Artigo
Vinicius Maranh... · Mar. 9, 2021 12min de leitura

# Construindo esteiras de CI/CD para implantar o IRIS no Kubernetes utilizando serviços da AWS

## Introdução

Com a transformação digital no mundo dos negócios, novos recursos ou funcionalidades nos softwares oferecidos por uma empresa, podem significar vantagem competitiva. No entanto, se o time de TI não estiver preparado com a cultura, metodologia, práticas e ferramentas corretas, pode ser muito difícil garantir a entrega dessas novas funcionalidades a tempo hábil.

Integração contínua (do inglês " Continuous Integration " , CI) e entrega contínua (do inglês " Continuous Delivery " , CD) incorporam uma cultura, um conjunto de princípios operacionais e uma coleção de práticas que permitem que as equipes de desenvolvimento entreguem esses novos recursos ou novas funcionalidades com maior frequência e confiabilidade.

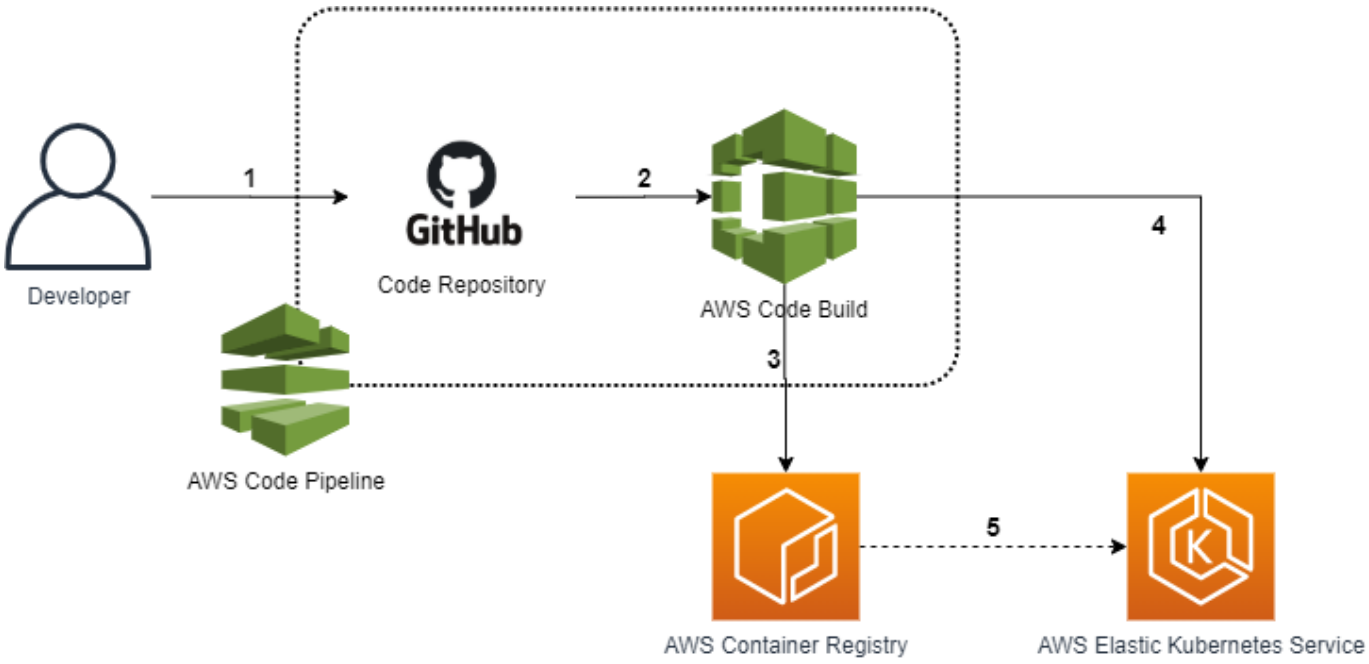
Neste artigo, veremos como criar e configurar uma esteira de CI/CD para uma aplicação desenvolvida na plataforma InterSystems IRIS utilizando serviços disponíveis na Amazon Web Services (AWS). Neste primeiro momento, vamos adaptar um workshop da própria AWS (link no final do artigo) porém, construindo e entregando um serviço REST implementado no InterSystems IRIS. Vamos ainda supor que este é um serviço " stateless " , ou seja, não há persistência de dados.

## Serviços Utilizados e Arquitetura

Neste artigo, utilizaremos os seguintes serviços para compor nossa esteira de CI/CD:

- GitHub: repositório de código
- AWS Code Build: serviço responsável por fazer a construção ( " build " ) da aplicação
- AWS Code Pipeline: serviço responsável por fazer a orquestração dos processos de construção ( " build " ) e implantação ( " deploy " )
- AWS Elastic Container Registry (ECR): repositório das imagens de container construídas
- AWS Elastic Kubernetes Service(EKS): Cluster de Kubernetes gerenciado pela AWS

A arquitetura e interação entre os serviços descritos acima pode ser ilustrada pelo seguinte diagrama:




1. Desenvolvedores " comitam " código para o repositório do GitHub. Quando um " commit " é realizado, o AWS Code Pipeline detecta automaticamente as mudanças e começa a processá-las através da esteira definida
2. O AWS CodeBuild empacota as mudanças do código juntamente com qualquer dependência e constrói a imagem Docker. Opcionalmente, não abordado neste artigo, outro estágio da esteira testa o código e o pacote, também utilizando o AWS CodeBuild.
3. A imagem Docker construída é armazenada no AWS Elastic Container Registry (ECR) após uma construção e/ou teste bem sucedido.
4. AWS CodePipeline realiza a substituição da tag da imagem definida no manifesto de deployment da aplicação no Kubernetes para apontar para a imagem recém construída e chama a API do Kubernetes para fazer a atualização dos pods
5. O Kubernetes, neste caso o AWS Elastic Kubernetes Service (EKS), faz a atualização dos " pods " para rodar a imagem recém construída e disponibilizada no AWS ECR.

