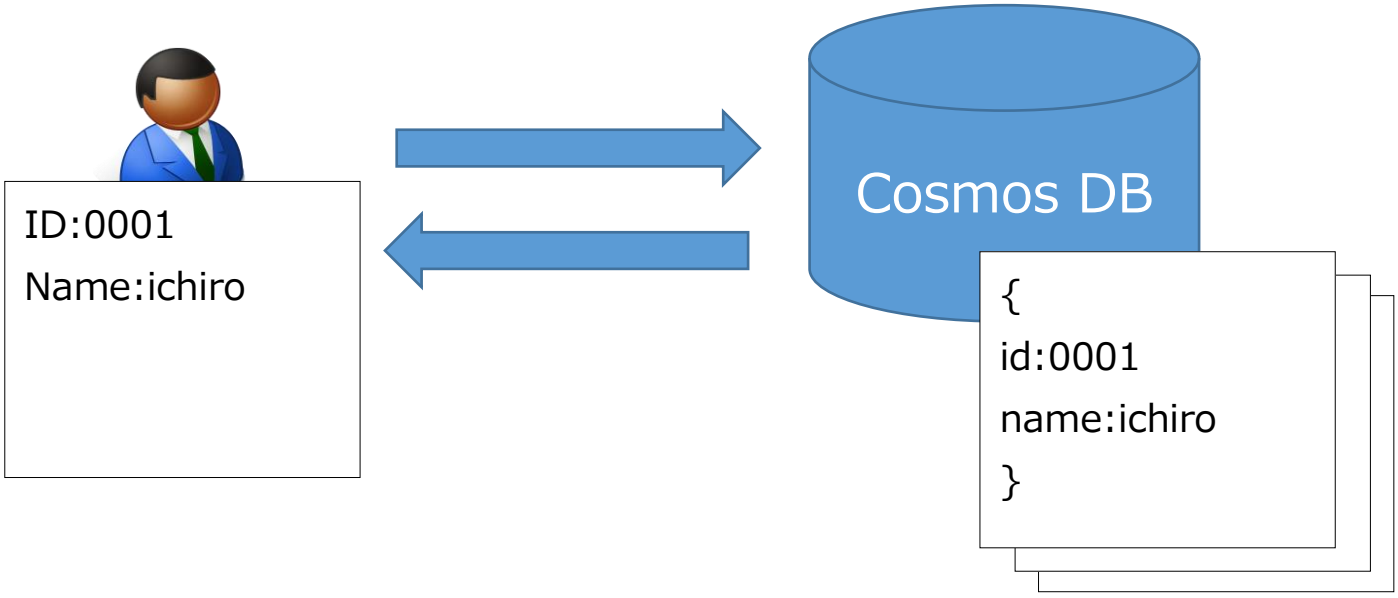


3　本自習書で開発するアプリケーションシステム

3.1 開発するアプリケーションについて

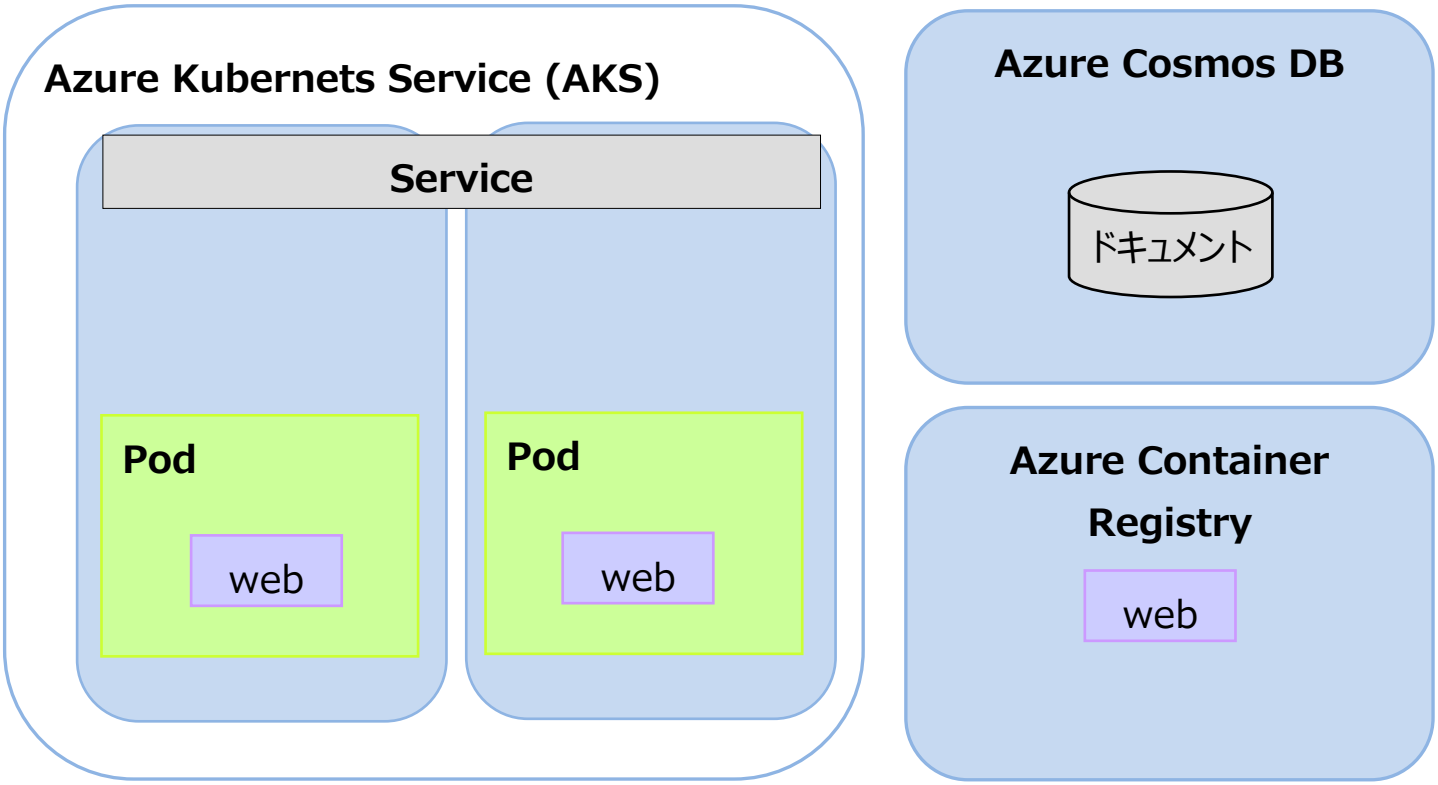
開発フローの中で開発するアプリケーションは、Azure の Cosmos DB に作成された JSON 形式のドキュメント型のデータを表示する PHP を作成します。
開発するアプリケーションが動作する OS は Linux 前提とします。

開発するアプリケーションの構成図を以下に示します。Azure の Cosmos DB から取得したデータを、Web ブラウザに表示する仕様になります。



3.2 構築する環境について

Azure 上に以下の環境を構築します。



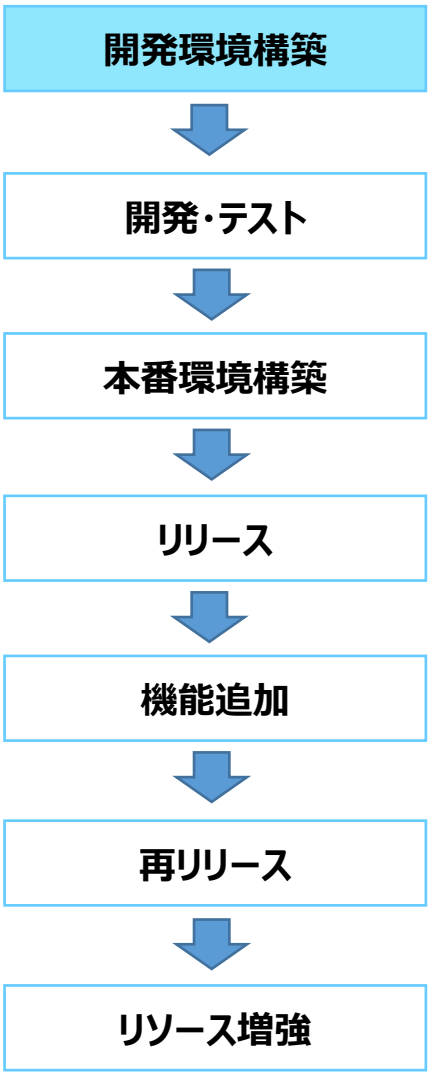
構築する各環境について、次の表にまとめます。

環境	説明
Azure Kubernetes Service (AKS)	AKS は、Azure Container Registry から Pull して Docker イメージをデプロイします。
Azure Container Registry	Docker の Private Registry になります。AKS は、Registry 上の Docker イメージを Pull して、デプロイを行い、Node 上にコンテナを配置します。
Azure Cosmos DB	前述した自習書で作成する PHP のアプリケーションと連携するドキュメント型のクラウド DB になります・

