記事
Toshihiko Minamoto · 2022年4月14日 12m read

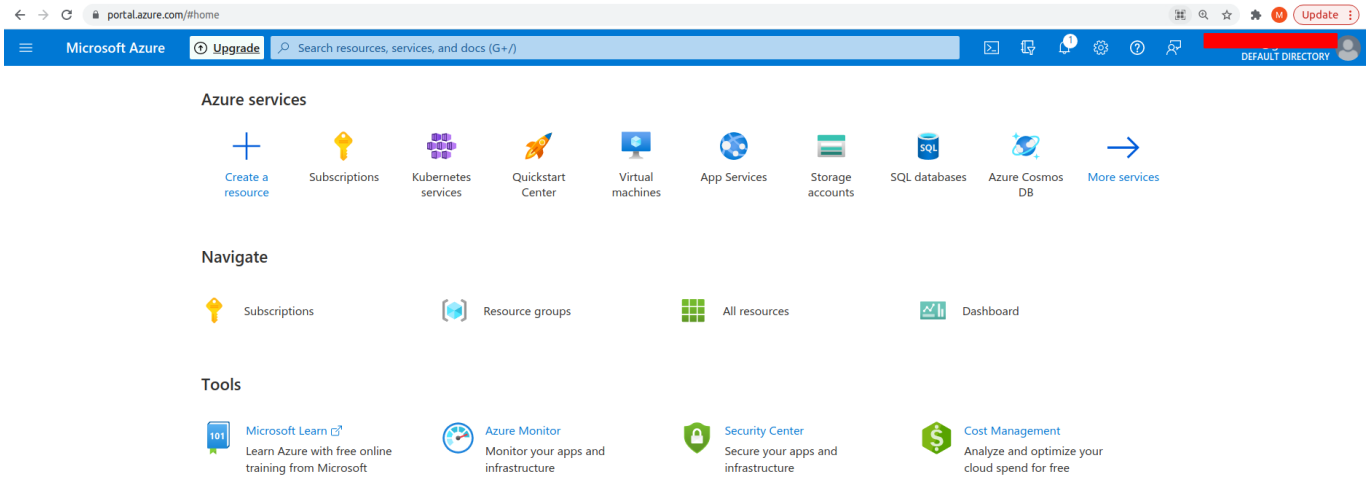
# CircleCI を使用して IRIS アプリケーションを Azure にデプロイする

IRIS ベースのアプリケーションを GCP Kubernetes で実行する方法については、すでに「InterSystems IRIS ソリューションを CircleCI を使用して GCP Kubernetes Cluster GKE へデプロイする」で検討しました。
また、IRIS ベースのアプリケーションを AWS Kubernetes で実行する方法については、「Amazon EKS を使用したシンプルな IRIS ベースの Web アプリケーションのデプロイ」で確認しました。
そこで今回は、アプリケーションを Azure Kubernetes Service
（AKS）にデプロイする方法を説明することにします。

Azure

この記事では、Azure の無料サブスクリプションを使用します。 価格の詳細については、Azure の価格表ページをご覧ください。

登録が完了すると、Microsoft Azure ポータルが表示されます。




便利なポータルではありますが、この記事では使用しません。 代わりに、Azure コマンドラインインターフェースをインストールしましょう。 執筆時点での最新バージョンは 2.30.0 です。

```
$ az version
{
  "azure-cli": "2.30.0",
  "azure-cli-core": "2.30.0",
  "azure-cli-telemetry": "1.0.6",
  "extensions": {}
}
```

それでは、Azure にログインしましょう。

```
$ az login
```

## CircleCI パイプライン

強力な CircleCI の CI/CD を使用して、AKS のセットアップと IRIS
アプリケーションのインストールを行います。 つまり、GitHub ベースのプロジェクトを使って、コードとしてのインフラストラクチャと共にいくつかのパイプラインファイルを追加し、GitHub
に変更をプッシュし直して、その結果を使いやすい CircleCI UI で確認します。

