Article
Mihoko Iijima · Jul 6, 2020 14m read

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

前回はシンプルなIRISアプリケーション をGoogleクラウドにデプロイしました。 今回は、同じプロジェクトをAmazon Web Services（アマゾンウェブサービス） のElastic Kubernetes Service（EKS）を使って、デプロイします。

IRISプロジェクトをあなた自身のプライベート・リポジトリにすでにFORKしていると想定します。この記事では<username>/my-objectscript-rest-docker-templateという名前にしています。 <rootrepodir>は、そのルートディレクトリです。

開始する前に、 AWS コマンドライン インターフェースと、Kubernetesクラスタ作成用のシンプルなCLIユーティリティeksctlをインストールします。AWSの場合 aws2 の使用を試すことができますが、ここ で説明するようにkube設定ファイルでaws2の使用法を設定する必要があります 。

AWS EKS

一般的なAWSリソースと同様に、EKSは無料ではありません 。
ただし、無料利用枠のアカウントを作成して、AWSの機能を試すことができます。
ただし、試してみたい機能のすべてが無料枠に含まれているわけではないことに注意してください。
ですから、現在の予算を管理し、金銭的な問題を理解するには、 これと これを読んでください。

ここで は既にAWSアカウントとrootアクセス権があり、このrootアクセス権を使用せず、管理者権限のあるユーザーが作成されていると想定します。
このユーザーのアクセスキーと秘密キーを [dev] プロファイル（またはあなたがつけたプロファイル名）の下のAWS認証情報ファイルに配置する必要があります。

```
$ cat /.aws/credentials
[dev]
awsaccesskeyid = ABCDEFGHIJKLMNOPQRST
awssecretaccesskey = 1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ1234
```

今回は、 AWS「eu-west-1」リージョンにリソースを作成しますが、あなたが今いる場所に最も近いリージョンを選択し、以下に記載されている「eu-west-1」のすべてを選択したリージョンで置き換えてください。

ちなみに、必要なすべてのファイル（.circleci 、eks/、k8s/）も、ここ に保存されており 、簡単にコピーと貼り付けができます。 必要なすべてのEKSリソースは最初から作成されます 。 Amazon EKS Workshop は、良いリソースだと思います。

次に、AWSへのアクセスを確認します（ここではダミーのアカウントを使用しています）。

```
$ export AWSPROFILE=dev
```

```
$ aws sts get-caller-identity
{
  "Account": "012345678910",
  "UserId": " ABCDEFGHIJKLMNOPQRSTU",
  "Arn": "arn:aws:iam::012345678910:user/FirstName.LastName"
}

$ eksctl version
[   ] version.Info{BuiltAt:"", GitCommit:"", GitTag:"0.10.2"}
```

すべてのデフォルト設定が適切であるという事実に従い、「 eksctl create cluster <clustername> --region eu-west-1 」を実行することもできますし、設定ファイルを作成して独自の設定を管理することもできます。