以降の章にて、開発フローに基づき、AKS を利用していきます。

4　開発環境構築

アプリケーションアーキテクトが開発環境構築を行うフェーズです。



4.1 事前準備

アプリケーションアーキテクトが使用する開発 PC の環境として、必要な事前準備を行います。

開発する PC の OS には、CentOS7 を使用します。

本自習書では、Azure への操作は Azure CLI を使います。
Azure CLI は、バージョン 2.0.27 以降を実行している必要があります。

バージョンを確認するには、以下のコマンドを実行します。

```
# az --version
```

インストールまたはアップグレードする必要がある場合は、以下の Azure CLI のインストールに関するページを参照してください。
https://docs.microsoft.com/ja-jp/cli/azure/install-azure-cli?view=azure-cli-latest

docker を使用します。
以下のコマンドを実行して、バージョンを確認できない場合は、インストールが必要です。

```
# docker version
```

インストールは、以下のページに従って行ってください。
https://docs.docker.com/install/linux/docker-ce/centos/#set-up-the-repository

docker-compose も使用します。
以下のコマンドを実行して、バージョンを確認できない場合は、インストールが必要です。

```
# docker-compose --version
```

インストールは、以下のページに従って行ってください。
https://docs.docker.com/compose/install/

4.2 Docker コンテナイメージ用プライベートレジストリの作成

4.2.1 Azure Container Registry の作成

Docker コンテナイメージ用のプライベート レジストリとして、Azure Container Registry を作成します。

Azure Container Registry を作成する場合は、まず、Azure リソースグループが必要です。
Azure リソースグループとは、関連するリソースを 1 つに括った論理的なコンテナーです。

az group create コマンドでリソース グループを作成します。
本自習書では、eastus リージョンに rgAksSelfstudy という名前のリソースグループを作成します。

```
# az group create --name rgAksSelfstudy --location eastus
```

az acr create コマンドを使用して、acrAksSelfStudy の名前で Azure Container Registry を作成します。

```
# az acr create --resource-group rgAksSelfStudy --name acrAksSelfStudy --sku Basic
```

4.2.2 Docker イメージの作成

開発環境用の Docker イメージを作成します。

以下の Dockerfile を作成します。本自習書では簡易的なアプリケーションの作成としているため、ここでは単純にします。

```
#1
FROM php:7.0-apache
#2
RUN pear install http_request2
```

#1 では、Docker Hub から、Apache 上で PHP が動作する Docker イメージを Pull します。

#2 では、PHP のパッケージ管理システム pear により、HTTP ライブラリである http_request2 をインストールします。こちらのライブラリは、別途使用する Cosmos DB へアクセスするための PHP ライブラリから呼び出されているために必要となります。

作成した Dockerfile からイメージを作成します。

```
# docker build -t web .
```

4.2.3 Docker イメージの Azure Container Registry への Push

az acr login コマンドで ACR インスタンスにログインします。

```
# az acr login --name acrAksSelfStudy
```

docker images コマンドで、現在のイメージ一覧を表示し、イメージのタグ名を確認します。

```
# docker images
```

Push する Docker イメージは、Azure Container Registry のログインサーバ名でタグ付けする必要があります。
このタグは、Azure Container Registry に Docker イメージを Push するときに、ルーティングするために使用されます。

Azure Container Registry のログインサーバ名を取得するには、次のコマンドを実行します。

```
# az acr list --resource-group rgAksSelfStudy --query "[].{acrLoginServer:loginServer}" --output
table
AcrLoginServer
-------------------------
acraksselfstudy.azurecr.io
```

取得した Azure Container Registry のログインサーバ名で、Push する Docker イメージにタグ名を付けます。
下記のとおり、イメージ名は、必ず、Azure Container Registry のログインサーバ名/イメージ名:タグ名である必要があります。

```
# docker tag web acraksselfstudy.azurecr.io/web:dev
```

タグを付けた後、docker images を実行してタグを確認します。
次の実行結果のとおり、「REPOSITORY」欄に「acraksselfstudy.azurecr.io/web」、「TAG」欄に「dev」と表示される行があれば、タグ付けは成功です。

```
# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED
SIZE
acraksselfstudy.azurecr.io/web      dev                 fba1b833071a        22 hours ago
387MB
web                                 latest              fba1b833071a        22 hours ago
387MB
php                                 7.0-apache          2c1b7b71162b        9 days ago
386MB
```

Azure Container Registry に Docker イメージをプッシュします。完了するまでに数分かかります。

```
# docker push acraksselfstudy.azurecr.io/web:dev
```

Azure Container Registry にプッシュされたイメージを確認するには、az acr repository list コマンドを使用します。

```
# az acr repository list --name acrAksSelfStudy --output table
Result
--------
web
```

さらに、特定のイメージのタグを表示するには、az acr repository show-tags コマンドを使用します。

```
# az acr repository show-tags --name acrAksSelfStudy --repository web --output table
Result
--------
dev
```

4.3 Azure Cosmos DB へアクセスするための PHP ライブラリの準備

以下の GitHub で公開されている Azure Cosmos DB へアクセスするための PHP ライブラリを git clone して使用します。
https://github.com/cocteau666/AzureDocumentDB-PHP
Cosmos DB へのアクセスは Rest インターフェースが用意されていますが、それらをラッパーして作られた PHP のライブラリです。

こちらの PHP ライブラリをそのまま使用すると、HTTP クライアントライブラリの http_request2 が SSL 接続する際に行っている証明書の検証で失敗します。
そのため、本自習書に限り、便宜上、一時的に無効にするよう phpdocumentdb.php の中の以下の箇所を変更します。

```
private function request($path, $method, $headers, $body = NULL)
{
$request = new Http_Request2($this->host . $path);
$request->setHeader($headers);
if ($method === "GET") {
$request->setMethod(HTTP_Request2::METHOD_GET);
} else if ($method === "POST") {
$request->setMethod(HTTP_Request2::METHOD_POST);
} else if ($method === "PUT") {
$request->setMethod(HTTP_Request2::METHOD_PUT);
} else if ($method === "DELETE") {
$request->setMethod(HTTP_Request2::METHOD_DELETE);
}
if ($body) {
$request->setBody($body);
}
$request->setConfig(array( // これを追加
'ssl_verify_peer' => false // これを追加
)); // これを追加
```

このソースをソースコード管理リポジトリ(Git や svn)に Push します。本自習書では、ソースコード管理についての説明は割愛させて頂きます

4.4 Azure Cosmos DB の作成

Azure Cosmos DB にドキュメントデータベースを作成します。

az cosmosdb create コマンドを使用して、Azure Cosmos DB アカウントを作成します。
アカウントの名前は cdbaksselfstudy、アカウントの種類は SQL API で作成します。アカウントの作成には数分かかります。

```
# az cosmosdb create --name cdbaksselfstudy --kind GlobalDocumentDB --resource-group
rgAksSelfStudy
```