O serviço REST implementado no InterSystems IRIS será, portanto, um " pod " dentro do EKS. Neste artigo, reaproveitamos o projeto " iris-rest-api-template " desenvolvido pelo @Evgeny Shvarov, disponível no github: https://github.com/intersystems-community/iris-rest-api-template

Este serviço implementa endpoints para criar, ler, atualizar ou remover uma pessoa (um registro da classe dc.Sample.Person).

Sem mais delongas, vamos pôr a mão na massa para criarmos uma esteira de CI/CD na prática com o InterSystems IRIS!

## Criação dos serviços e projetos

Um pré-requisito para este projeto é possuir um cluster do EKS rodando. Para isso, é possível seguir essa documentação para entender como provisionar um cluster EKS na AWS:

https://docs.aws.amazon.com/eks/latest/userguide/getting-started.html

A criação dos projetos do AWS Code Build e do AWS Code Pipeline será realizada através de um template do AWS Cloud Formation, assim como o repositório do ECR, o " bucket " do S3 e as " Policies " do Identity Access Manager da AWS. Esse template está disponibilizado no repositório deste artigo. O Cloud Formation é um serviço da AWS que permite modelar e configurar recursos da AWS a partir de um arquivo com essas definições, seguindo a metodologia de infraestrutura como código.

## Pré-requisitos da AWS e do EKS

Para fazermos a implantação via esteira de CI/CD no EKS, há algumas configurações que precisamos fazer antes. Primeiramente, será necessário criar uma role no AWS Identity and Access Management (IAM) para permitir que o serviço do CodeBuild possa interagir com o EKS, já que o CodeBuild que executará o comando para fazer o " deploy " no EKS. Vamos fazer a criação desta " role " e adicionar uma " inline Policy " através do AWS CLI. Para informações sobre como instalar o AWS CLI, seguir esta documentação:

https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html

Com o AWS CLI instalado, executar os seguintes comandos:

```
TRUST="{ \"Version\": \"2012-10-17\", \"Statement\": [ { \"Effect\": \"Allow\", \"Pri
ncipal\": { \"AWS\": \"arn:aws:iam::${ACCOUNT_ID}:root\" }, \"Action\": \"sts:AssumeR
ole\" } ] }"
```

```
echo '{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": "eks:D
escribe*", "Resource": "*" } ] }' > /tmp/iam-role-policy
```

```
aws iam create-role --role-name EksDemoSE --assume-role-policy-
document "$TRUST" --output text --query 'Role.Arn'
```

```
aws iam put-role-policy --role-name EksDemoSE --policy-name eks-describe --policy-
document file:///tmp/iam-role-policy
```

Agora que temos a " role " criada, é necessário adicionar esta role a ConfigMap aws auth no EKS. Para isso, execute os seguintes comandos (não remova espaços em branco):

```
ROLE="    - rolearn: arn:aws:iam::${ACCOUNT_ID}:role/EksDemoSE\n      username: build
\n      groups:\n        - system:masters"
```

```
kubectl get -n kube-system configmap/aws-auth -o yaml | awk "/mapRoles: \|/{print;pri
nt \"$ROLE\";next}1" > /tmp/aws-auth-patch.yml
```

```
kubectl patch configmap/aws-auth -n kube-system --patch "$(cat /tmp/aws-auth-
patch.yml)"
```

Com isso, o CodeBuild já consegue interagir com o EKS através dos comandos " kubectl ".

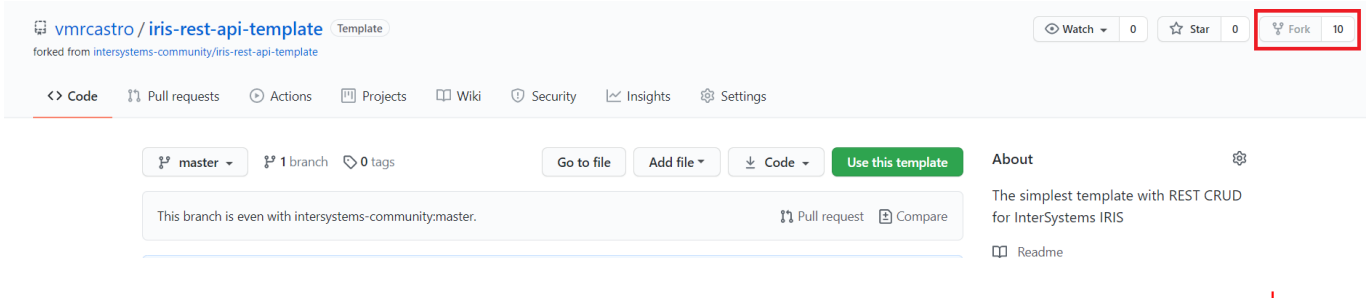
## Repositório GitHub

O próximo passo é criar um repositório no GitHub. É possível fazer um fork do repositório utilizado neste artigo para já criar um repositório juntamente com todos os arquivos necessários.

Para isso, basta fazer o login com sua conta do GitHub, acessar o repositório deste artigo em

https://github.com/vmrcastro/iris-rest-api-template

e clicar em Fork:

