文章
jieliang liu · 一月 7, 2021 阅读大约需 20 分钟

Open Exchange

使用 GitHub Actions 在 EKS 上部署 InterSystems IRIS 解决方案

假设你想了解 InterSystems 在数据分析方面能提供什么。 你研究了理论，现在想要进行一些实践。
幸运的是，InterSystems 提供了一个项目：Samples BI，其中包含了一些很好的示例。 从 README
文件开始，跳过任何与 Docker 相关的内容，直接进行分步安装。 启动虚拟实例 安装 IRIS，按照说明安装 Samples
BI，然后用漂亮的图表和表格让老板眼前一亮。 到目前为止还不错。

但是不可避免地，你需要进行更改。

事实证明，自己保留虚拟机存在一些缺点，交给云服务商保管是更好的选择。 Amazon
看起来很可靠，你只需创建一个 AWS 帐户（入门免费），了解到使用 root 用户身份执行日常任务是有害的
，然后创建一个常规的具有管理员权限的 IAM 用户。

点击几下鼠标，就可以创建自己的 VPC 网络、子网和虚拟 EC2 实例，还可以添加安全组来为自己开放 IRIS
Web端口 (52773) 和 ssh 端口 (22)。 重复 IRIS 和 Samples BI 的安装。 这次使用 Bash
脚本，如果你喜欢，也可以使用 Python。 再一次让老板刮目相看。

但是无处不在的 DevOps 运动让你开始了解基础架构即代码，并且你想要实现它。 你选择了
Terraform，因为它是众所周知的，而且它的方法非常通用，只需微小调整即可适合各种云提供商。 使用 HCL 语言
描述基础架构，并将 IRIS 和 Samples BI 的安装步骤转换到 Ansible。 然后再创建一个 IAM 用户使 Terraform
正常工作。 全部运行一遍。 获得工作奖励。

渐渐地你会得出结论，在我们这个微服务时代，不使用 Docker 就太可惜了，尤其是 InterSystems
还会告诉你怎么做。 返回到 Samples BI 安装指南并阅读关于 Docker 的几行内容，似乎并不复杂：

```
  $ docker pull intersystemsdc/iris-community:2019.4.0.383.0-zpm
$ docker run --name irisce -d --publish 52773:52773 intersystemsdc/iris-community:2019.4.0.383.0-zpm
$ docker exec -it irisce iris session iris
USER>zpm
zpm: USER>install samples-bi
```

将浏览器定向到 http://localhost:52773/csp/user/DeepSee.UserPortal.Home.zen?$NAMESPACE=USER
后，再次去老板那里，因为做得好而获得一天假期。

然后你开始明白，“ docker run ”只是开始，至少需要使用docker-compose。 没问题：

```
  $ cat docker-compose.yml
version: "3.7"
services:
  irisce:
  containername: irisce
  image: intersystemsdc/iris-community:2019.4.0.383.0-zpm
  ports:
  - 52773:52773
$ docker rm -f irisce # We don ’ t need the previous container
$ docker-compose up -d
```

这样你使用 Ansible 安装了 Docker 和 docker-
compose，然后运行了容器，如果机器上还没有映像，则会下载一个映像。 最后安装了 Samples BI。

你一定喜欢 Docker，因为它是各种内核素材的又酷又简单的接口。 你开始在其他地方使用

Docker，并且经常启动多个容器。 还发现容器必须经常互相通信，这就需要了解如何管理多个容器。

终于，你发现了 Kubernetes。

从 docker-compose 快速切换到 Kubernetes 的一个方法是使用 kompose。 我个人更喜欢简单地从手册中复制 Kubernetes 清单，然后自己编辑，但是 kompose 在完成小任务方面做得很好：