az cosmosdb database create コマンドを使用して、testdb の名前でデータベースを作成します。

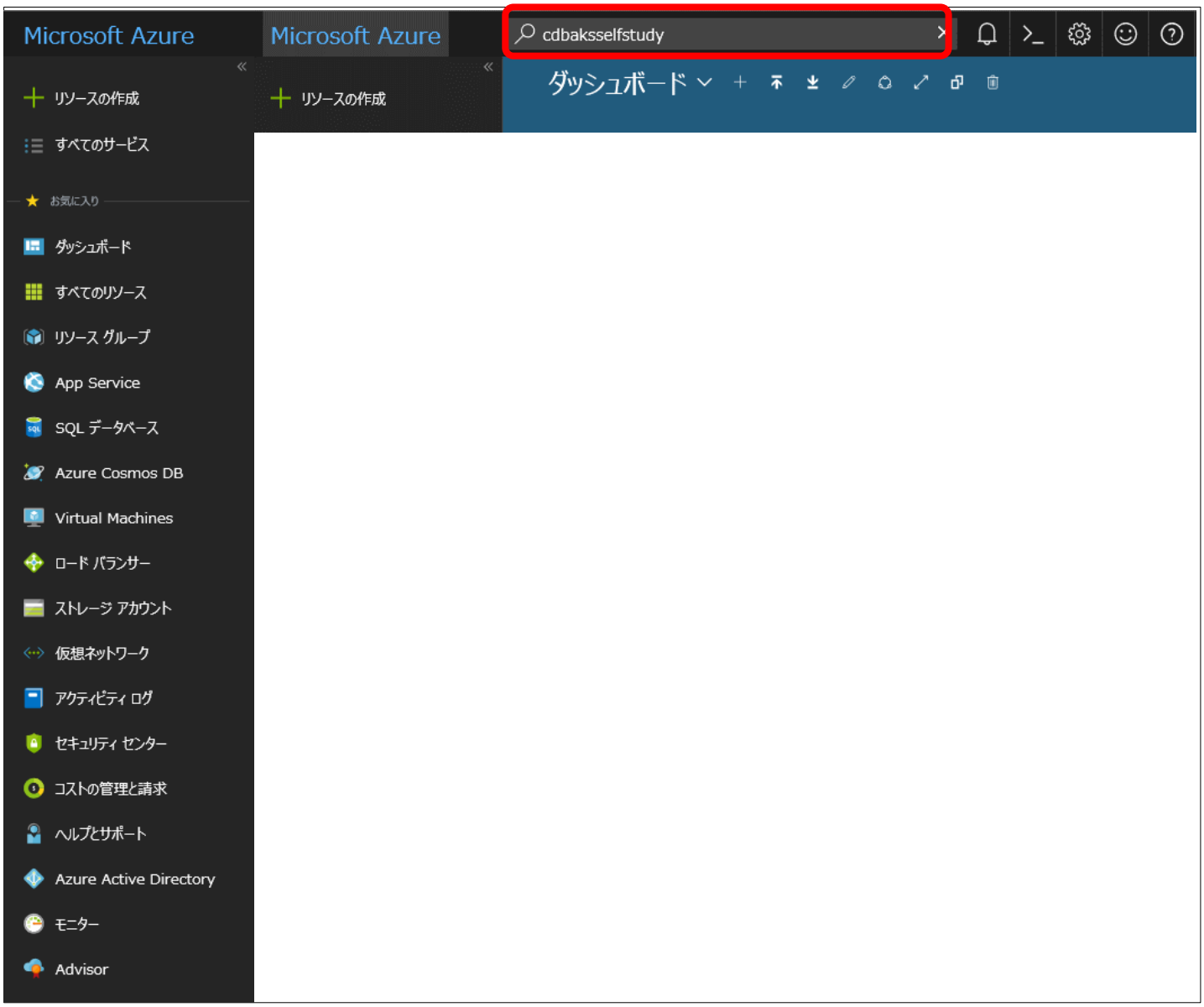
```
# az cosmosdb database create --name cdbaksselfstudy --db-name testdb --resource-group rgAksSelfStudy
```

az cosmosdb collection create コマンドを使用して、testcoll の名前でコレクションを作成します。

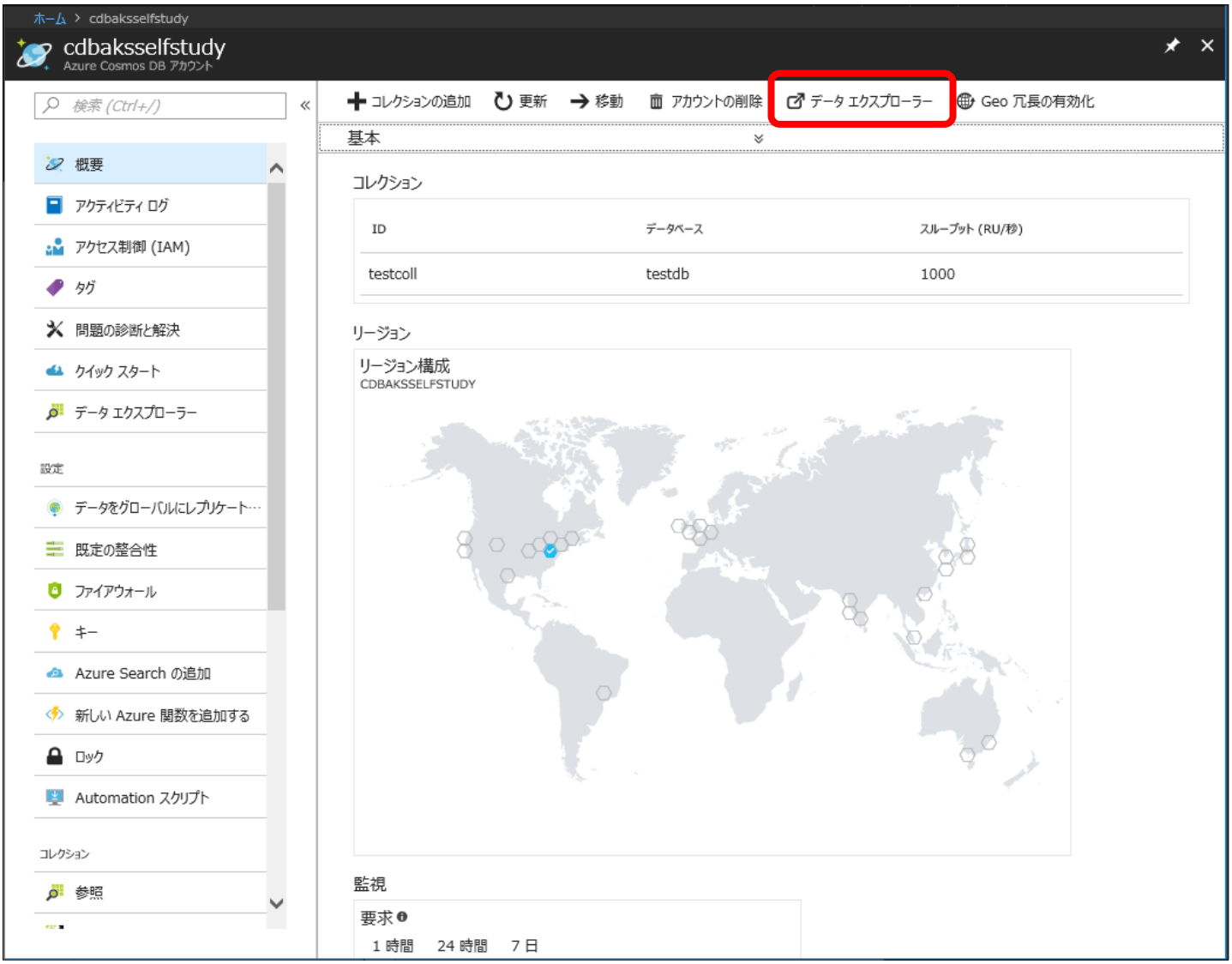
```
# az cosmosdb collection create --collection-name testcoll --name cdbaksselfstudy --db-name testdb --resource-group rgAksSelfStudy
```

ドキュメントを作成します。ドキュメントについては、Azure CLI では作成できないため、Azure ポータル画面から作成します。

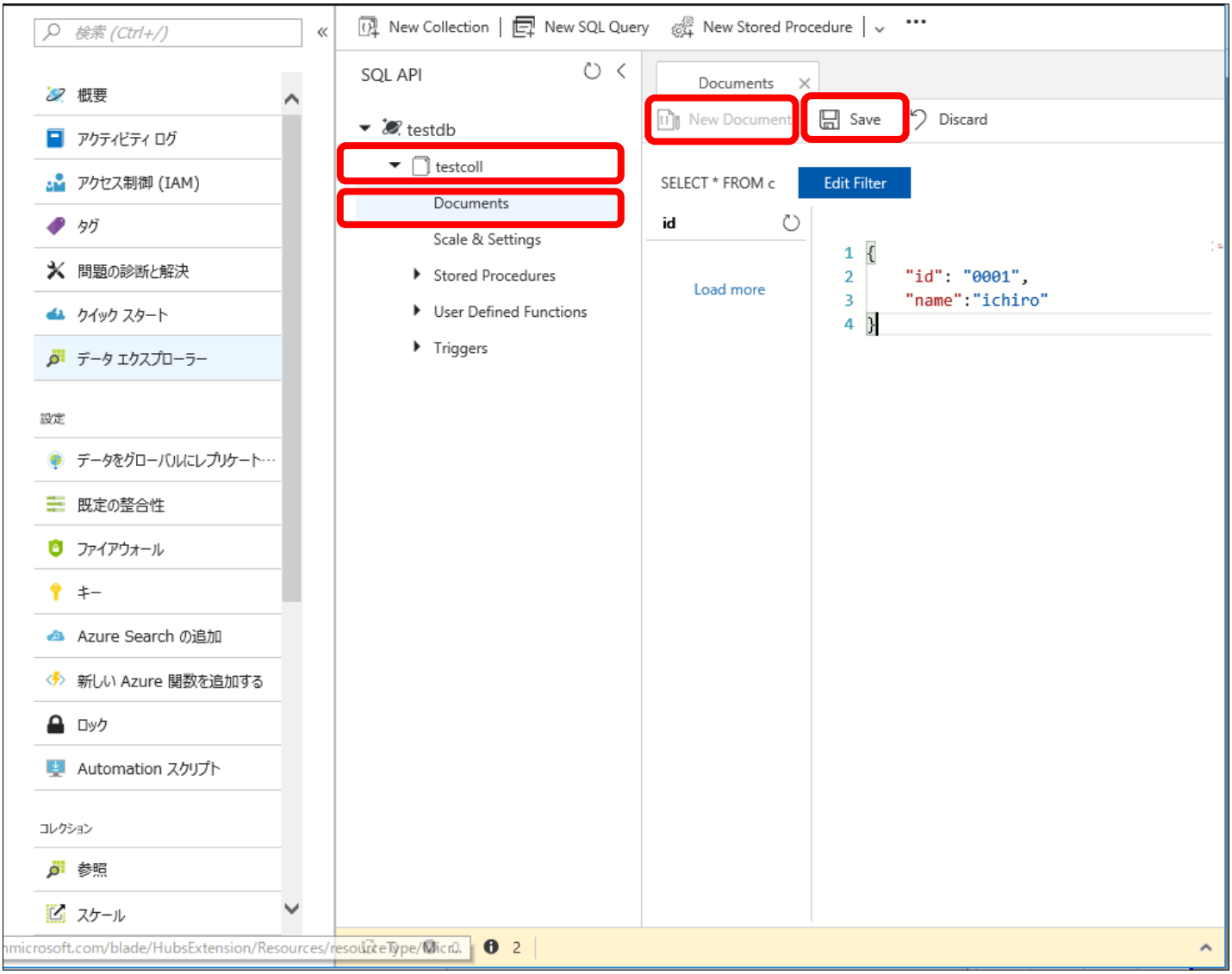
Azure ポータルにサインインした後、画面上部にある次の赤枠の検索ボックスに、Cosmos DB のアカウント名 cdbaksselfstudy を入力して検索します。