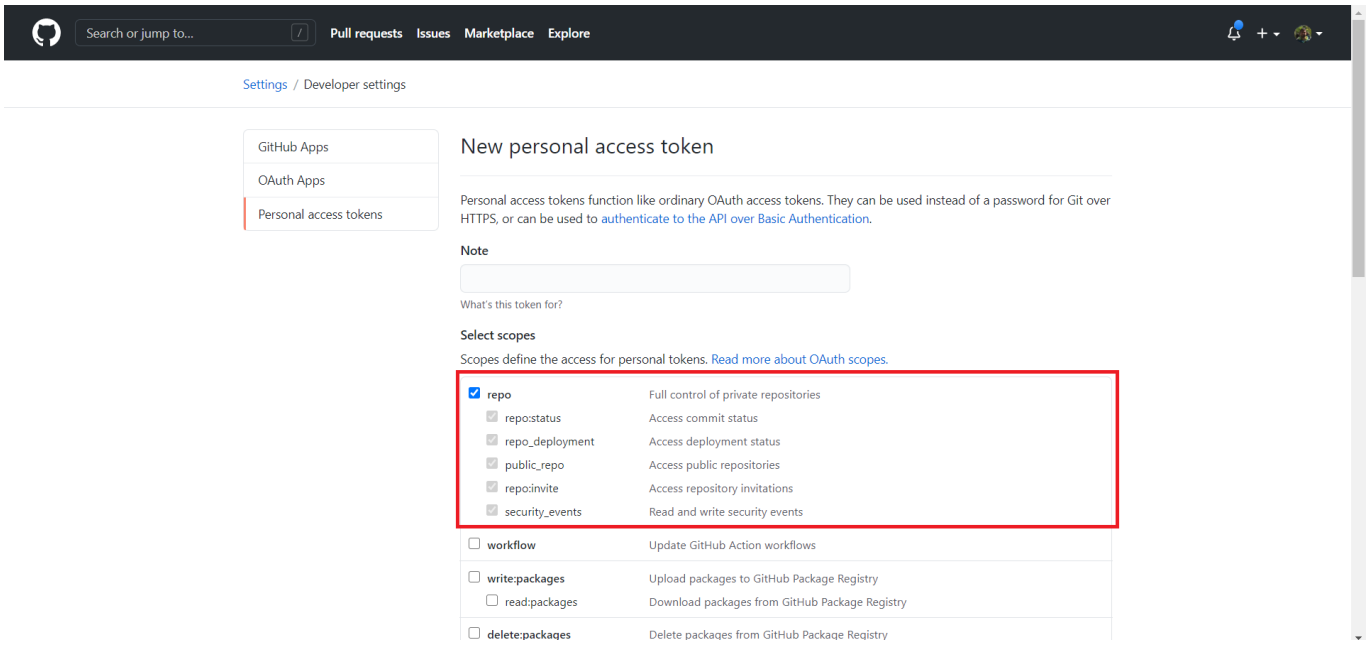



Para permitir o CodePipeline receber " callbacks " do GitHub, de forma que o gatilho para disparar a execução da nossa esteira de CI/CD seja quando um commit for feito para o repositório, é necessário gerar um token de acesso do GitHub.

Para isso acesse a página

https://github.com/settings/tokens/new

Marque a caixa " repo "



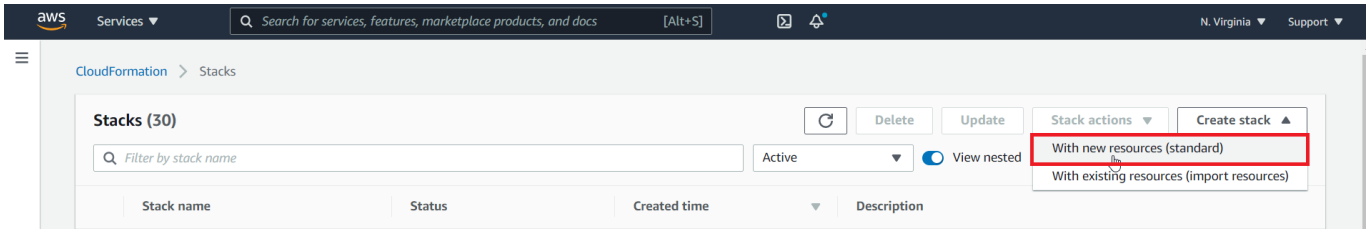

E clique no botão " Generate Token " no final da página.

Na página seguinte, copie o token gerado e grave em um lugar seguro. Necessitaremos dele nos próximos passos. Se atente para o fato de que este token é exibido somente uma vez após a criação e não é possível recuperá-lo posteriormente.

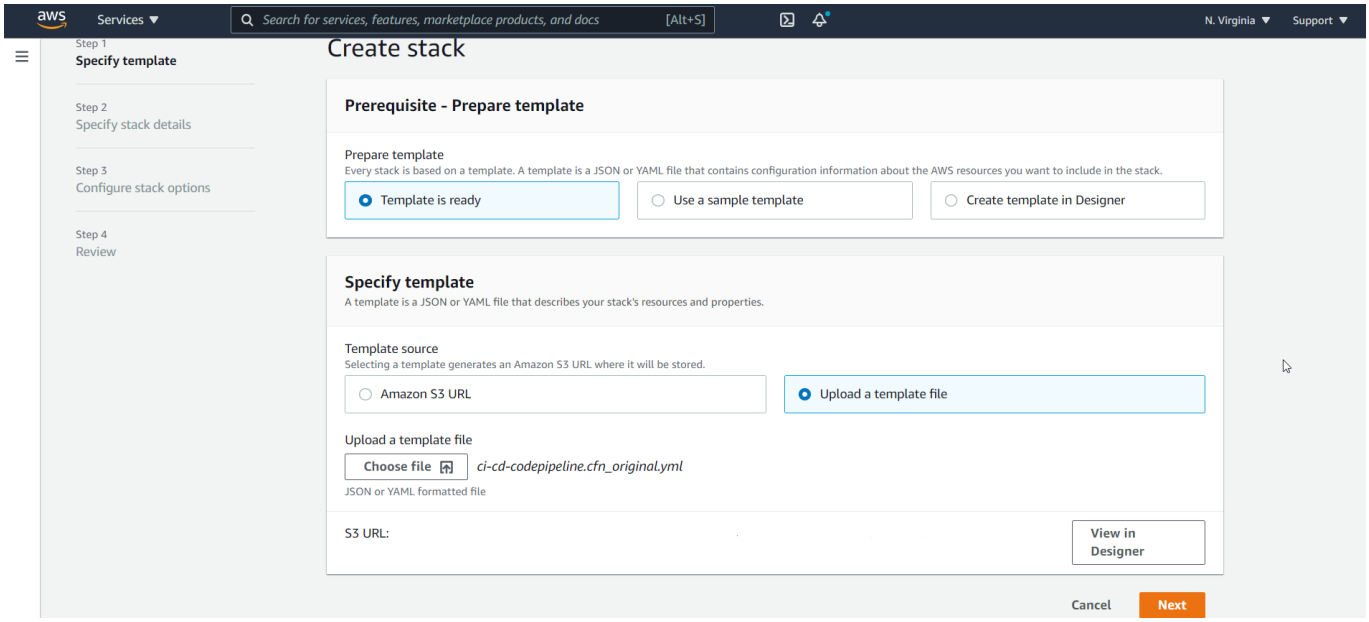
## Criação e configuração do CodePipeline

Como havia sido mencionado anteriormente, a criação e configuração dos projetos do CodePipeline e do

CodeBuild serão realizados através do Cloud Formation. Portanto, abra a console do Cloud Formation e clique em " Create Stack " e, em seguida, " With new resources (standard) " .