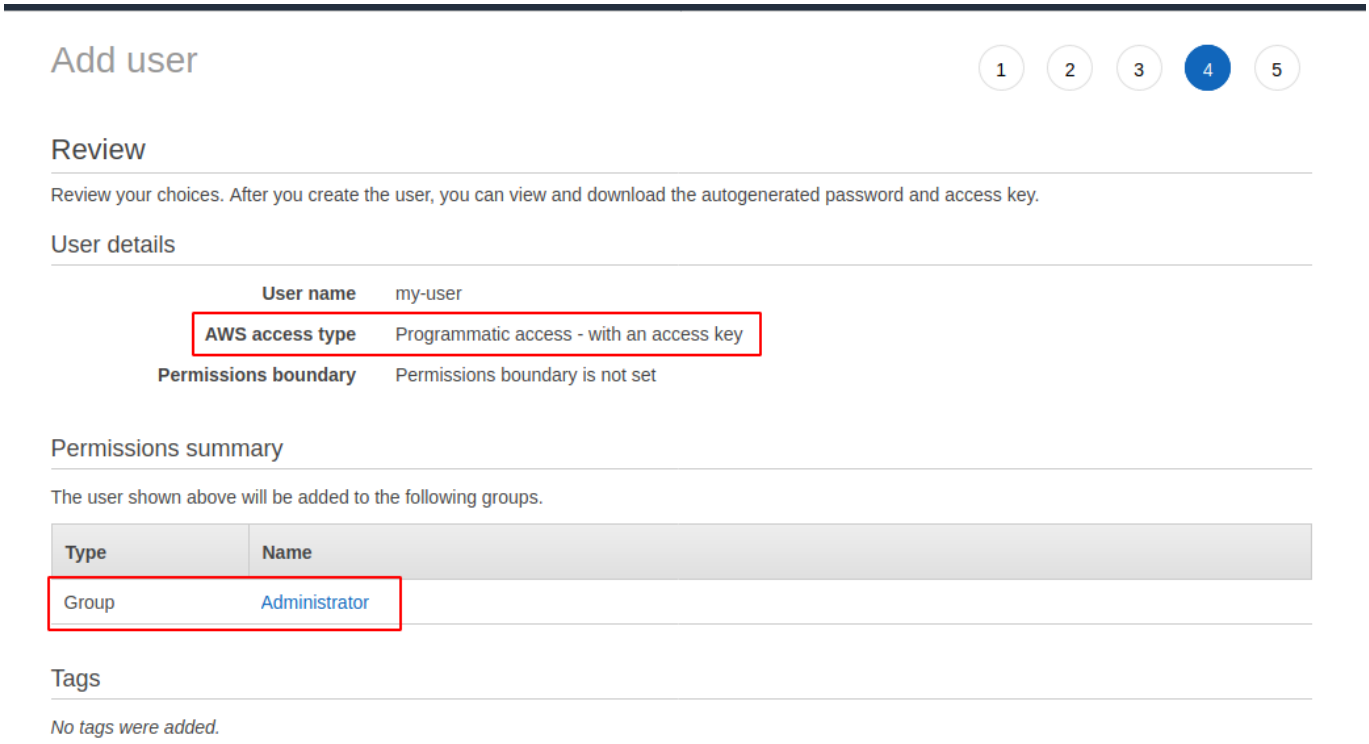
```
 $ kompose convert -f docker-compose.yml
INFO Kubernetes file "irisce-service.yaml" created
INFO Kubernetes file "irisce-deployment.yaml" created
```

现在你有了可以发送到某个 Kubernetes 集群的部署和服务文件。 你发现可以安装 minikube，它允许你运行一个单节点 Kubernetes 集群，这正是你现阶段所需要的。 在摆弄一两天 minikube 沙盒之后，你已经准备好在 AWS 云中的某处使用真实的 Kubernetes 部署。

设置

我们一起来进行吧。 此时，我们做以下几个假设：

首先，我们假设你有一个 AWS 帐户，你知道其 ID，并且未使用 root 凭据。
你创建了一个具有管理员权限且只能以编程方式访问的 IAM 用户（我们称之为“ my-user ”），并存储了其凭据。
你还创建了另一个具有相同权限的 IAM 用户，名为“ terraform ”：



Terraform 将以它的名义进入你的 AWS 帐户，并创建和删除必要资源。 这两个用户的广泛权限将通过演示来说明。
你在本地保存了这两个 IAM 用户的凭据：

```
 $ cat ~/.aws/credentials
[terraform]
aws_access_key_id = ABCDEFGHIJKLMNOPQRST
aws_secret_access_key = ABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890123
[my-user]
aws_access_key_id = TSRQPONMLKJIHGFEDCBA
aws_secret_access_key = TSRQPONMLKJIHGFEDCBA01234567890123
```