GitHub アカウントを使えば、簡単に CircleCI との統合を作成できます。 詳細については、「Seamless integration with GitHub」の記事をご覧ください。

では、「Amazon EKS を使用したシンプルな IRIS ベースの Web アプリケーションのデプロイ」で使用したプロジェクトの更新バージョンを使用しましょう。 つまり、secured-rest-api です。
それを開き、Use this Template をクリックして、新しいリポジトリ内にバージョンを作成します。
この記事では、そこに含まれるコードサンプルを参照します。

ローカルにリポジトリを Clone し、以下に示すファイルを含む .circleci/ ディレクトリを作成します。

```
$ tree .circleci/
.circleci/
??? config.yml
??? continue.yml
```

ダイナミックコンフィグとパスのフィルタリング
を使用して、変更のあるファイルに応じて全体または一部を実行するようにパイプラインを設定します。
この例では、Terraform コードに変更がある場合にのみ Terraform ジョブを実行します。
最初の config.yml ファイルは単純です。 2 つ目の .circleci/continue.yml を呼び出し、Terraform
コードが最新である場合に特定のブール値パラメーターを渡します。

```
$ cat config.yml
version: 2.1
# Enable CircleCI's dynamic configuration feature
setup: true
# Enable path-based pipeline
orbs:
  path-filtering: circleci/path-filtering@0.1.0
workflows:
  Generate dynamic configuration:
    jobs:
      - path-filtering/filter:
          name: Check updated files
          config-path: .circleci/continue.yml
          base-revision: master
          mapping: |
            terraform/.* terraform-job true
```

2 つ目の continue.yml ファイルを説明する前に、この secured-rest-app プロジェクトを CircleCI
に追加して、.circleci/config.yml の変更を GitHub にプッシュしましょう。

```
$ git add .circleci/config.yml
$ git commit -m "Add circleci config.yml"
$ git push
```

そして、CircleCI Projects ページを開いて、プロジェクトを選択し、Set Up Project をクリックします。

提示される推奨事項に従って、Setup Workflow を有効にします（詳細は、「CircleCI のダイナミックコンフィグの使用を開始する」をご覧ください。

これで、2 つ目の continue.yml ファイルに進む準備が整いました。
このファイルの構造は次のようになっています。

Version: CircleCI パイプラインのバージョンです。
Parameters: Terraform が実行中であるかどうかを決定する変数です。
Orbs: 他の人が作成した再利用可能な構成の一部です。
Executors: 一部のジョブに使用する Asure コマンドラインを含む docker イメージです。
Jobs: 実際のデプロイステップです。
Workflows: Terraform の有無に関係なくパイプラインを実行するためのロジックです。
Jobs セクションには、次のジョブが含まれます。

Build and push Docker image to ACR: このジョブは、az コマンドラインツールがインストールされた docker イメージ内で実行します。 Azure にログインし、イメージをビルドして Azure Container Registry（ACR）にプッシュします。

Terraform: このジョブは、Terraform orb を使用してインフラストラクチャを作成します。 詳細は、以下の Terraform に関するセクションをご覧ください。

Setup packages: このジョブは、IRIS アプリケーションといくつかのサービスアプリケーションをインストールします。
詳細は、以下の「パッケージのセットアップ」セクションをご覧ください。

## Terraform

インフラストラクチャの作成には、infrastructure as code アプローチを使用して、Terraform の力を利用します。
Terraform は Azure プラグインを使って AKS と対話します。
ラッパーの役割を果たし、リソース作成を単純化する AKS Terraform モジュール を使用すると便利です。

Terraform を使用して AKS リソースを使用する例は、「Creating a Kubernetes Cluster with AKS and Terraform」にあります。 ここでは、Terraform がデモと単純化の目的でですべてのリソースを管理するように、Owner ロールを割り当てます。
アプリケーションとしての Terraform は Service Principal を使用して Azure に接続します。 厳密には、「Create an Azure service principal with the Azure CLI」に説明されているとおりに、Owner ロールを Service Principal に割り当てます。