後者は、そのようなファイルをバージョン管理システム（VCS）に保存できるため、よりよい方法です。設定の例はここにあります。 さまざまな設定に関するここの記述を読んだら、独自の設定を作成してみましょう。

```
 mkdir <rootrepodir>/eks; cd <rootrepodir>/eks
$ cat cluster.yaml

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: dev-cluster
  region: eu-west-1
  version: '1.14'

vpc:
  cidr: 10.42.0.0/16
  nat:
  gateway: Single
  clusterEndpoints:
  publicAccess: true
  privateAccess: false

nodeGroups:
  - name: ng-1
  amiFamily: AmazonLinux2
  ami: ami-059c6874350e63ca9 # AMI is specific for a region
  instanceType: t2.medium
  desiredCapacity: 1
  minSize: 1
  maxSize: 1

  # Worker nodes won't have an access FROM the Internet
  # But have an access TO the Internet through NAT-gateway
  privateNetworking: true

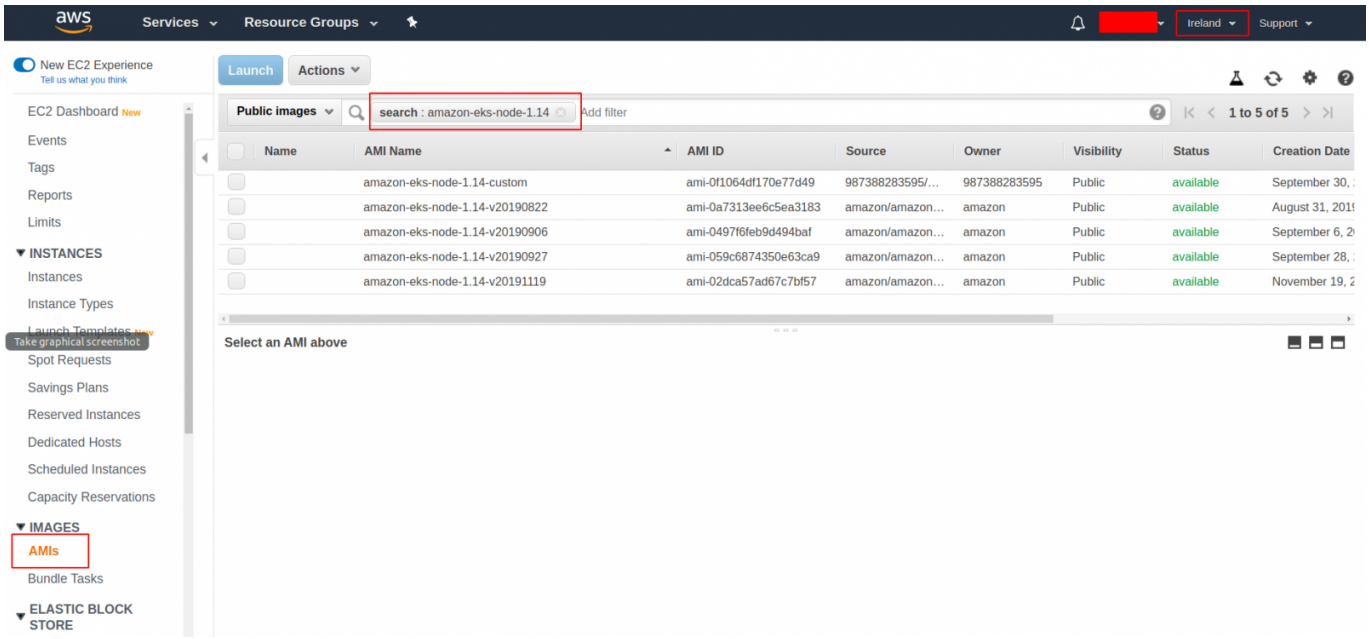
  # We don't need to SSH to nodes for demo
  ssh:
  allow: false

  # Labels are Kubernetes labels, shown when 'kubectl get nodes --show-labels'
  labels:
  role: eks-demo
  # Tags are AWS tags, shown in 'Tags' tab on AWS console'
  tags:
```

```
  role: eks-demo

# CloudWatch logging is disabled by default to save money
# Mentioned here just to show a way to manage it
#cloudWatch:
# clusterLogging:
# enableTypes: []
```

「nodeGroups.desiredCapacity = 1」は本番環境では意味がありませんが、デモでは問題ありません。
また、AMIイメージはリージョン間で異なる可能性があることに注意してください。 「amazon-eks-node-1.14」を探し、最新の1つを選択します。



次に、クラスター（コントロールプレーンとワーカーノード）を作成します。
$ eksctl create cluster -f cluster.yaml

ちなみに、クラスターが不要になった場合は、以下を使用してクラスターを削除できます。
$ eksctl delete cluster --name dev-cluster --region eu-west-1 --wait