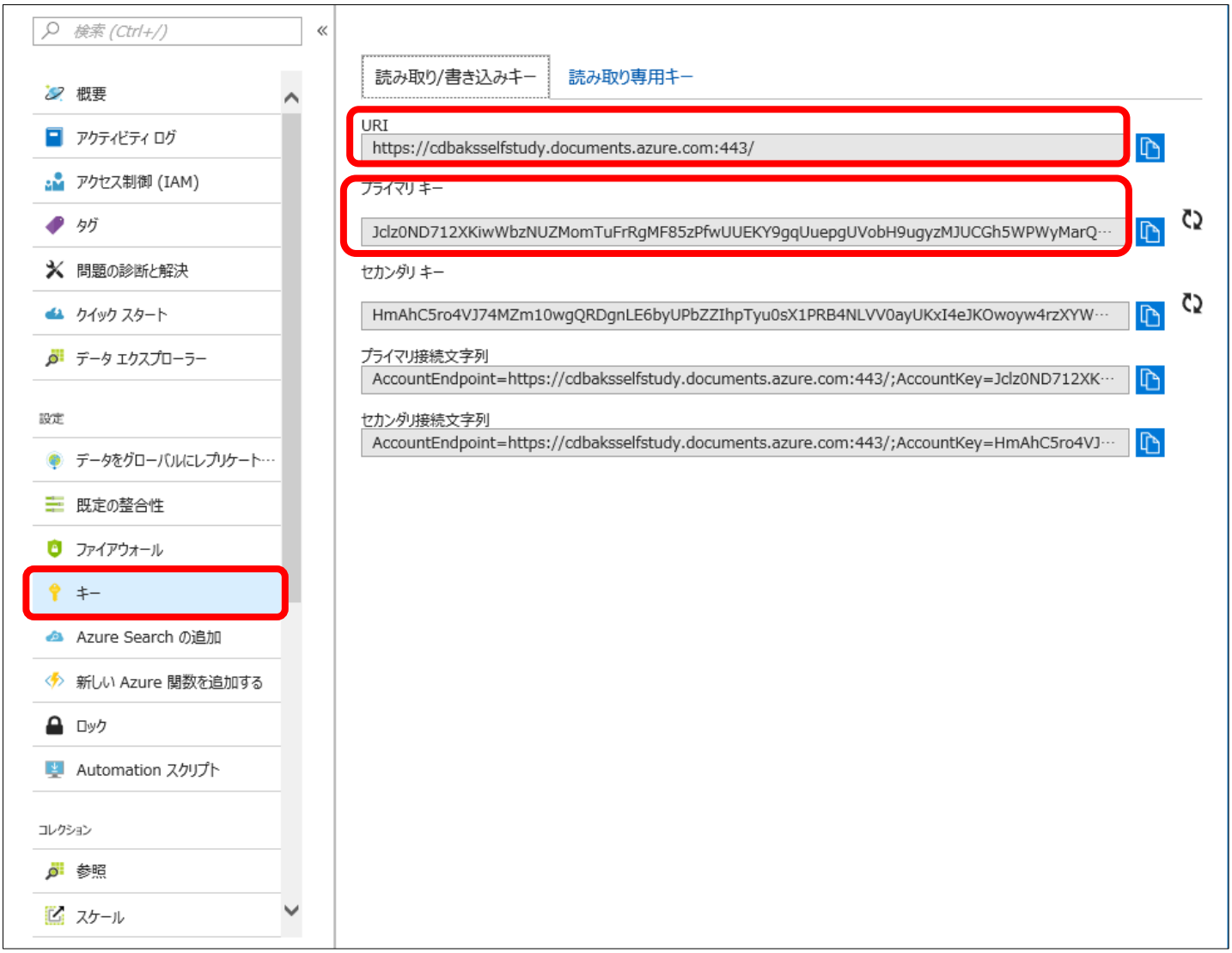
画面上部にある次の赤枠の[データエクスプローラー]をクリックします。



表示された画面で、次の赤枠のうち、[testcoll]→[Documents]→[New Document]の順にクリックします。
行数が表示されたエディタの箇所に、次の画面のとおり、JSON のデータを入力して、赤枠の「Save」をクリックします。
id と name だけのシンプルな JSON です。これでドキュメントの作成は完了です。

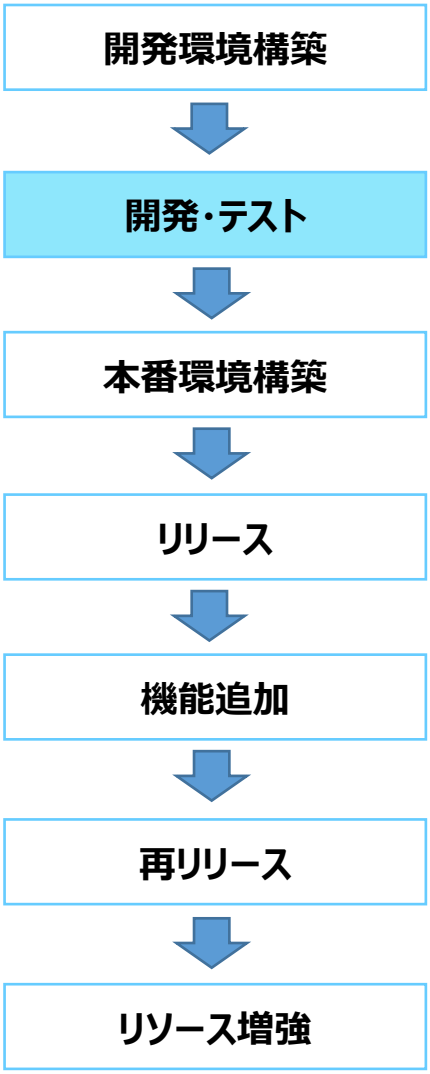


Cosmos DB への接続に必要な情報をプログラマに情報共有できるよう事前に取得します。
次の画面の赤枠の[キー]をクリックし、表示された赤枠部分の[URI]と[プライマリキー]を控えておきます。



5　開発・テスト

プログラマが開発・テストを行うフェーズです。



5.1 事前準備

プログラマが使用する開発 PC への事前準備は、先述 4.1 事前準備 と同様です。

5.2 開発

はじめに、開発 PC の任意のディレクトリで docker-compose.yml という名前のファイル名を作成します。
ファイルには、以下の内容を記述します。

```
web:
  image: acraksselfstudy.azurecr.io/web:dev # (1)
  container_name: web
  ports:
    - "80:80" # (2)
  volumes:
    - ./src:/var/www/html # (3)
```

docker-compose.yml の記載内容について、以下に補足します。

(1)は、Azure Container Registry に上げた開発用の Docker イメージを Pull しています。

(2)は、ホストの 80 番ポートからコンテナの 80 番ポートにポートフォワードしております。内部の Web サーバーにアクセスするためです。

(3)は、ホストの src ディレクトリをコンテナ内の/var/www/html にマウントしています。ホストの src ディレクトリに PHP のファイルを置けば、すぐに実行可能になるためです。ADD や COPY では、ソースを変更する度に、ビルドのし直しが発生するのでかなり不便になります。
なお、ホストの src ディレクトリは、実際の開発では、ソースコード管理リポジトリから取得してきたディレクトリで、ソースコード管理リポジトリに管理されます。したがって、先述 4.3 Azure Cosmos DB へアクセスするための PHP ライブラリの準備で用意した phpdocumentdb.php も存在します。