ローカルマシンでコマンドをいくつか実行してみましょう。 Azure サブスクリプション ID を環境変数に保存します。

```
$ export AZ_SUBSCRIPTION_ID=$(az account show --query id --output tsv)
$ az ad sp create-for-
rbac -n "Terraform" --role="Owner" --scopes="/subscriptions/${AZ_SUBSCRIPTION_ID}"
…
{
  "appId": "&lt;appId>",
  "displayName": "&lt;displayName>",
  "name": "&lt;name>",
  "password": "&lt;password>",
  "tenant": "&lt;tenant>"
}
```

後で、Service Principals をリスト表示して Terraform
という表示名を探すと、appId と tenantId を見つけ出すことができます。

```
$ az ad sp list --display-
name "Terraform" | jq '.[] | "AppId: \(.appId), TenantId: \(.appOwnerTenantId)"'
```

ただし、この方法ではパスワードが表示されません。
パスワードを忘れた場合には、資格情報をリセットするしかありません。

パイプラインでは、AKS の作成には、一般に公開されている Azure Terraform モジュールと Terraform
バージョン 1.0.11 を使用します。

取得した、Terraform が Azure への接続に使用する資格情報を使用して、CircleCI project 設定
に環境変数を設定します。 また、DOMAINNAME 環境変数も設定します。 このチュートリアルは demo-iris.myardyas.club ドメイン名を使用していますが、実際にはユーザーが登録したドメイン名を使用します。
パイプラインでこの変数を使用して、IRIS アプリケーションへの外部アクセスを有効にします。 CircleCI 変数と az create-for-rbac コマンドのマッピングは次のとおりです。

```
ARM_CLIENT_ID: appId
ARM_CLIENT_SECRET: password
ARM_TENANT_ID: tenant
ARM_SUBSCRIPTION_ID: Value of environment variable AZ_SUBSCRIPTION_ID
DOMAIN_NAME: your domain name
```

Terraform Remote の状態を有効にするには、Azure Storage の Terraform の状態を使用します。
これを実現するために、ローカルマシンで次のコマンドを実行してみましょう。

```
$ export RESOURCE_GROUP_NAME=tfstate
$ export STORAGE_ACCOUNT_NAME=tfstate14112021 # Must be between 3 and 24 characters i
n length and use numbers and lower-case letters only
$ export CONTAINER_NAME=tfstate

# Create resource group
$ az group create --name ${RESOURCE_GROUP_NAME} --location eastus

# Create storage account
$ az storage account create --resource-group ${RESOURCE_GROUP_NAME} --name ${STORAGE_
ACCOUNT_NAME} --sku Standard_LRS --encryption-services blob
```

```
# Enable versioning. Read more at https://docs.microsoft.com/ja-
jp/azure/storage/blobs/versioning-overview
$ az storage account blob-service-properties update --account-
name ${STORAGE_ACCOUNT_NAME} --enable-versioning true

# Create blob container
$ az storage container create --name ${CONTAINER_NAME} --account-
name ${STORAGE_ACCOUNT_NAME}
```

Terraform ディレクトリに配置した Terraform コードです。 これは、次の 3 つのファイルに分割されています。

provider.tf: Azure プラグインのバージョンと、Terraform
状態を保存するリモートストレージへのパスを設定します。
variables.tf: Terraform モジュールの入力データです。
main.tf: 実際のリソースの作成です。

Azure リソースグループ、パブリック IP、Azure コンテナレジストリなどを作成します。 ネットワーキングと Azure Kubernetes サービスについては、一般に公開されている Terraform モジュールを活用します。

# パッケージのセットアップ

新たに作成された AKS クラスタにインストールするものは、helm ディレクトリにあります。
説明的な Helmfile
アプローチを使用することで、アプリケーション
とその設定を helmfile.yaml ファイルに定義することができます。