クラスターの作成には約15分かかります。 この間、eksctlの出力を確認できます。

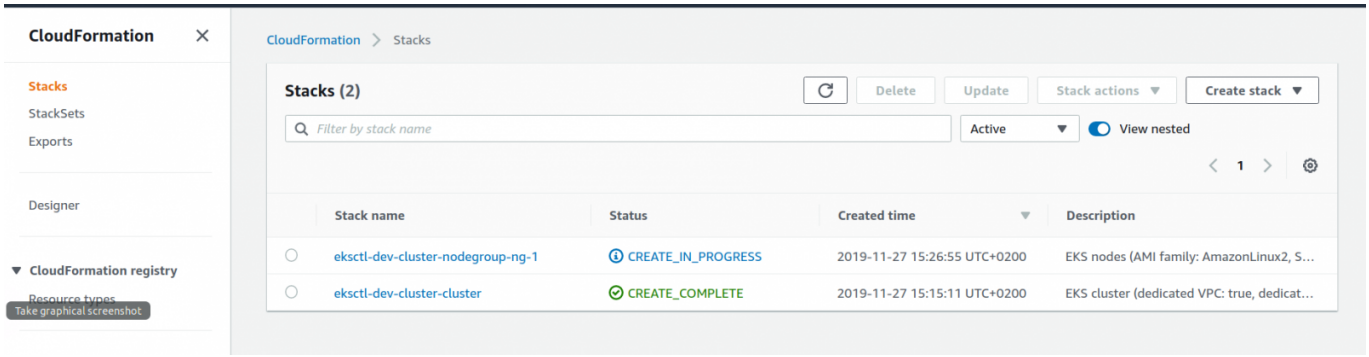
```
[redacted]eks$ eksctl create cluster -f cluster.yaml
[ℹ]  eksctl version 0.10.2
[ℹ]  using region eu-west-1
[ℹ]  setting availability zones to [eu-west-1c eu-west-1b eu-west-1a]
[ℹ]  subnets for eu-west-1c - public:10.42.0.0/19 private:10.42.96.0/19
[ℹ]  subnets for eu-west-1b - public:10.42.32.0/19 private:10.42.128.0/19
[ℹ]  subnets for eu-west-1a - public:10.42.64.0/19 private:10.42.160.0/19
[ℹ]  nodegroup "ng-1" will use "ami-059c6874350e63ca9" [AmazonLinux2/1.14]
[ℹ]  using Kubernetes version 1.14
[ℹ]  creating EKS cluster "dev-cluster" in "eu-west-1" region
[ℹ]  1 nodegroup (ng-1) was included (based on the include/exclude rules)
[ℹ]  will create a CloudFormation stack for cluster itself and 1 nodegroup stack(s)
[ℹ]  will create a CloudFormation stack for cluster itself and 0 managed nodegroup stack(s)
[ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=eu-west-1 --cluster=dev-cluster'
[ℹ]  CloudWatch logging will not be enabled for cluster "dev-cluster" in "eu-west-1"
[ℹ]  you can enable it with 'eksctl utils update-cluster-logging --region=eu-west-1 --cluster=dev-cluster'
[ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "dev-cluster" in "eu-west-1"
[ℹ]  2 sequential tasks: { create cluster control plane "dev-cluster", create nodegroup "ng-1" }
[ℹ]  building cluster stack "eksctl-dev-cluster-cluster"
[ℹ]  deploying stack "eksctl-dev-cluster-cluster"
[ℹ]  building nodegroup stack "eksctl-dev-cluster-nodegroup-ng-1"
[ Take graphical screenshot eksctl-dev-cluster-nodegroup-ng-1"
[✔]  all EKS cluster resources for "dev-cluster" have been created
[✔]  saved kubeconfig as "/home/no-name-pc/.kube/config"
[ℹ]  adding identity "arn:aws:iam::[redacted]:role/eksctl-dev-cluster-nodegroup-ng-1-NodeInstanceRole-FNE1UXTB4TW4" to auth ConfigMap
[ℹ]  nodegroup "ng-1" has 0 node(s)
[ℹ]  waiting for at least 1 node(s) to become ready in "ng-1"
[ℹ]  nodegroup "ng-1" has 1 node(s)
[ℹ]  node "ip-10-42-140-98.eu-west-1.compute.internal" is ready
[✖]  unable to use kubectl with the EKS cluster (check 'kubectl version'): usage: aws [options] <command> <subcommand> [<subcommand> ...] [parameters]
To see help text, you can run:

  aws help
  aws <command> help
  aws <command> <subcommand> help
aws: error: argument operation: Invalid choice, valid choices are:

create-cluster                          | delete-cluster
describe-cluster                        | describe-update
list-clusters                           | list-updates
update-cluster-version                  | update-kubeconfig
wait                                    | help
Unable to connect to the server: getting credentials: exec: exit status 2

[ℹ]  cluster should be functional despite missing (or misconfigured) client binaries
[✔]  EKS cluster "dev-cluster" in "eu-west-1" region is ready
[redacted]eks$
```