注意：不要复制和粘贴上面的凭据。 它们在这里作为示例提供，不再存在。 请编辑 ~/.aws/credentials 文件并引入你自己的记录。

其次，我们将在文中使用虚拟的 AWS 帐户 ID (01234567890) 和 AWS 区域 “ eu-west-1 ”。 可以随意使用其他区域。

第三，我们假设你知道 AWS 不是免费的，你需要为使用的资源付费。

接下来，您已经安装了 AWS CLI 实用程序，以便与 AWS 进行命令行通信。 你可以尝试使用 aws2，但你需要在 kube 配置文件中特别设置 aws2 的用法，如这里所述。

你还安装了 kubectl 实用程序来与 AWS Kubernetes 进行命令行通信。

并且你也针对 docker-compose.yml 安装了 kompose 实用程序，来转换 Kubernetes 清单。

最后，你创建了一个空的 GitHub 仓库，并将其克隆到主机上。 我们将其根目录引用为 。
在此仓库中，我们将创建并填充三个目录：.github/workflows/、k8s/ 和 terraform/。

请注意，所有相关代码都在 github-eks-samples-bi 仓库中复制，以简化拷贝和粘贴。

我们继续。

AWS EKS 预置

我们已经在文章使用 Amazon EKS 部署简单的基于 IRIS 的 Web 应用程序中知道了 EKS。
那时，我们以半自动方式创建了一个集群。 即，我们在一个文件中描述集群，然后从本地机器手动启动 eksctl 实用程序，该实用程序根据我们的描述创建集群。

eksctl 是为创建 EKS
集群而开发的，它非常适合概念验证实现，但对于日常使用来说，最好使用更通用的工具，例如 Terraform。 AWS EKS 简介是一个非常好的资源，其中介绍了创建 EKS 集群所需的 Terraform 配置。
花一两个小时熟悉一下，决不会是浪费时间。

你可以在本地操作 Terraform。 为此，你需要一个二进制文件（在撰写本文时，我们使用最新的 Linux 版本 0.12.20 ），并且 IAM 用户 “ terraform ” 需要有足够的权限才能让 Terraform 进入 AWS。 创建目录 /terraform/ 以存储 Terraform 代码：

```
  $ mkdir /terraform
$ cd /terraform
```

你可以创建一个或多个 *.tf* 文件（它们会在启动时合并）。 只需复制并粘贴 AWS EKS 简介
中的代码示例，然后运行如下命令：

```
  $ export AWS_PROFILE=terraform
$ export AWS_REGION=eu-west-1
$ terraform init
$ terraform plan -out eks.plan
```

你可能会遇到一些错误。 如果遇到的话，可以在调试模式下操作，但记得稍后关闭该模式：

```
  $ export TF_LOG=debug
$ terraform plan -out eks.plan

$ unset TF_LOG
```

这个经验会很有用，你很可能会启动一个 EKS 集群（使用 “ terraform apply ” 进行该操作）。 在 AWS
控制台中查看：

觉得厌烦时就清理掉：

```
  $ terraform destroy
```

然后进入下一阶段，开始使用 Terraform EKS 模块，尤其它也基于同一 EKS 简介。 在 examples/ 目录中，你将看到如何使用它。 你还会在那里找到其他示例。