次に、phpdocumentdb.php も存在する src ディレクトリ配下に test.php という名前で以下の内容のファイルを作成します。

```
<?php
// Cosmos DB に接続するライブラリのパスを指定する
require_once './phpdocumentdb.php';
// DocmentDB のホスト名を指定する
$host = '<先述 4.3 Azure Cosmos DB の作成にて、[キー]設定を控えたうちの URI>';
// DocumentDB のマスターキーを指定する
$master_key = '<先述 4.3 Azure Cosmos DB の作成にて、[キー]設定を控えたうちのプライマリキー>';
// DocumentDB に接続する
$documentdb = new DocumentDB($host, $master_key,$debug = true);
// データベースを選択する
$db = $documentdb->selectDB("testdb");
// コレクションを選択する
$col = $db->selectCollection("testcoll");
// クエリを実行する
$result = $col->query("SELECT * FROM c");
// JSON をデコードする
$result_json = json_decode($result);
?>
<html>
<head>
<meta charset="UTF-8">
<title>CosmosDB</title>
</head>
<body>
ID:<?php echo $result_json->Documents[0]->id; ?><br>
Name:<?php echo $result_json->Documents[0]->name; ?>
</body>
</html>
```

5.3 テスト

5.3.1 コンテナイメージの作成

Docker Compose で、コンテナイメージの作成、およびアプリケーションの起動を行います。Docker Compose は、コンテナイメージの作成と複数コンテナのアプリケーションのデプロイを自動的に行います。

以下のとおり、予め Azure Container Registry にログインしてから、docker-compose.yaml ファイルを実行します。

```
# az acr login --name acrAksSelfStudy
# docker-compose up –d
```

docker images コマンドを使って、作成されたイメージを確認します。
次の実行結果のとおり、「REPOSITORY」欄に「acraksselfstudy.azurecr.io/web」、「TAG」欄に「dev」と表示される行があれば、イメージが作成されています。

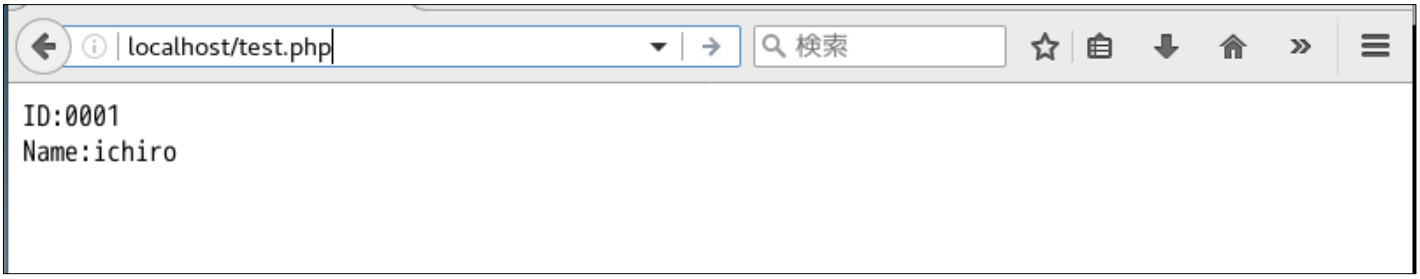
```
# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED
SIZE
acraksselfstudy.azurecr.io/web      dev                 72826e6076e9        16 hours ago
387MB
```

以下のとおり、docker ps コマンドを実行して、作成されたイメージからコンテナが実行していることを確認します。

```
# docker ps
CONTAINER ID        IMAGE                                COMMAND                  CREATED
STATUS              PORTS                    NAMES
1cfb034c1821        acraksselfstudy.azurecr.io/web:dev   "docker-php-entrypoi…"   11 minutes
ago         Up 11 minutes       0.0.0.0:80->80/tcp   web
```

5.3.2 ブラウザでの動作テストとコンテナのクリーンアップ

ブラウザで「http://localhost/test.php」にアクセスして、実行中のアプリケーションを確認します。
以下のとおり、Cosmos DB に登録したデータが表示されれば、正常動作しています。



テストにより正常動作を確認できたソースは、実際の開発では、ソースコード管理リポジトリにプッシュします。ソースコード管理リポジトリについては、本自習書では割愛します。

アプリケーションのテストが成功したら、不要になったコンテナとリソースを削除します。

以下のとおり、実行中のコンテナを停止します。

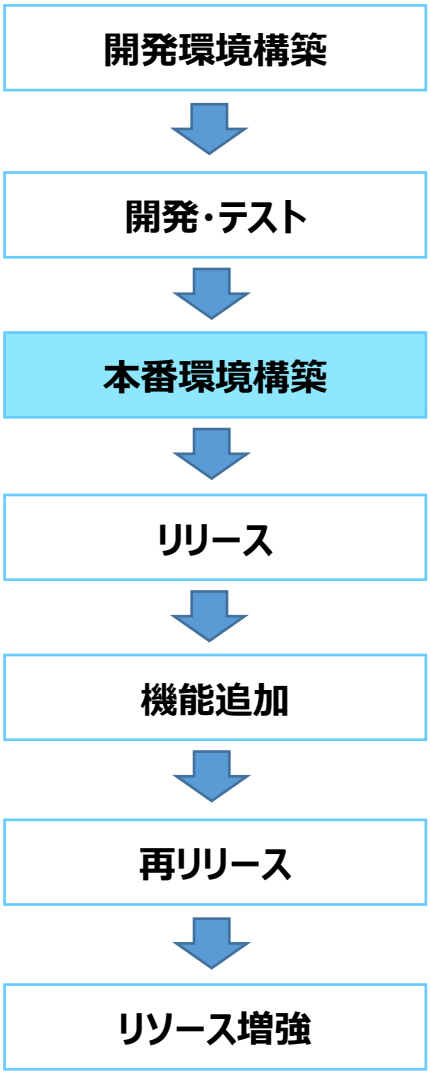
```
# docker-compose stop
```

以下のとおり、停止しているコンテナーとリソースを削除します。

```
# docker-compose down
```

6　本番環境構築

アプリケーションアーキテクトにより、Azure Kubernetes Service(AKS)の本番環境を構築するフェーズです。



6.1 AKS クラスタのデプロイ

AKS がプレビューである間、新しいクラスタを作成するには、サブスクリプションに機能を要求するための機能フラグが必要です。

次のとおり az provider register コマンドの実行で、AKS プロバイダの登録や AKS クラスタ作成でタイムアウトさせないためのプロバイダも登録します。

```
# az provider register -n Microsoft.ContainerService
# az provider register --namespace Microsoft.Compute
```

```
# az provider register --namespace Microsoft.Network
```

次のとおり、先述 4.2.1 Azure Container Registry の作成にて作成した rgAksSelfStudy という名前のリソースグループに, myAKSCluste という名前のクラスタを作成するコマンドを実行します。このコマンドは、作成する AKS クラスター用にサービスプリンシパルを作成します。

```
# az aks create --resource-group rgAksSelfStudy --name myAKSCluster --node-count 2 --generate-ssh-keys
```

以下のようなエラーが出力された場合は、Azure のログインユーザは、「全体管理者」に設定するなど、サービスプリンシパルの作成が可能な権限に設定することが必要です。

```
Directory permission is needed for the current user to register the application. For how to configure, please refer 'https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-create-service-principal-portal'. Original error: Insufficient privileges to complete the operation.
```

6.2 kubectl CLI のインストール

アプリケーションアーキテクトが使用する開発 PC から、Kubernetes クラスタに接続するには、kubectl(Kubernetes コマンドラインクライアント)を使用します。開発 PC ローカルにインストールするには、次のコマンドでインストールします。

```
# az aks install-cli
```

Kubernetes クラスタに接続するように kubectl を構成するには、次のコマンドを実行します。完了するまでしばらく待ちます。

```
# az aks get-credentials --resource-group=rgAksSelfStudy --name=myAKSCluster
```

次のとおり、クラスタへの接続を確認するには、kubectl get nodes コマンドを実行します。クラスタ作成コマンドで、ノード 2 個の指定で作成しているため、実行結果に、STATUS 欄が Ready、ROLES 欄に agent と表示される行が 2 行あることを確認します。

```
# kubectl get nodes
NAME                       STATUS   ROLES    AGE     VERSION
aks-nodepool1-44764324-0   Ready    agent    19m     v1.7.9
aks-nodepool1-44764324-1   Ready    agent    19m     v1.7.9
```

6.3 Azure Container Registry(ACR)認証の構成

AKS クラスタと ACR レジストリとの間で認証が構成されている必要があります。 その際、ACR レジストリからイメージをプルするための適切な権限を ACS の ID に付与します。