セットアップは、単一の helmfile sync コマンドで実行します。 このコマンドによって、IRIS アプリケーションと 2 つの追加アプリケーション、cert-manager、および ingress-nginx
がインストールされ、外部からアプリケーションを呼び出せるようになります。 詳細については、GitHub の「releases」セクションをご覧ください。

IRIS アプリケーションは、「CircleCI ビルドで GKE の作成を自動化する」の説明と同じ Helm
チャートを使用してインストールします。 単純化するために、deployment を使用します。
つまり、ポッドの再起動時に、データは保持されません。
永続させる場合は、Statefulset またはより優れた Kubernetes IRIS Operator
（IKO）を使用することをお勧めします。 IKO
デプロイメントの例は、iris-k8s-monitoring リポジトリにあります。

パイプラインの実行

.circleci/、terraform/、および helm/ ディレクトリを追加したら、それらを GitHub にプッシュします。

```
$ git add .
$ git commit -m "Setup everything"
$ git push
```

すべて問題がなければ、以下のような CircleCI UI 画面が表示されます。

## ドメインレジストラでの A レコードの設定

後もう一つ残っているのは、Terraform が Azure で作成したパブリック IP とDomain Registrar 近ソースのドメイン名の A レコードでバインディングを作成することです。

クラスタに接続しましょう。

```
$ az aks get-credentials --resource-group demo --name demo
```

ingress-nginx で公開されるパブリック IP アドレスを定義します。

```
$ kubectl -n ingress-nginx get service ingress-nginx-
controller -ojsonpath='{.spec.loadBalancerIP}'
x.x.x.x
```

次のようにして、この IP をドメインレジストラ（GoDaddy、Route53、GoogleDomains など）に設定します。

```
YOUR_DOMAIN_NAME = x.x.x.x
```

ここで、DNS の変更が世界中に伝搬されるまでしばらく待ってから、結果を確認します。

```
$ dig +short YOUR_DOMAIN_NAME
```

応答は x.x.x.x となります。

## テスト

ドメイン名を demo-iris.myardyas.club と仮定して、手動テストを実施します。 Let's Encrypt のステージング用証明書を使用しているため、ここでは、証明書の確認は省略しましょう。
本番環境では、lets-encrypt-production （こちら） に置き換える必要があります。
また、メールアドレス（こちら）を example@gmail.com ではないものに設定することもお勧めします。

```
$ curl -sku Bill:ChangeMe https://demo-iris.myardyas.club/crudall/_spec | jq .
…
```

人（ユーザー）を作成する:

```
$ curl -ku John:ChangeMe -XPOST -H "Content-Type: application/json"  https://demo-
iris.myardyas.club/crud/persons/ -d '{"Name":"John Doe"}'
```

人が作成されたかどうかを確認する:

```
$ curl -sku Bill:ChangeMe https://demo-iris.myardyas.club/crud/persons/all | jq .
[
  {
    "Name": "John Doe"
```

```
  }
...
```

まとめ

これで以上です！ Terraform と CircleCI ワークフローを使用して、Azure クラウドに Kubernetes クラスタを作成する方法を確認しました。 IRIS インストールでは、最も簡単な Helm チャートを使用しました。本番環境では、このチャートを拡張し、少なくともデプロイを Statefulset に置き換えるか、IKO を使用することをお勧めします。

作成したリソースが不要になったら、忘れずに削除しましょう。 Azure には、無料利用枠が用意されており、AKS は無料ですが、AKS クラスタの実行用に設計されたリソースは有料です。

#Azure #DevOps #Kubernetes #InterSystems IRIS

---

**ソースURL:**
https://jp.community.intersystems.com/post/circleci-%E3%82%92%E4%BD%BF%E7%94%A8%E3%81%97%E3%81%A6-iris-%E3%82%A2%E3%83%97%E3%83%AA%E3%82%B1%E3%83%BC%E3%82%B7%E3%83%A7%E3%83%B3%E3%82%92-azure-%E3%81%AB%E3%83%87%E3%83%97%E3%83%AD%E3%82%A4%E3%81%99%E3%82%8B