Na seção " Prepare Template " , deixe marcada a caixa " Template is ready " e, na seção " Specify Template " , marque a caixa " Upload template file " . Clique no botão " Choose file " e aponte para o arquivo " ci-cd-codepipeline.cfn.yml " na pasta " IaC " do projeto presente no repositório do GitHub. Em seguida, clique em " Next " .



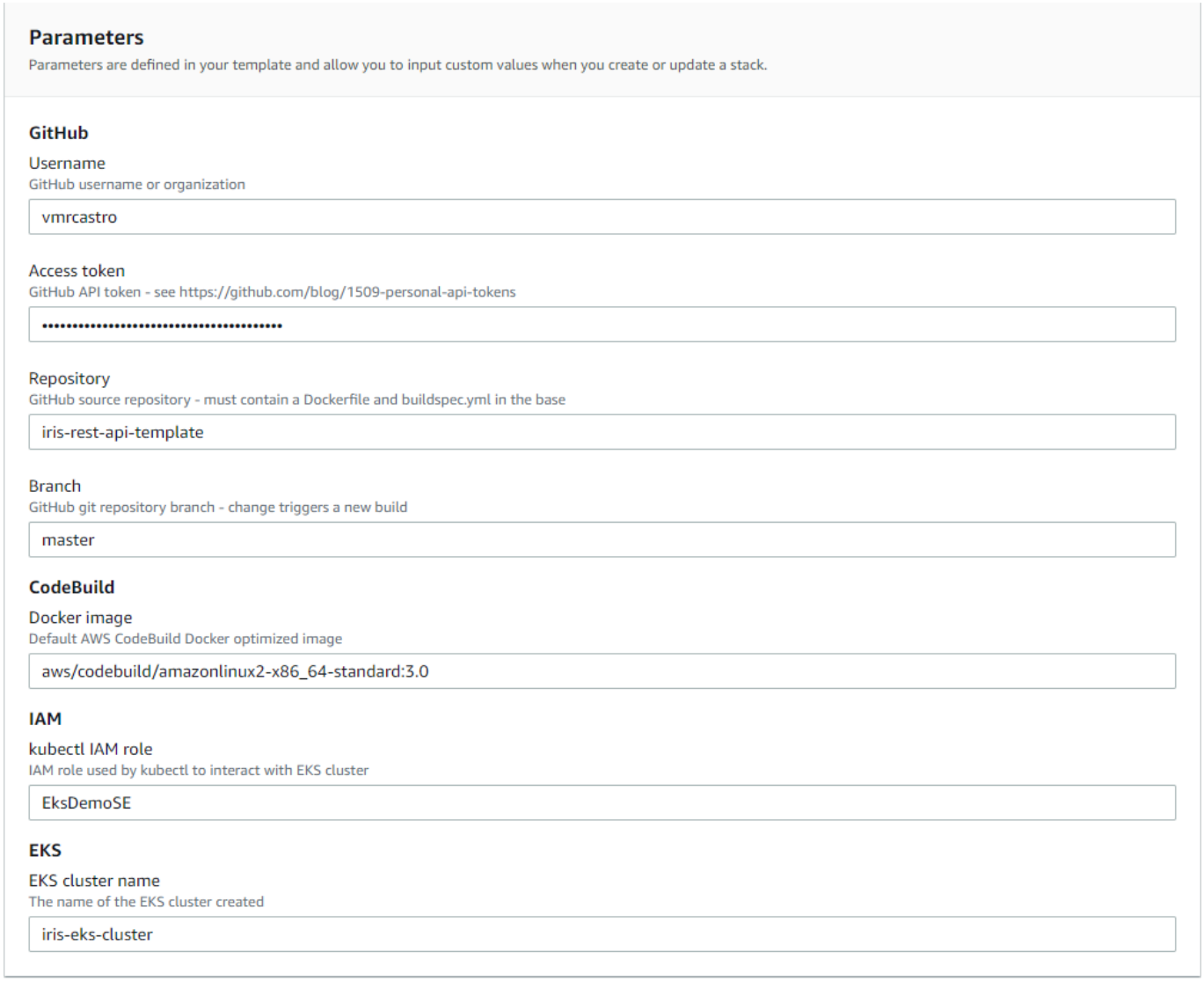

Na tela seguinte, na seção " Stack name " , dê um nome para o projeto. Na seção " Parameters " , altere os campos de acordo com as instruções abaixo. Os campos não mencionados, devem continuar com o valor padrão.