まず、次のとおり、AKS に対して構成されているサービスプリンシパルの ID を取得します。

```
# CLIENT_ID=$(az aks show --resource-group rgAksSelfStudy --name myAKSCluster --query "servicePrincipalProfile.clientId" --output tsv)
```

次に、次のとおり、ACR レジストリのリソース ID を取得します。

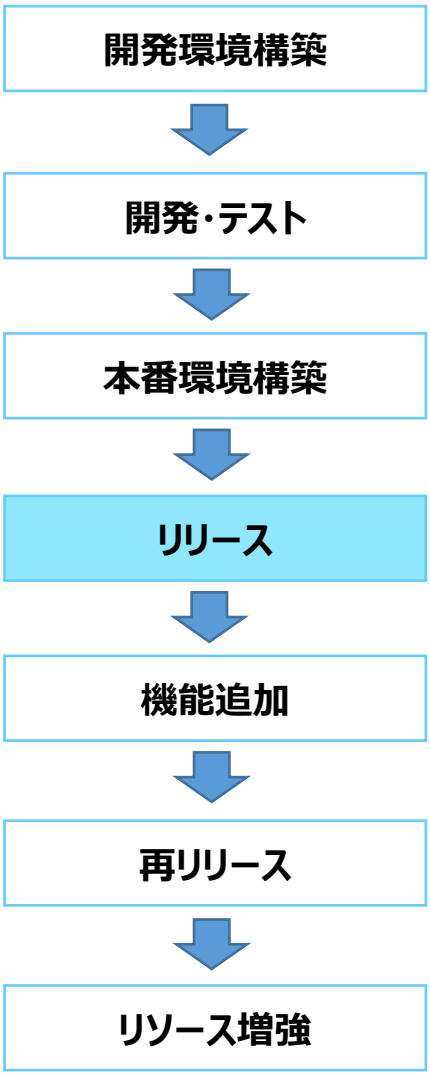
```
# ACR_ID=$(az acr show --name acrAksSelfStudy --resource-group rgAksSelfStudy --query "id" --output tsv)
```

最後に、適切なアクセス権を付与するロールの割り当てを、次のコマンドにより、作成します。

```
# az role assignment create --assignee $CLIENT_ID --role Reader --scope $ACR_ID
```

7　リリース

アプリケーションアーキテクトにより、作成したアプリケーションを本番環境にリリースするフェーズです。



7.1 リリースするコンテナイメージの作成

本番稼働用のイメージを作成します。任意のディレクトリで、以下の Dockerfile を作成します。

```
FROM acraksselfstudy.azurecr.io/web:dev

COPY ./src/ /var/www/html/
```

こちらは、先述にて、作成した開発用の Docker イメージを Pull して、src ディレクトリ配下にあるソースの phpdocumentdb.php と test.php を、コンテナイメージにコピーするという内容です。

こちらの Dockerfile から本番用のイメージをビルドし、Azure Container Registry に Push します。

```
# docker build -t acraksselfstudy.azurecr.io/web:1.0 .
# az acr login --name acrAksSelfStudy
# docker push acraksselfstudy.azurecr.io/web:1.0
```

本番用のイメージに、1.0 というタグを付けました。これは、このアプリケーションのリリースバージョンになります。以後、機能追加等でリリースをするたびに、このバージョン番号を増やしていきます。

本番用イメージに 1.0 が、ACR レジストリにプッシュされていることを、次のとおり、確認します。

```
# az acr repository list --name acraksselfstudy --output table
Result
--------
web
# az acr repository show-tags --name acraksselfstudy --repository web --output table
Result
--------
1.0
dev
```

7.2 AKS クラスタへの本番用コンテナイメージのデプロイ

作成したコンテナイメージを AKS にデプロイする作業を行います。

AKS にデプロイするために、Kubernetes のマニフェストファイルを作成します。アプリケーションアーキテクトが使用する開発 PC の任意のディレクトリに、次の内容で、ファイル名 deploy.yml として作成します。

```
apiVersion: apps/v1beta1
kind: Deployment   # (1)
metadata:
  name: web-deployment # (2)
spec:
  replicas: 2 # (3)
  template:
    metadata:
      labels:
        app: web # (4)
    spec:
      containers:
      - name: web # (5)
        image: acraksselfstudy.azurecr.io/web:1.0 # (6)
        ports:
        - containerPort: 80 # (7)
---
apiVersion: v1
kind: Service # (8)
metadata:
  name: web # (9)
spec:
  type: LoadBalancer # (10)
  ports:
```

```
  - port: 80 #(11)
  selector:
    app: web # (12)
```

Kubernetes のマニフェストファイルの内容について、補足します。

(1)は、設定の種別を表します。この設定は、デプロイするための設定なので、Deployment を記述します。

(2)は、このデプロイの設定を一意に識別するためのキーになります。

(3)は、Pod を起動する数です。この値に基づき、Pod が各 Node に作成されます。リリース当初は最小限に指定して、適宜、利用者のアクセスが増え次第、この値を変更していきます。

(4)は、このデプロイを適用するコンテナのラベルになります。(5)でコンテナのラベルを定義していますが、ここで定義したラベルのコンテナがデプロイ対象ということになります。

(5)は、Pod 内に作成するコンテナのコンテナ名です。

(6)は、Pod 内に作成するコンテナのイメージです。先述にて、作成したアプリケーションのイメージのバージョン 1.0 を指定していることがわかります。

(7)は、Pod 内で開放するポート番号です。HTTP 経由でアクセスさせるので、80 を指定しています。

(8)は、種別を指定しています。ここでは Service を定義しますので、Service とします。

(9)は、この Service を一意に識別するための名称です。任意の名前を指定します。

(10)は、Service のタイプを定義します。「LoadBalancer」と指定します。

(11)は、外部からアクセスできるように公開するポートになります。

(12)は、ラベルを指定しています。ここで指定したラベルが Service 経由でアクセスする対象の Pod になります。デプロイする時に Pod に「app: web」というラベルを指定したので、その値をここで指定します。

次のとおり、kubectl create コマンドを実行すると、作成したマニフェストファイルに従って、AKS にリソースの作成とデプロイを行います。

```
# kubectl create -f deploy.yml
deployment "web-deployment" created
service "web" created
```

アプリケーションをインターネットに公開する AKS の Service が作成されます。これにはしばらく時間がかかります。
次のとおり、進行状況を監視するには、kubectl get service コマンドに--watch 引数を指定して実行します。

```
# kubectl get service web --watch
NAME      TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
web       LoadBalancer   10.0.252.237   <pending>      80:32228/TCP   13s
web       LoadBalancer   10.0.252.237   52.170.113.23  80:32228/TCP   4m
```

web サービスの EXTERNAL-IP 欄は"保留中"で表示されます。しばらくすると、EXTERNAL-IP 欄が「保留中」から「IP アドレス」に変わったら、CTRL-C を使用して kubectl ウォッチ プロセスを停止します。

さらに、次のとおり、kubectl get pods コマンドで、Pod が 2 個作成されていて、2 個とも Status 欄が「Running」となっていることを確認します。

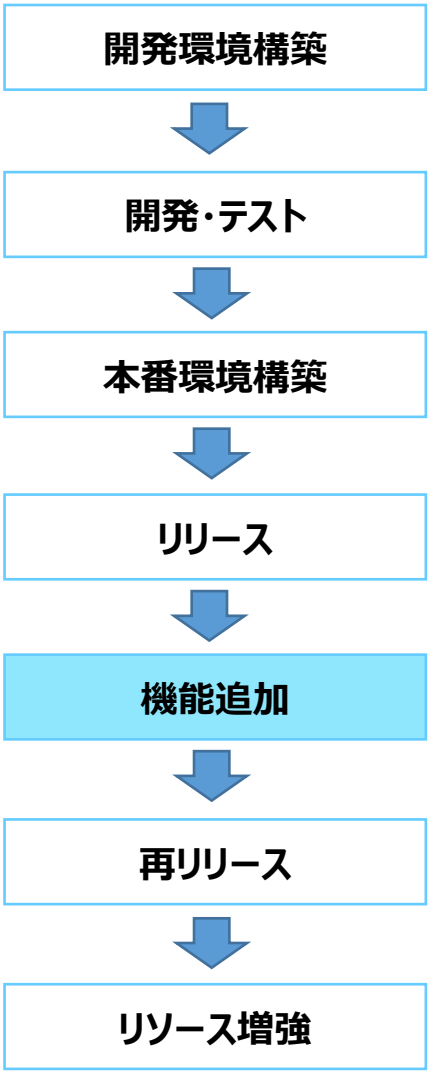
```
# kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
web-deployment-1486729925-7tq7r     1/1       Running   0          16m
web-deployment-1486729925-md3t9     1/1       Running   0          16m
```