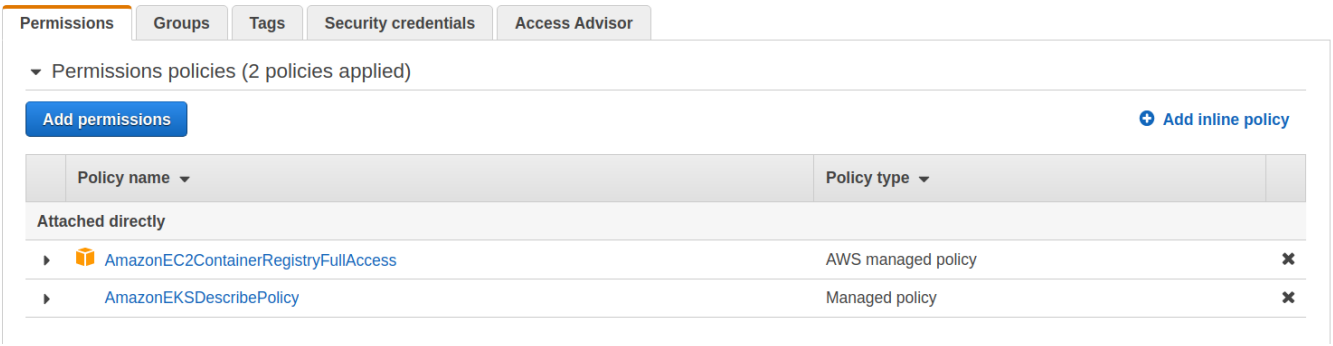
CloudFormationコンソールを参照すると、2つのスタックがあります。それぞれにドリルダウンして、[リソース]タブを参照すると、何が作成されるかを正確に確認でき、[イベント]
タブで、リソース作成の現在の状態を確認できます。



クラスターは正常に作成されましたが、eksctl の出力で「EKSクラスターでkubectlを使用できません」というメッセージがあり、接続に問題があったことがわかります。
aws-iam-authenticator（IAM）をインストールしてkubeコンテキストを作成し、これを解決しましょう。
$ which aws-iam-authenticator
/usr/local/bin/aws-iam-authenticator
$ aws eks update-kubeconfig --name dev-cluster --region eu-west-1

$ kubectl get nodes
NAME STATUS ROLES AGE VERSION
ip-10-42-140-98.eu-west-1.compute.internal Ready <none> 1m v1.14.7-eks-1861c5

これで動作するはずですが、管理者権限を持つユーザーでクラスターを作成しました。 CircleCIからの通常のデプロイ処理では、プログラムによるアクセスのみで、次のポリシーが付与されている特別なAWSユーザー（この例ではCircleCIと名付けられたユーザー）を作成する と良いでしょう。



最初のポリシーはAWSに組み込まれているため、それを選択するだけで済みます。
2つ目は自分で作成する必要があります。 作成プロセスの説明はここにあります。
ポリシー「 AmazonEKSDescribePolicy 」は次のようになります。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "eks:DescribeCluster",
            "eks:ListClusters"
         ],
         "Resource": "*"
      }
   ]
}
```

ユーザーの作成後、ユーザーのアクセスキーと秘密のアクセスキーを保存します。これらのキーはすぐに必要になります。

また、この記事
で説明されているように、Kubernetesクラスター内でこのユーザー権限を付与したいと考えています。
つまり、EKSクラスターを作成した後は、IAMユーザー、すなわち作成者のみがそれにアクセスできます。
CircleCIユーザーを追加するには、クラスターのAWS認証設定（configmap aws-auth、
'data'セクション）のデフォルトの空の「mapUsers」セクションをkubectl edit
を使って（ ‘ 01234567890 ’ の代わりに自分のアカウントIDを使います）次の行に置き換える必要があります。