GitHub

- Username: seu username do GitHub
- Access Token: Access Token para o Code Pipeline interagir com seu repositório do git, gerado no final da seção anterior deste artigo
- Repository: o nome do seu repositório do GitHub. Neste caso, é o seu repositório oriundo do fork, demonstrado anteriormente neste artigo, cujo nome é " iris-rest-api-template "
- Branch: defina qual será a branch que o CodePipeline ficará " ouvindo " para disparar a execução da esteira assim que houver um commit

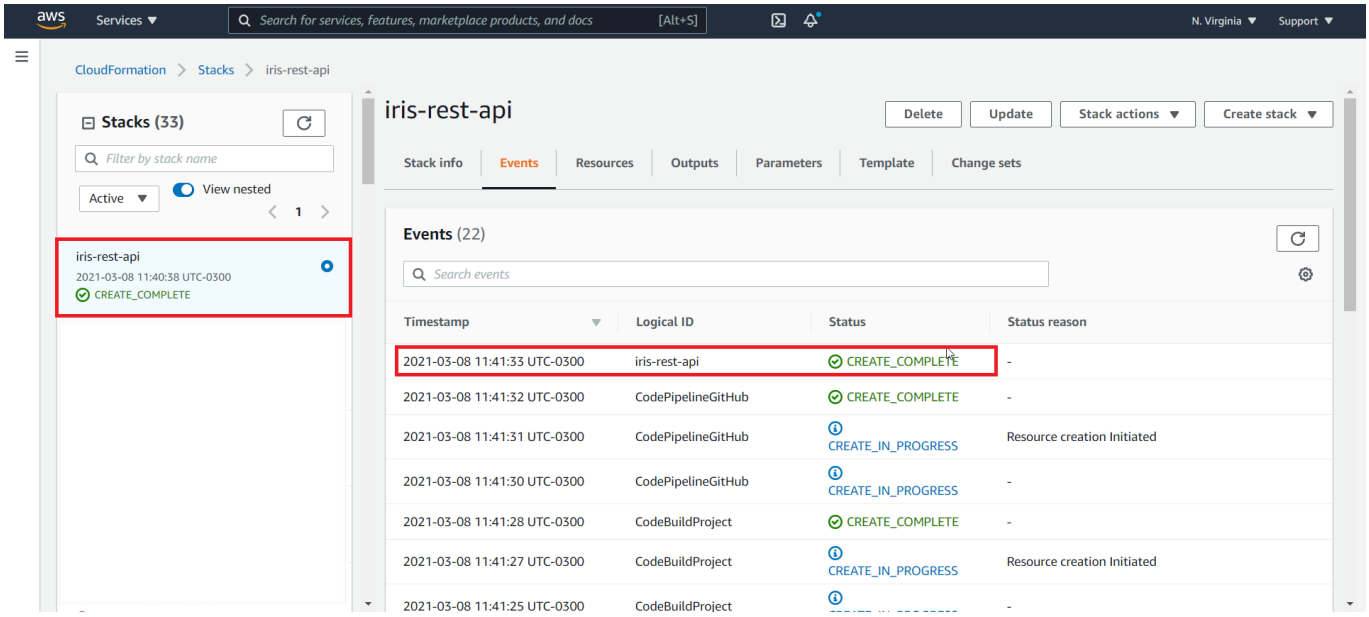
EKS

- EKS cluster name: nome do cluster de EKS já provisionado na AWS



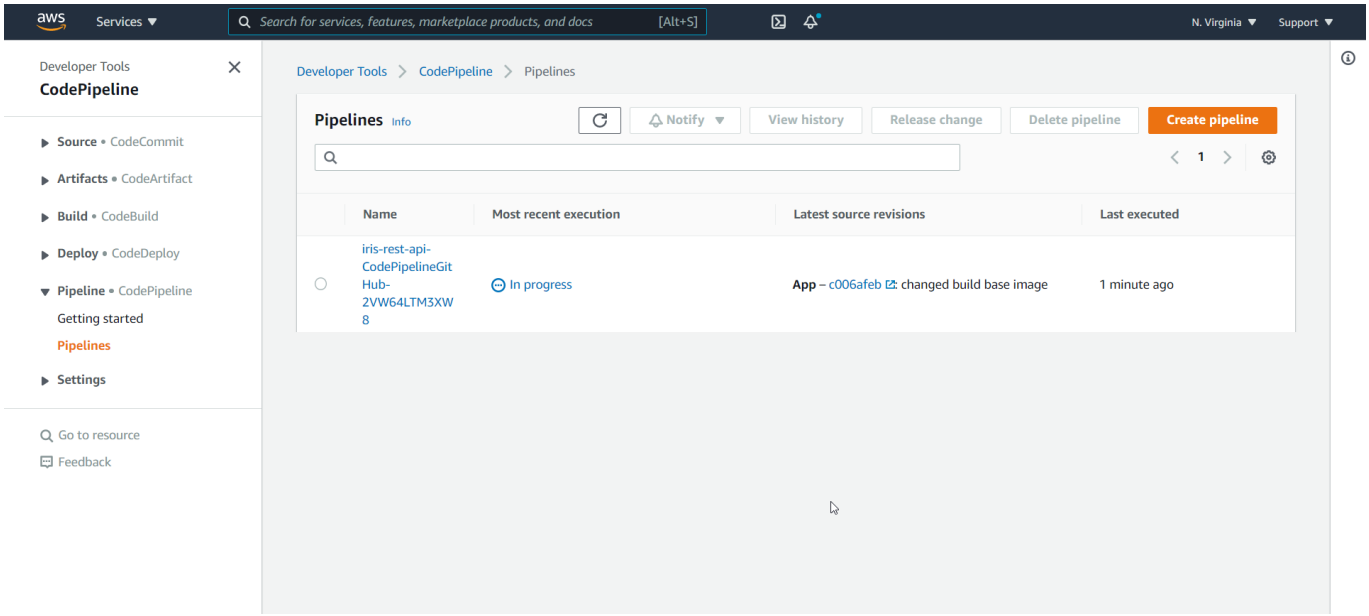

Uma vez todos os campos preenchidos, clique em " Next " . Na tela seguinte, não é necessário fazer nenhuma alteração, somente clicar em " Next " . Finalmente, na última tela, confira os valores definidos e, se estiver tudo certo, marque a caixa " I acknowledge that AWS CloudFormation might create IAM resources. " e clique no botão " Create stack " .