EXTERNAL-IP 欄に IP アドレスが表示されたら、その IP アドレスでブラウザから表示確認します。
次の画面が表示されれば、デプロイが正常に実行されているので、リリースは完了です。

アプリケーションが表示されなかった場合は、AKS クラスタと ACR レジストリとの認証に関する構成が原因になっている可能性があります。

8　機能追加

プログラマが、ユーザーから要望があり、作成したアプリケーションに対して機能追加するフェーズです。



8.1 PHP ソースの変更

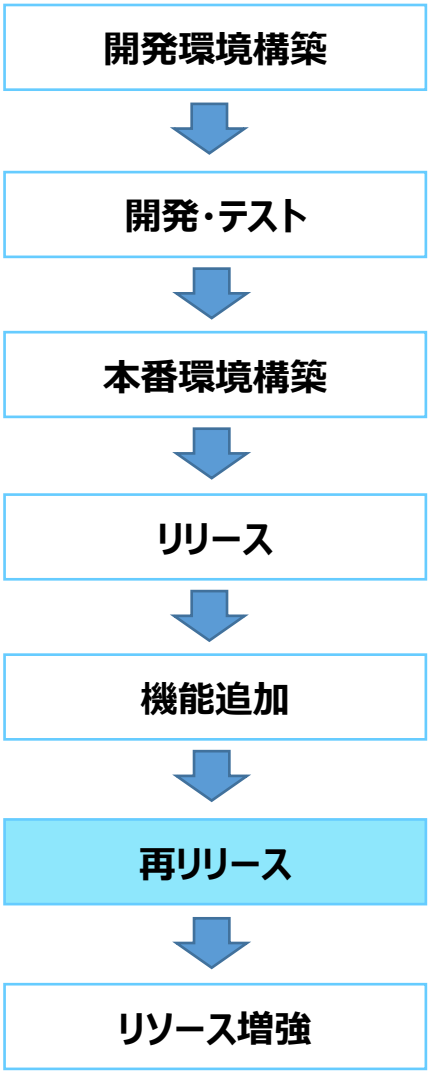
test.php を、以下のとおり、アプリケーションのバージョンを表示するよう、「//機能追加」がある行を追加します。

```
<?php
// Cosmos DB に接続するライブラリのパスを指定する
require_once './phpdocumentdb.php';
// DocmentDB のホスト名を指定する
$host = 'https://cdbaksselfstudy.documents.azure.com:443/';
// DocumentDB のマスターキーを指定する
$master_key =
'Jclz0ND712XKiwWbzNUZMomTuFrRgMF85zPfwUUEKY9gqUuepgUVobH9ugyzMJUCGh5WPWyMarQ28
WFA6hCoCQ==';
// DocumentDB に接続する
$documentdb = new DocumentDB($host, $master_key,$debug = true);
// データベースを選択する
$db = $documentdb->selectDB("testdb");
// コレクションを選択する
$col = $db->selectCollection("testcoll");
// クエリを実行する
$result = $col->query("SELECT * FROM c");
// JSON をデコードする
$result_json = json_decode($result);
?>
<html>
<head>
<meta charset="UTF-8">
<title>CosmosDB</title>
</head>
<body>
<?php echo "v2.0"; //機能追加 ?><br>
ID:<?php echo $result_json->Documents[0]->id; ?><br>
Name:<?php echo $result_json->Documents[0]->name; ?>
</body>
</html>
```

こちらを、実際の開発では、ソースコード管理システム(Git や svn)に Push します。ソースコード管理リポジトリについては、本自習書では、割愛します。

9　再リリース

アプリケーションアーキテクトにより、機能追加したアプリケーションを本番環境にリリースするフェーズです。



先述の 7 リリースにおいて、リリースした方法と基本的には変わりません。

本番用のイメージをビルドし、ACR レジストリにアップロードします。機能追加したファイルの test.php を取得して、以下のコマンドを実行します。

```
# docker build -t acraksselfstudy.azurecr.io/web:2.0 .
# docker push acraksselfstudy.azurecr.io/web:2.0
```

タグをバージョン 2.0 に変更しています。これで、バージョン 2.0 のタグがついた Docker イメージが ACR レジストリに Push されます。

ACR レジストリにプッシュされたことを、次のとおり、確認します。

```
# az acr repository list --name acraksselfstudy --output table
Result
--------
web
# az acr repository show-tags --name acraksselfstudy --repository web --output table
Result
1.0
2.0
dev
```

次に、作成した本番用のイメージを AKS にデプロイします。
次のとおり、Kubernetes のマニフェストファイルを再作成しないでも、kubectl set コマンドで簡単に機能追加したアプリケーションを更新できます。

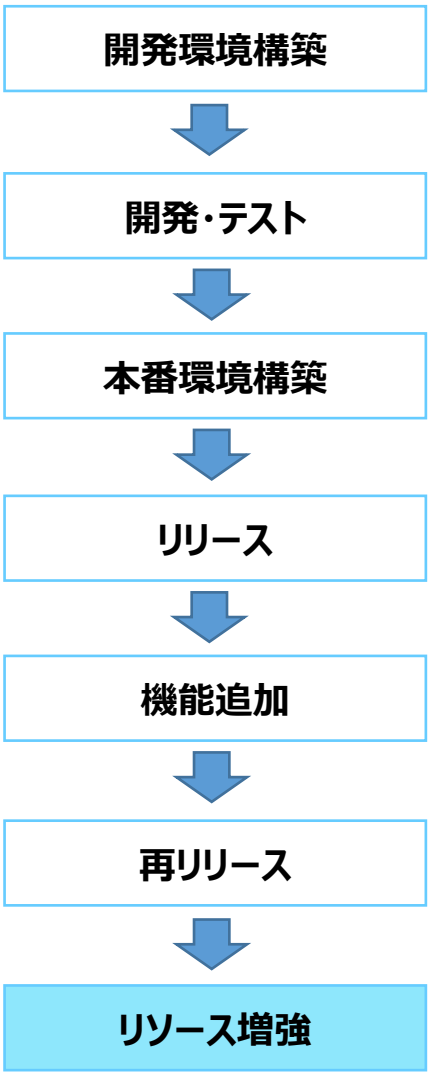
```
# kubectl set image deployment web-deployment web=acraksselfstudy.azurecr.io/web:2.0
deployment "web-deployment" image updated
```

kubectl get services コマンドで IP アドレスを再確認し、その IP アドレスでブラウザから表示確認します。
v2.0 が冒頭に表示される次の画面が表示されれば、デプロイが正常に実行されているので、再リリースは完了です。

このとき、Rolling Update(無停止更新)が行われており、サービスの停止なく、リリースができます。

10 AKS リソースの増強

アプリケーションアーキテクトにより、本番環境の AKS リソースを増強するフェーズです。例えば、サービスの利用者が増え、レスポンスが遅くなってきた場合、ノードの数や Pod の数を増やすことで、リソースを増強し、対応します。



10.1 AKS クラスタのノードのスケーリング

AKS クラスタ作成時は、ノード 2 個でしたが、ノード 3 個にスケーリングします。

次のコマンドで、myAKSCluster という名前の AKS クラスタのノードの数を 3 に増やしています。 コマンドが完了するまでにしばらく時間がかかります。

```
# az aks scale --resource-group=rgAksSelfStudy --name=myAKSCluster --node-count 3
```

コマンド出力結果に含まれる"agentPoolProfiles"の内容で「count」の値が「3」となります。

こちらの"agentPoolProfiles"の内容を以下に抜粋します。

```
  "agentPoolProfiles": [
    {
      "additionalProperties": {},
      "count": 3,
      "dnsPrefix": null,
      "fqdn": null,
      "name": "nodepool1",
      "osDiskSizeGb": null,
      "osType": "Linux",
      "ports": null,
      "storageProfile": "ManagedDisks",
      "vmSize": "Standard_DS1_v2",
      "vnetSubnetId": null
    }
```

また、kubectl get nodes コマンドでも、次のとおり、ノードが 3 個にスケーリングしていることを確認できます。

```
# kubectl get nodes
NAME                       STATUS    ROLES     AGE       VERSION
aks-nodepool1-44764324-0   Ready     agent     5h        v1.7.9
aks-nodepool1-44764324-1   Ready     agent     5h        v1.7.9
aks-nodepool1-44764324-2   Ready     agent     14m       v1.7.9
```

10.2 Pod のスケーリング

ここまでで、コンテナデプロイ時、Pod2 個を作成しました。 これを Pod3 個にスケーリングします。

現在の Pod を確認するには、kubectl get コマンドを実行します。

```
# kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
web-deployment-4255986759-gnvdh     1/1       Running   0          58m
web-deployment-4255986759-r2c6c     1/1       Running   0          58m
```