我们对示例进行了一定的简化。 以下是主文件，其中调用了 VPC 创建和 EKS 创建模块：

```
  $ cat /terraform/main.tf
terraform {
  requiredversion = ">= 0.12.0"
  backend "s3" {
  bucket = "eks-github-actions-terraform"
  key = "terraform-dev.tfstate"
  region = "eu-west-1"
  dynamodbtable = "eks-github-actions-terraform-lock"
  }
}
provider "kubernetes" {
  host = data.awseksclusterendpoint
  clustercacertificate = base64decode(data.awseksclusterclustercertificateauthority.0.data)
  token = data.awseksclusterauthclustertoken
  loadconfigfile = false
  version = "1.10.0"
}

locals {
  vpcname = "dev-vpc"
  vpccidr = "10.42.0.0/16"
  privatesubnets = ["10.42.1.0/24", "10.42.2.0/24"]
  publicsubnets = ["10.42.11.0/24", "10.42.12.0/24"]
  clustername = "dev-cluster"
  clusterversion = "1.14"
  workergroupname = "worker-group-1"
  instancetype = "t2.medium"
  asgdesiredcapacity = 1
}

data "awseksclusterid" {
  name = module.eks.clusterid
}

data "awseksclusterauthid" {
  name = module.eks.clusterid
}

data "awsavailabilityzones" "available" {
}

module "vpc" {
  source = "git::https://github.com/terraform-aws-modules/terraform-aws-vpc?ref=master"

  name = local.vpcname
  cidr = local.vpccidr
  azs = data.awsavailabilityzones.available.names
  privatesubnets = local.privatesubnets
  publicsubnets = local.publicsubnets
  enablenatgateway = true
  singlenatgateway = true
  enablednshostnames = true

  tags = {
  "kubernetes.io/cluster/${local.clustername}" = "shared"
```

```
  }

  publicsubnettags = {
  "kubernetes.io/cluster/${local.clustername}" = "shared"
  "kubernetes.io/role/elb" = "1"
  }

  privatesubnettags = {
  "kubernetes.io/cluster/${local.clustername}" = "shared"
  "kubernetes.io/role/internal-elb" = "1"
  }
}

module "eks" {
  source = "git::https://github.com/terraform-aws-modules/terraform-aws-eks?ref=master"
  clustername = local.clustername
  clusterversion = local.clusterversion
  vpcid = module.vpc.vpcid
  subnets = module.vpc.privatesubnets
  writekubeconfig = false

  workergroups = [
  {
  name = local.workergroupname
  instancetype = local.instancetype
  asgdesiredcapacity = local.asgdesiredcapacity
  }
  ]

  mapaccounts = var.mapaccounts
  maproles = var.maproles
  mapusers = var.mapusers
}
```

我们再仔细看一下 main.tf 中的 *"terraform"* 块：

```
  terraform {
  requiredversion = ">= 0.12.0"
  backend "s3" {
  bucket = "eks-github-actions-terraform"
  key = "terraform-dev.tfstate"
  region = "eu-west-1"
  dynamodbtable = "eks-github-actions-terraform-lock"
  }
}
```

这里需要指出，我们将遵守不低于 Terraform 0.12 的语法（与早期版本相比有了很大变化），同时，Terraform 不应该将其状态存储在本地，而是远程存储在 S3 存储桶中。

不同的人可以从不同的地方更新 terraform 代码确实很方便，这意味着我们需要能够锁定用户的状态，因此我们使用 dynamodb 表添加了一个锁。 有关锁定的更多信息，请参见状态锁定页面。

由于存储桶的名称在整个 AWS 中应该是唯一的，因此你不能再使用名称 " eks-github-actions-terraform " 。
请想一个你自己的名称，并确保它没有被占用（应该收到 *NoSuchBucket* 错误）：

```
  $ aws s3 ls s3://my-bucket
调用 ListObjectsV2 操作时发生错误 (AllAccessDisabled)：对此对象的所有访问均已禁用
$ aws s3 ls s3://my-bucket-with-name-that-impossible-to-remember
调用 ListObjectsV2 操作时发生错误 (NoSuchBucket)：指定的存储桶不存在
```

想好一个名称，创建存储桶（我们这里使用 IAM 用户 “ terraform ”。
它拥有管理员权限，因此可以创建存储桶），并为其启用版本管理（这在配置出错时能让你省心）：

```
 $ aws s3 mb s3://eks-github-actions-terraform --region eu-west-1
makebucket: eks-github-actions-terraform
$ aws s3api put-bucket-versioning --bucket eks-github-actions-terraform --versioning-configuration Status=Enabled
$ aws s3api get-bucket-versioning --bucket eks-github-actions-terraform
{
 "Status": "Enabled"
}
```

对于 DynamoDB，不需要唯一性，但你需要先创建一个表：

```
 $ aws dynamodb create-table \
 --region eu-west-1 \
 --table-name eks-github-actions-terraform-lock \
 --attribute-definitions AttributeName=LockID,AttributeType=S \
 --key-schema AttributeName=LockID,KeyType=HASH \
 --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

注意，如果 Terraform 操作失败，你可能需要从 AWS 控制台手动删除锁。 但这样做时要小心。

对于 main.tf 中的 eks/vpc 模块，引用 GitHub 上提供的模块很简单：

```
 git::https://github.com/terraform-aws-modules/terraform-aws-vpc?ref=master
```

现在看一下另外两个 Terraform 文件（variables.tf 和 outputs.tf）。 第一个文件保存了 Terraform 变量：