Aguarde até o seu projeto do Cloud Formation ficar com o status de " CREATE_COMPLETE " :

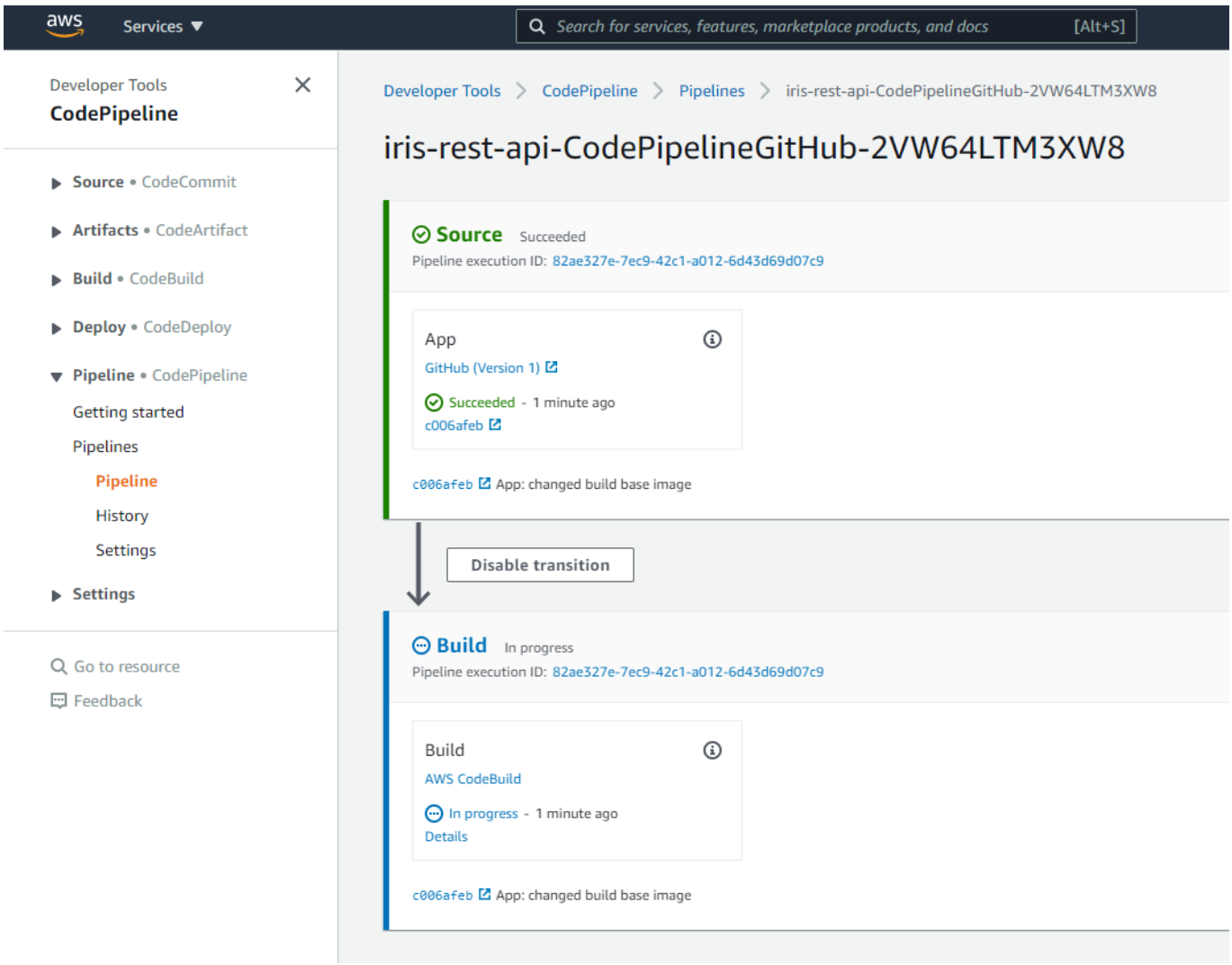



# Acompanhando a construção e entrega do projeto no Code Pipeline

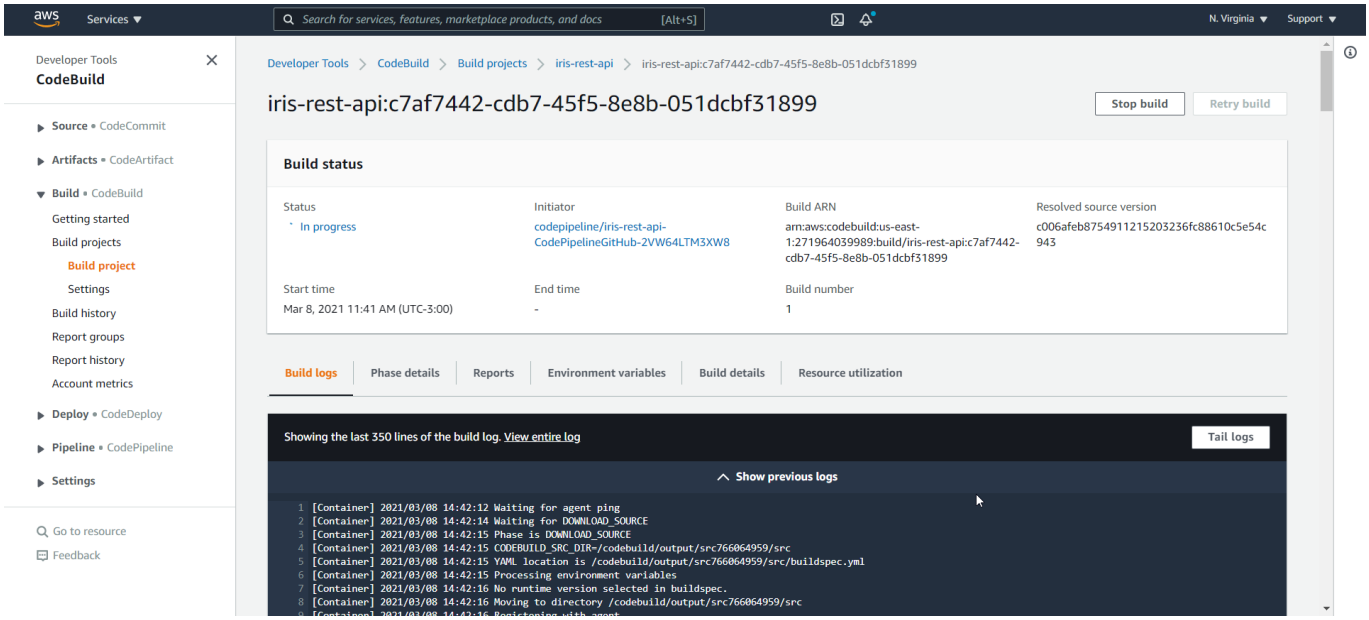
Uma vez criado todos os recursos através do Cloud Formation, o processo de construção e implantação ( “ deploy ” ) é executado automaticamente pela primeira vez. Para acompanhar, vá até a console do Cloud Formation, expanda a seção “ Pipeline ” no menu esquerdo, clique em “ Pipelines ” e localize o seu projeto.