次に、kubectl scale コマンドを使って、デプロイメントの web-deployment に含まれる Pod の数を手動で 3 個に変更します。

```
# kubectl scale --replicas=3 deployment/web-deployment
deployment "web-deployment" scaled
```

kubectl get pods を実行して、Kubernetes が Pod を作成していることを確認します。スケーリング実行直後は、次のとおり、追加した Pod の STATUS 欄が「ContainerCreating」となります。

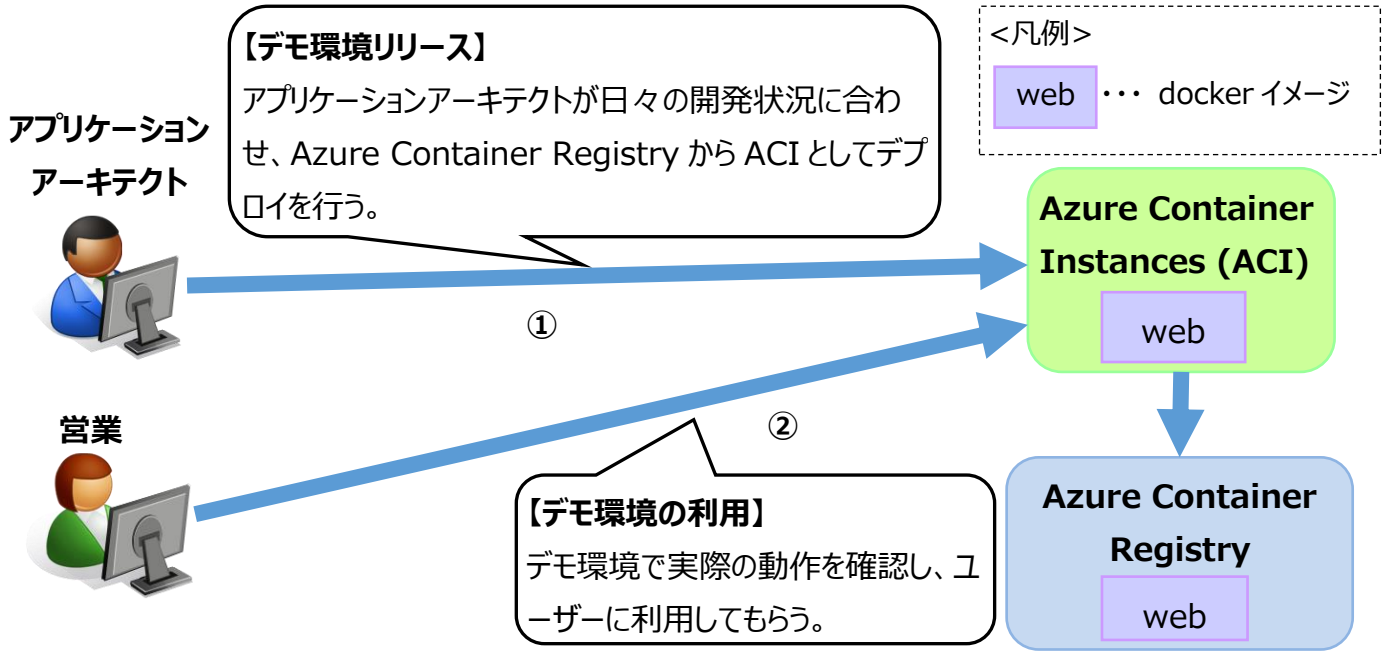
```
# kubectl get pods
NAME                                READY     STATUS              RESTARTS   AGE
web-deployment-4255986759-7bcsd     0/1       ContainerCreating   0          9s
web-deployment-4255986759-gnvdh     1/1       Running             0          1h
web-deployment-4255986759-r2c6c     1/1       Running             0          1h
```

しばらくすると、次のとおり、追加した Pod の STATUS 欄が「Running」となり、実行するようになります。

```
# kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
web-deployment-4255986759-7bcsd     1/1       Running   0          2m
web-deployment-4255986759-gnvdh     1/1       Running   0          1h
web-deployment-4255986759-r2c6c     1/1       Running   0          1h
```

11 Azure Container Instances (ACI) の活用方法

最後に、 Azure Container Instances (ACI) の活用についてご説明します。
今回は以下のようにデモ環境を構築して営業およびユーザーが利用できるようにするモデルケースを想定し、Azure Container Instances (ACI) を一つだけ作成します。簡単な数コマンドを実行することで、最新版のデモ環境を営業とユーザーに提供できます。



このようなモデルケースで利用する場合、利用用途がデモ環境であるので、単一のコンテナが起動していれば十分です。コンテナのオーケストレーションを前提とした Azure Kubernetes Service(AKS)では設定が煩雑になり、このような用途には不向きです。Azure Container Instances (ACI) であれば、そういった不要な設定なしで、単一のコンテナを簡単にデプロイできます。

11.1 Azure Container Instances (ACI) デプロイのための情報確認

az acr login コマンドで Azure Container Registry にログインします。

引数	意味
name	Azure Container Registry の ID です。

```
# az acr login --name acrAksSelfStudy
```

az acr update コマンドにより Azure Container Registry の管理者権限を有効にします。

引数	意味
name	Azure Container Registry の ID です。
admin-enabled	管理者権限の有効(true)、または無効(false)です。今回は有効にしています。

```
# az acr update -n acrAksSelfStudy --admin-enabled true
```

az acr credential show コマンドにより Azure Container Registry のパスワードを取得します。

引数	意味
name	Azure Container Registry の ID です。
query	取得する値です。この場合パスワードの取得となります。

```
# az acr credential show --name acrAksSelfStudy --query "passwords[0].value"
"ifGXBsgf9Sy999TdwLB=Mxkk2HTwG2ka"
```

上記の作業は一回限りとなり、日々のデモ環境デプロイ作業においては不要です。

11.2 Azure Container Instances (ACI) デプロイ

az container create コマンドを実行して Azure Container Instances をデプロイします。

引数	意味
resource-group	Azure Container Instances をデプロイする対象となるリソースグループです。
name	デプロイする Azure Container Instances の ID です。
image	デプロイする Docker イメージです。今回は<9 再リリース>で push したバージョン 2.0 のイメージからデプロイします。
cpu	割り当てる CPU のコア数です。今回は 1 つでデプロイしています。
memory	割り当てる GB 単位のメモリです。今回は 1GB でデプロイしています。
registry-username	Docker イメージを参照する Azure Container Registry の ID です。
registry-password	Docker イメージを参照する Azure Container Registry のパスワードです。
dns-name-label	DNS 名ラベルです。ブラウザで確認するためのグローバルな FQDN を作成するために指定します。
ports	開放するポート番号です。今回は Web アプリケーションのため 80 番をオープンしています。

コマンド実行後に DNS 名ラベルに関するエラーメッセージが表示された場合は、成功するまで変更して再実行してください。

```
# az container create --resource-group rgAksSelfStudy --name acitest-demo --image
acraksselfstudy.azurecr.io/web:2.0 --cpu
1 --memory 1 --registry-username acraksselfstudy --registry-password
ifGXBsgf9Sy999TdwLB=Mxkk2HTwG2ka --dns-name-label acitestdemo --ports 80
```

デプロイの状態を表示するには、 az container show コマンドを使用します。

引数	意味
resource-group	Azure Container Instances がデプロイされているリソースグループです。
name	Azure Container Instances の ID です。
query	取得する値です。この場合デプロイの状態となります。

状態が Pending から Running に変わるまで、コマンドを繰り返しましょう。このコマンドには 1 分もかかりません。

Running になったら、次の手順に進みます。

```
# az container show --resource-group rgAksSelfStudy  --name acitest-demo --query
instanceView.state
“Pending”
```

デプロイに成功すると、az container show コマンドで完全修飾ドメイン名（FQDN）を取得できます。

引数	意味
resource-group	Azure Container Instances がデプロイされているリソースグループです。
name	Azure Container Instances の ID です。
query	取得する値です。この場合 FQDN となります。

```
# az container show --resource-group rgAksSelfStudy --name acitest-demo --query ipAddress.fqdn
“acitestdemo.eastus.azurecontainer.io”
```

ブラウザから acitestdemo.eastus.azurecontainer.io/test.php にアクセスし、表示を確認します。
次の画面が表示されれば、デプロイが正常に実行されているので、リリースは完了です。

v2.0
ID:0001
Name:ichiro

az container logs コマンドでログ出力を表示できます。

引数	意味
resource-group	Azure Container Instances がデプロイされているリソースグループです。
name	Azure Container Instances の ID です。

```
# az container logs --resource-group rgAksSelfStudy --name acitest-demo
```

az container logs コマンドでコンテナを破棄できます。再デプロイする場合は破棄する必要があります。

引数	意味
resource-group	Azure Container Instances がデプロイされているリソースグループです。
name	Azure Container Instances の ID です。

```
# az container delete --resource-group rgAksSelfStudy --name acitest-demo
Are you sure you want to perform this operation? (y/n): y
(省略)
```