```
 $ cat /terraform/variables.tf
variable "region" {
 default = "eu-west-1"
}
variable "mapaccounts" {
 description = "Additional AWS account numbers to add to the aws-auth configmap. See
examples/basic/variables.tf for example format."
 type = list(string)
 default = []
}

variable "maproles" {
 description = "Additional IAM roles to add to the aws-auth configmap."
 type = list(object({
 rolearn = string
 username = string
 groups = list(string)
 }))
 default = []
}

variable "mapusers" {
 description = "Additional IAM users to add to the aws-auth configmap."
 type = list(object({
 userarn = string
 username = string
 groups = list(string)
 }))
 default = [
 {
 userarn = "arn:aws:iam::01234567890:user/my-user"
 username = "my-user"
 groups = ["system:masters"]
 }
 ]
```

```
}
```

这里最重要的部分是将 IAM 用户“ my-user ”添加到 mapusers 变量中，但你应该使用自己的帐户 ID 替换 01234567890。

这有什么用？ 当通过本地 kubectl 客户端与 EKS 通信时，它会向 Kubernetes API 服务器发送请求，每个请求都要经过身份验证和授权过程，这样 Kubernetes 就可以知道谁发送了请求，以及它们可以做什么。 因此 Kubernetes 的 EKS 版本会要求 AWS IAM 帮助进行用户身份验证。 如果发送请求的用户列在 AWS IAM 中（这里我们指向其 ARN），请求将进入授权阶段，该阶段将由 EKS 自己处理，但要依据我们的设置。 这里要指出的是，IAM 用户“ my-user ”非常酷 组“ system: masters ”。

最后，output.tf 文件描述了 Terraform 在完成工作后应该打印的内容：

```
 $ cat /terraform/outputs.tf
output "clusterendpoint" {
 description = "Endpoint for EKS control plane."
 value = module.eks.clusterendpoint
}
output "clustersecuritygroupid" {
 description = "Security group ids attached to the cluster control plane."
 value = module.eks.clustersecuritygroupid
}

output "configmapawsauth" {
 description = "A kubernetes configuration to authenticate to this EKS cluster."
 value = module.eks.configmapawsauth
}
```

Terraform 部分的描述完成。 我们很快就会回来，看看如何启动这些文件。

Kubernetes 清单

到目前为止，我们已经解决了*在哪里*启动应用程序的问题。 现在我们来看看要运行*什么*。

回想一下 /k8s/ 目录中的 docker-compose.yml（我们重命名了服务，添加了几个不久就会被 kompose 用到的标签）：

```
 $ cat /k8s/docker-compose.yml
version: "3.7"
services:
 samples-bi:
 containername: samples-bi
 image: intersystemsdc/iris-community:2019.4.0.383.0-zpm
 ports:
 - 52773:52773
 labels:
 kompose.service.type: loadbalancer
 kompose.image-pull-policy: IfNotPresent
```

运行 kompose，然后添加下面突出显示的内容。 删除注释（使内容更容易理解）：

```
 $ kompose convert -f docker-compose.yml --replicas=1
$ cat /k8s/samples-bi-deployment.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
 labels:
 io.kompose.service: samples-bi
 name: samples-bi
```

```
spec:
  replicas: 1
  strategy:
  type: Recreate
  template:
  metadata:
  labels:
  io.kompose.service: samples-bi
  spec:
  containers:
  - image: intersystemsdc/iris-community:2019.4.0.383.0-zpm
  imagePullPolicy: IfNotPresent
  name: samples-bi
  ports:
  - containerPort: 52773
  resources: {}
  lifecycle:
  postStart:
  exec:
  command:
  - /bin/bash
  - -c
  - |
  echo -e "write \nhalt" > test
  until iris session iris < test; do sleep 1; done
  echo -e "zpm \ninstall samples-bi \nquit \nhalt" > samplesbiinstall
  iris session iris < samplesbiinstall
  rm test samplesbiinstall
  restartPolicy: Always
```

我们使用 Recreate 更新策略，这意味着先删除 pod，然后重新创建。
这对于演示目的是允许的，让我们可以使用更少的资源。
我们还添加了 postStart 挂钩，该挂钩在 pod 启动后立即触发。 我们等待至 IRIS 启动，然后从默认的 zpm-repository 安装 samples-bi 包。
现在我们添加 Kubernetes 服务（同样没有注释）：