Se esta página foi aberta logo após a execução bem sucedida do Cloud Formation ou logo após um commit no seu repositório GitHub configurado no projeto, provavelmente o projeto estará com o status de “ ’ In Progress ” . Para acompanhar detalhes da execução, clique no nome do seu projeto, e a seguinte página deve abrir:

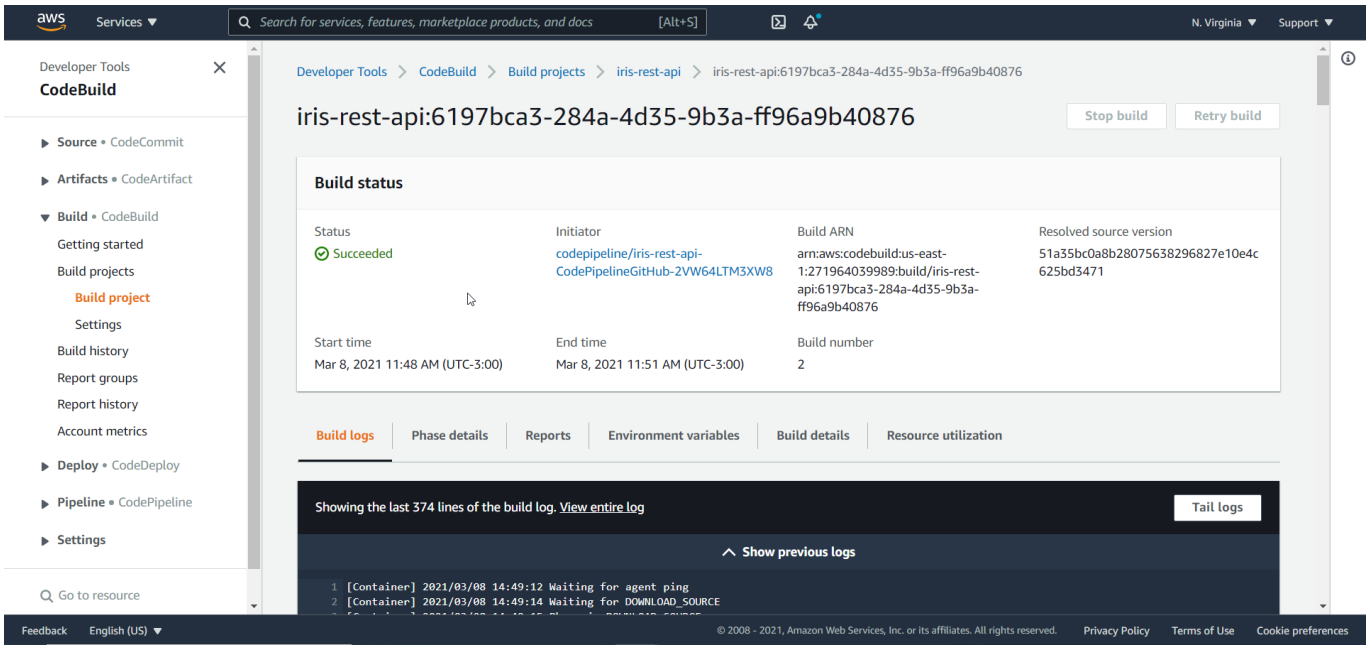



Na tela acima, podemos observar que a etapa “ Source ” foi executada com sucesso. Nesta etapa, o CodePipeline faz o download do repositório do GitHub configurado, armazena no S3 e disponibiliza para o CodeBuild executar a etapa “ Build ” , que está em execução. Clicando em “ Details ” , é possível acompanhar os logs da etapa, conforme mostrado na imagem a seguir:



Ainda é possível ficar " seguindo " os logs desta etapa, clicando no botão " Tail logs " :


Uma vez a etapa " Build " completada com sucesso, será possível observar o status " Succeded " :




O término com sucesso desta etapa significa que o projeto foi construído e foi feito o deploy no cluster de Kubernetes (EKS) configurado. Para verificarmos, podemos executar alguns comandos a partir de um " shell " que tenha o " kubectl " configurado para interagir com o EKS.

Primeiramente, vamos verificar se o deployment do nosso projeto, denominado " demose " , está rodando:

```
$ kubectl get deployments

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE

demosebr                      1/1     1            1           131m
```

Podemos verificar também os pods que estão em execução e sua respectiva " idade " :

```
$ kubectl get pods

NAME                                    READY   STATUS    RESTARTS   AGE

demosebr-7f5dbdfb46-mkgsg               1/1     Running   0          2m8s
```

Aparentemente, nosso projeto foi entregue com sucesso ao EKS!

Vamos agora testar nosso serviço REST. Para isso, precisamos verificar o endereço do " service " do Kubernetes que será responsável por enviar requisições externas aos respectivos " pods " . Para isso, vamos executar o comando e copiar o conteúdo do campo da linha " demosebr " e da coluna " EXTERNAL-IP " . Este é o endereço para qual devemos enviar as requisições.

```
$ kubectl get svc

NAME                          TYPE           CLUSTER-IP      EXTERNAL-
IP                               PORT(S)                            AGE

demosebr                      LoadBalancer   10.100.103.71   a2472[…]76.us-
east-1.elb.amazonaws.com   52773:31752/TCP,1972:31866/TCP   130m
```

Veja que as portas expostas neste serviço foram a 52773 (webserver) e a 1972 (superserver). A configuração desse serviço é definida no arquivo " deploy.yml " , na raíz do nosso repositório.