```
$ kubectl -n kube-system edit configmap aws-auth
...
data:
...
  mapUsers: |
  - userarn: arn:aws:iam::01234567890:user/CircleCI
  username: circle-ci
  groups:
  - system:masters
```

以前の記事のKubernetesマニフェストを使用します
（「Googleクラウドの前提条件」のセクションを参照）。以前のやり方と違う点は、デプロイのイメージフィー

ルドでプレースホルダを使うということだけです。 これらのマニフェストを<rootrepodir>/k8sディレクトリに保存します。 デプロイファイルの名前がdeployment.tplに変更されたことに注意してください。
$ cat <rootrepodir>/k8s/deployment.tpl
...
spec:
containers:
- image: DOCKERREPONAME/iris-rest:DOCKERIMAGETAG
...

CircleCI

CircleCI側のデプロイ処理は、GKEに使用される処理に似ています。

- リポジトリをPullする
- Dockerイメージをビルドする
- Amazon クラウドで認証する
- イメージをAmazon Elastic Container Registry（ECR）にアップロードする
- このイメージを基にしたAWS EKSでコンテナを実行する

前回と同様に、作成およびテスト済みのCircleCI構成テンプレートorbsを利用します。

- イメージを構築してECRにPushするためのaws-ecr orb
- AWS認証のためのaws-eks orb
- Kubernetesマニフェストのデプロイのためのkubernetes orb

デプロイ構成は次のようになります。
$ cat <rootrepodir>/.circleci/config.yml
version: 2.1
orbs:
  aws-ecr: circleci/aws-ecr@6.5.0
  aws-eks: circleci/aws-eks@0.2.6
  kubernetes: circleci/kubernetes@0.10.1
jobs:
  deploy-application:
  executor: aws-eks/python3
  parameters:
  cluster-name:
  description: |
  Name of the EKS cluster
  type: string
  aws-region:
  description: |
  AWS region
  type: string
  account-url:
  description: |
  Docker AWS ECR repository url
  type: string
  tag:
  description: |
  Docker image tag
  type: string
  steps:
  - checkout
  - run:

```
  name: Replace placeholders with values in deployment template
  command: |
  cat k8s/deployment.tpl |\
  sed "s|DOCKERREPONAME|<< parameters.account-url >>|" |\
  sed "s|DOCKERIMAGETAG|<< parameters.tag >>|" > k8s/deployment.yaml; \
  cat k8s/deployment.yaml
  - aws-eks/update-kubeconfig-with-authenticator:
  cluster-name: << parameters.cluster-name >>
  install-kubectl: true
  aws-region: << parameters.aws-region >>
  - kubernetes/create-or-update-resource:
  action-type: apply
  resource-file-path: "k8s/namespace.yaml"
  show-kubectl-command: true
  - kubernetes/create-or-update-resource:
  action-type: apply
  resource-file-path: "k8s/deployment.yaml"
  show-kubectl-command: true
  get-rollout-status: true
  resource-name: deployment/iris-rest
  namespace: iris
  - kubernetes/create-or-update-resource:
  action-type: apply
  resource-file-path: "k8s/service.yaml"
  show-kubectl-command: true
  namespace: iris
workflows:
  main:
  jobs:
  - aws-ecr/build-and-push-image:
  aws-access-key-id: AWSACCESSKEYID
  aws-secret-access-key: AWSSECRETACCESSKEY
  region: AWSREGION
  account-url: AWSECRACCOUNTURL
  repo: iris-rest
  create-repo: true
  dockerfile: Dockerfile-zpm
  path: .
  tag: ${CIRCLESHA1}
  - deploy-application:
  cluster-name: dev-cluster
  aws-region: eu-west-1
  account-url: ${AWSECRACCOUNTURL}
  tag: ${CIRCLESHA1}
  requires:
  - aws-ecr/build-and-push-image
```

ワークフロー
のセクションにはジョブリストが含まれ、各ジョブはaws-ecr/build-and-push-image
などのorbから呼び出すか、構成で「deploy-application」を使って直接定義できます。

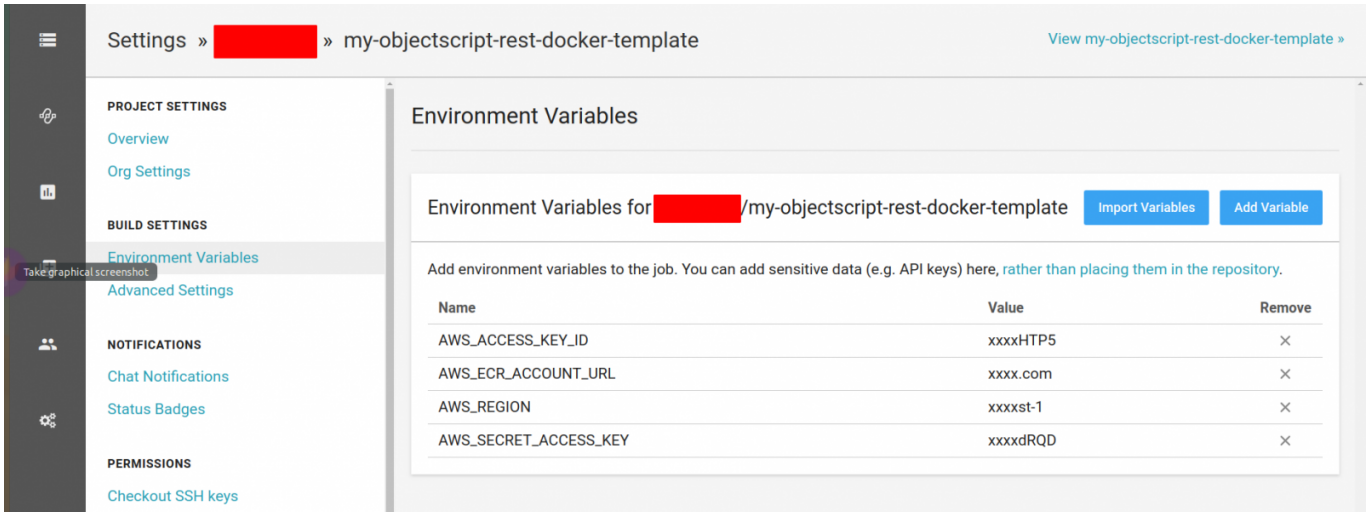
次のコードは、aws-ecr/build-and-push-imageジョブが終了した後で、deploy-applicationジョブが呼び出されることを意味します。