```
  $ cat /k8s/samples-bi-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
  io.kompose.service: samples-bi
  name: samples-bi
spec:
  ports:
  - name: "52773"
  port: 52773
  targetPort: 52773
  selector:
  io.kompose.service: samples-bi
  type: LoadBalancer
```

是的，我们将在“ 默认 ”命名空间中部署，该命名空间适合演示。

好了，现在我们知道了运行*位置*和*内容*。 还剩下*方式*需要了解。

GitHub Actions 工作流程

我们不需要每件事都从头开始做，而是创建一个工作流程，类似于使用 GitHub Actions 在 GKE 上部署 InterSystems IRIS 解决方案中所述的工作流程。 这次，我们不必担心构建容器。 GKE 特定的部分已替换为特定于 EKS。

粗体部分与接收提交消息和在条件步骤中使用它有关：

```
  $ cat /.github/workflows/workflow.yaml
name: Provision EKS cluster and deploy Samples BI there
on:
  push:
  branches:
  - master
# Environment variables.
# ${{ secrets }} are taken from GitHub -> Settings -> Secrets
# ${{ github.sha }} is the commit hash
env:
  AWSACCESSKEYID: ${{ secrets.AWSACCESSKEYID }}
  AWSSECRETACCESSKEY: ${{ secrets.AWSSECRETACCESSKEY }}
  AWSREGION: ${{ secrets.AWSREGION }}
  CLUSTERNAME: dev-cluster
  DEPLOYMENTNAME: samples-bi

jobs:
  eks-provisioner:
  # Inspired by:
  ## https://www.terraform.io/docs/github-actions/getting-started.html
  ## https://github.com/hashicorp/terraform-github-actions
  name: Provision EKS cluster
  runs-on: ubuntu-18.04
  steps:
  - name: Checkout
  uses: actions/checkout@v2

  - name: Get commit message
  run: |
  echo ::set-env name=commitmsg::$(git log --format=%B -n 1 ${{ github.event.after }})

  - name: Show commit message
  run: echo $commitmsg

  - name: Terraform init
  uses: hashicorp/terraform-github-actions@master
  with:
  tfactionsversion: 0.12.20
  tfactionssubcommand: 'init'
  tfactionsworkingdir: 'terraform'

  - name: Terraform validate
  uses: hashicorp/terraform-github-actions@master
  with:
  tfactionsversion: 0.12.20
  tfactionssubcommand: 'validate'
  tfactionsworkingdir: 'terraform'

  - name: Terraform plan
   if: "!contains(env.commitmsg, '[destroy eks]')"
  uses: hashicorp/terraform-github-actions@master
  with:
  tfactionsversion: 0.12.20
  tfactionssubcommand: 'plan'
  tfactionsworkingdir: 'terraform'

  - name: Terraform plan for destroy
  if: "contains(env.commitmsg, '[destroy eks]')"
```

```
uses: hashicorp/terraform-github-actions@master
with:
tfactionsversion: 0.12.20
tfactionssubcommand: 'plan'
args: '-destroy -out=./destroy-plan'
tfactionsworkingdir: 'terraform'

- name: Terraform apply
if: "!contains(env.commitmsg, '[destroy eks]')"
uses: hashicorp/terraform-github-actions@master
with:
tfactionsversion: 0.12.20
tfactionssubcommand: 'apply'
tfactionsworkingdir: 'terraform'

- name: Terraform apply for destroy
 if: "contains(env.commitmsg, '[destroy eks]')"
uses: hashicorp/terraform-github-actions@master
with:
tfactionsversion: 0.12.20
tfactionssubcommand: 'apply'
args: './destroy-plan'
tfactionsworkingdir: 'terraform'

kubernetes-deploy:
name: Deploy Kubernetes manifests to EKS
needs:
- eks-provisioner
runs-on: ubuntu-18.04
steps:
- name: Checkout
uses: actions/checkout@v2

- name: Get commit message
run: |
echo ::set-env name=commitmsg::$(git log --format=%B -n 1 ${{ github.event.after }})

- name: Show commit message
run: echo $commitmsg

- name: Configure AWS Credentials
if: "!contains(env.commitmsg, '[destroy eks]')"
uses: aws-actions/configure-aws-credentials@v1
with:
aws-access-key-id: ${{ secrets.AWSACCESSKEYID }}
aws-secret-access-key: ${{ secrets.AWSSECRETACCESSKEY }}
aws-region: ${{ secrets.AWSREGION }}