Portanto, agora só resta enviar uma requisição para testar nosso serviço REST rodando no InterSystems IRIS. Para isso, eu utilizarei o cURL (mas utilize o cliente REST que você está mais habituado).

```
$ curl -v -u '_SYSTEM:SYS' -X POST 'http:// a2472[…]76.us-east-1.elb.amazonaws.com:52
773/crud/persons/' -H  'accept: application/json' -H  'Content-Type: application/json
' -d '{"Name":"Elon Mask","Title":"CEO","Company":"Tesla","Phone":"123-123-1233","DOB
":"1982-01-19"}'
```

```
*   Trying 52.[…].100:52773...

* TCP_NODELAY set

* Connected to a2472[…]76.us-east-1.elb.amazonaws.com (52.[…].100) port 52773 (#0)

* Server auth using Basic with user '_SYSTEM'

> POST /crud/persons/ HTTP/1.1

> Host: a2472[…]76.us-east-1.elb.amazonaws.com:52773

> Authorization: Basic X1NZU1RFTTpTWVM=

> User-Agent: curl/7.68.0

> accept: application/json

> Content-Type: application/json

> Content-Length: 94
```

```
>

* upload completely sent off: 94 out of 94 bytes

* Mark bundle as not supporting multiuse

< HTTP/1.1 204 No Content

< Date: Mon, 08 Mar 2021 17:26:30 GMT

< Server: Apache

< CACHE-CONTROL: no-cache

< EXPIRES: Thu, 29 Oct 1998 17:04:19 GMT

< PRAGMA: no-cache

<

* Connection #0 to host a2472[…]76.us-east-1.elb.amazonaws.com left intact
```

E voila! Recebemos o código HTTP 204 de resposta, então aparentemente o serviço REST está funcional e o " Elon Mask " foi cadastrado! Para termos certeza, vamos fazer outra requisição REST para listar todas as pessoas cadastradas, desta vez, sem o verbose do cURL para imprimirmos somente o corpo da resposta:

```
$ curl -u '_SYSTEM:SYS' -X GET 'http:// a2472[…]76.us-
east-1.elb.amazonaws.com:52773/crud/persons/all' -H  'accept: application/json'
```

```
[{"Name":"Elon Mask","Title":"CEO","Company":"Tesla","Phone":"123-123-1233","DOB":"19
82-01-19"}]
```

Como pode ser observado, o serviço REST do IRIS esta 100% funcional!
**Obs:** Não utilize autenticação básica (usuário e senha) em requisições HTTP e não utilize o usuário SYSTEM. Esta é uma configuração totalmente insegura.

## Entendendo o funcionamento da esteira

Se você chegou até aqui, você pode estar se perguntando:

Como ou onde é definido as etapas para fazer a construção e implantação ( " deploy " ) do projeto?

Esta é uma excelente pergunta e você verá que as respostas estão em arquivos dentro do próprio repositório!

Tudo começa na definição da etapa " Build " do nosso projeto do CodePipeline. Esta etapa é definida pelo arquivo denominado " buildspec.yml " . Dentro deste arquivo, dividimos a etapa " Build " em 3 fases: " prebuild " , " build " e " postbuild " . Cada fase irá conter um conjunto de comandos diferentes, também definidos dentro do mesmo arquivo.

Na fase de " prebuild " , basicamente definimos uma " tag " para a imagem do nosso serviço REST que estamos construindo, substituímos a string " CONTAINERIMAGE " no arquivo " deploy.yml " pelo nome da imagem que estamos fazendo o build e fazemos o login no repositório de imagens Docker da AWS, o ECR, para posteriormente enviarmos a imagem construída para lá.

O arquivo " deploy.yml " mencionado anteriormente também está presente na raíz do repositório. Ele é o manifesto

do Kubernetes, que indica todos os recursos que devem ser criados pelo Kubernetes para implantar nossa aplicação.

Já na fase de " build " , não há nenhuma novidade. É nela em que é feita a construção da imagem Docker da nossa aplicação através do comando " docker build " . Lembrando que este comando irá construir a imagem a partir do arquivo " Dockerfile " , também presente na raíz do projeto. Além de construir a imagem, estamos etiquetando com a " tag " definida na fase anterior.

Finalmente, na fase de " postbuild " , fazemos o upload da imagem construída para o ECR, configuramos o kubectl para o EKS desejado e aplicamos o arquivo de manifesto do Kubernetes, o " deploy.yml " .

## Considerações finais

Neste artigo prático foi abordado como construir uma esteira CI/CD para fazer a implantação do InterSystems IRIS no Kubernetes utilizando os serviços da AWS. A aplicação escolhida é simples e a implantação realizada não contempla alguns aspectos, como persistência de dados, por exemplo. O objetivo deste artigo é dar uma ideia dos primeiros passos para implementar uma esteira CI/CD. Caso tenha interesse, poste nos comentários deste artigo suas dúvidas ou detalhes que você deseja entender mais.

Até a próxima!

## Fontes consultadas e utilizadas

https://community.intersystems.com/post/continuous-delivery-your-intersystems-solution-using-gitlab-part-i-git

https://www.eksworkshop.com/intermediate/220codepipeline/

https://openexchange.intersystems.com/package/iris-rest-api-template

#AWS #Conteinerização #DevOps #Docker #Entrega Contínua #GitHub #Integração Contínua #Kubernetes #Nuvem #InterSystems IRIS

**URL de origem:**https://pt.community.intersystems.com/post/construindo-esteiras-de-cicd-para-implantar-o-iris-no-kubernetes-utilizando-servi%C3%A7os-da-aws