```
requires:
- aws-ecr/build-and-push-image
```

[ジョブ] セクションには、デプロイ・アプリケーションジョブの説明と、次のような定義された手順のリストが含まれています。

- checkoutで、GitリポジトリからPullする
- runで、Docker-imageリポジトリとタグを動的に設定するスクリプトを実行する
- aws-iam-authenticatorを使用する aws-eks /update-kubeconfig-with-authenticatorを使用して Kubernetesへの接続を設定する
- CircleCIから「kubectl apply」を実行する方法として数回使用されるkubernetes/create-or-update-resource

変数を使用しますが、もちろんそれらはCircleCIの「環境変数」タブで定義する必要があります。

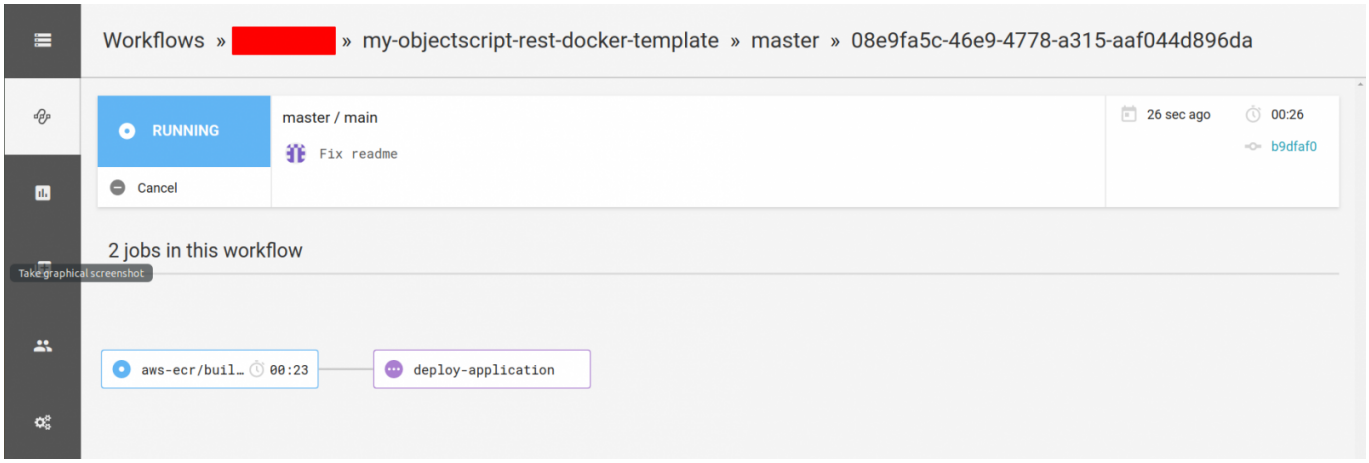


次の表は、使用される変数の意味を示しています。

AWSACCESSKEYID	CircleCI IAMユーザーのアクセスキー
AWSSECRETACCESSKEY	CircleCI IAMユーザーの秘密キー
AWSREGION	eu-west-1、この場合
AWSECRACCOUNTURL	0 12 34567890.dkr.ecr.eu-west-1.amazonaws.comなどのAWS ECR Docker レジストリのURL 「01234567890」がアカウント IDの場合

デプロイ処理をトリガーする方法は次のとおりです。
$ git add .circleci/ eks/ k8s/
$ git commit -m “ AWS EKS deployment ”
$ git push

これにより、このワークフローにおける2つのジョブが表示されます。



どちらのジョブもクリック可能であり、これにより、実行した手順の詳細を確認できます。
デプロイには数分かかります。
完了したら、KubernetesリソースとIRISアプリケーション自体のステータスを確認できます。
$ kubectl -n iris get pods -w # Ctrl+C to stop $ kubectl -n iris get service
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
iris-rest LoadBalancer 172.20.190.211 a3de52988147a11eaaaff02ca6b647c2-663499201.eu-west-1.elb.amazonaws.com 52773:32573/TCP 15s

DNSレコードが反映されるまで数分かかります。 それまでは、curlを実行すると「ホストを解決できませんでした」というエラーが表示されます。
$ curl -XPOST -H "Content-Type: application/json" -u system:SYS a3de52988147a11eaaaff02ca6b647c2-663499201.eu-west-1.elb.amazonaws.com:52773/person/ -d '{"Name":"John Dou"}' $ curl -XGET -u system:SYS a3de52988147a11eaaaff02ca6b647c2-663499201.eu-west-1.elb.amazonaws.com:52773/person/all
[{"Name":"John Dou"},]

まとめ

一見するとAWS
EKSへのデプロイはGKEへのデプロイよりも複雑に見えますが、それほど大きな違いはありません。
組織でAWSを使用している場合は、Kubernetesをスタックに追加する方法を理解されたと思います。

最近、EKS
APIが拡張され、管理グループをサポートできるようになりました
。これにより、コントロールプレーンとデータプレーンを全体としてデプロイでき、これは将来有望と思われます。
さらに、コンテナ用のAWSサーバーレスコンピューティングエンジンであるFargateが利用可能になりました。

最後に、AWS
ECRに関する簡単な注意事項を記します：イメージにライフサイクルポリシー
を設定することを忘れないでください。

InterSystems Open Exchangeで関連アプリケーション をご確認ください。

#AWS #Kubernetes #Cloud #Containerization #Deployment #InterSystems IRIS #InterSystems IRIS for Health #Open Exchange

Source URL:https://community.intersystems.com/node/478961