- name: Apply Kubernetes manifests
if: "!contains(env.commitmsg, '[destroy eks]')"
working-directory: ./k8s/
run: |
aws eks update-kubeconfig --name ${CLUSTERNAME}
kubectl apply -f samples-bi-service.yaml
kubectl apply -f samples-bi-deployment.yaml
kubectl rollout status deployment/${DEPLOYMENTNAME}
```

当然，我们需要设置 “ terraform ” 用户的凭据（从 /.aws/credentials 文件中获取），让 Github 使用它的机密：

注意工作流程的突出显示部分。 我们可以通过推送包含短语 “ [destroy eks] ” 的提交消息来销毁 EKS 集群。
请注意，我们不会使用这样的提交消息来运行 “ kubernetes apply ” 。
运行管道，但首先要创建一个 .gitignore 文件：

```
 $ cat /.gitignore
.DSStore
terraform/.terraform/
terraform/*.plan
terraform/*.json
$ cd
$ git add .github/ k8s/ terraform/ .gitignore
$ git commit -m "GitHub on EKS"
$ git push
```

在 GitHub 仓库页面的 “ Actions ” 选项卡上监视部署过程。 请等待成功完成。

第一次运行工作流程时，“ Terraform apply ” 步骤需要 15 分钟左右，大约与创建集群的时间一样长。
下次启动时（如果未删除集群），工作流程会快很多。 你可以将此签出：

```
 $ cd
$ git commit -m "Trigger" --allow-empty
$ git push
```

当然，最好检查一下我们做了什么。 这次可以在你的笔记本电脑上使用 IAM “ my-user ” 的凭据：

```
 $ export AWSPROFILE=my-user
$ export AWSREGION=eu-west-1
$ aws sts get-caller-identity
$ aws eks update-kubeconfig --region=eu-west-1 --name=dev-cluster --alias=dev-cluster
$ kubectl config current-context
dev-cluster
$ kubectl get nodes
NAME STATUS ROLES AGE VERSION
ip-10-42-1-125.eu-west-1.compute.internal Ready 6m20s v1.14.8-eks-b8860f

$ kubectl get po
NAME READY STATUS RESTARTS AGE
samples-bi-756dddffdb-zd9nw 1/1 Running 0 6m16s

$ kubectl get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 172.20.0.1 443/TCP 11m
samples-bi LoadBalancer 172.20.33.235 a2c6f6733557511eab3c302618b2fae2-622862917.eu-west-1.elb.amazonaws.com 52773:31047/TCP 6m33s
```

访问
http://a2c6f6733557511eab3c302618b2fae2-622862917.eu-west-1.elb.amazonaws.com:52773/csp/user/DeepSee.UserPortal.Home.zen?$NAMESPACE=USER
（将链接替换为你的外部 IP），然后输入 “ system ” 、 “ SYS ” 并更改默认密码。 您应该看到一系列 BI 仪表板：

点击每个仪表板的箭头可以深入了解：

记住，如果重启 samples-bi pod，所有更改都将丢失。 这是有意的行为，因为这是演示。
如果你需要保留更改，我在 github-gke-zpm-registry/k8s/statefulset.tpl 仓库中创建了一个示例。

完成后，删除你创建的所有内容：

```
 $ git commit -m "Mr Proper [destroy eks]" --allow-empty
$ git push
```

结论

在本文中，我们将 eksctl 实用程序替换成 Terraform 来创建 EKS 集群。 这是向 “ 编纂 ” 您的所有 AWS 基础架构迈出的一步。
我们展示了如何使用 Github Actions 和 Terraform 通过 git push 轻松部署演示应用程序。
我们还向工具箱中添加了 kompose 和 pod 的 postStart 挂钩。
这次我们没有展示 TLS 启用。 我们将在不久的将来完成这项任务。

#AWS #Docker #Kubernetes #云 #容器化 #开发运维 #InterSystems IRIS #Open Exchange
在 InterSystems Open Exchange 上检查相关应用程序

源 URL:https://cn.community.intersystems.com/post/%E4%BD%BF%E7%94%A8-github-actions-%E5%9C%A8-eks-%E4%B8%8A%E9%83%A8%E7%BD%B2-intersystems-iris-%E8%A7%A3%E5%86%B3%E6%96%B9%E6%A1%88