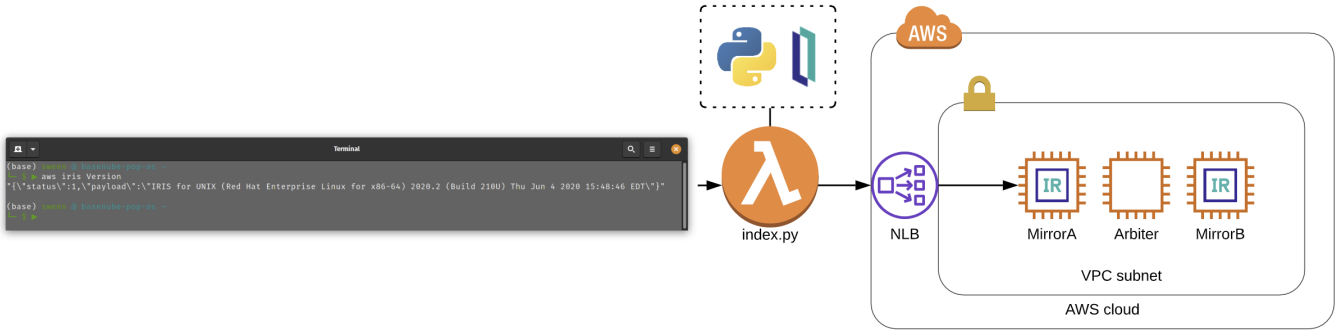
Artículo
Ricardo Paiva · 19 feb, 2021 Lectura de 7 min

La API nativa de IRIS para Python en AWS Lambda

Si está buscando una forma ingeniosa para integrar su solución de IRIS en el ecosistema de Amazon Web Services, en una aplicación sin servidor o en Boto3 (un potente script de Python), usar la API nativa de IRIS para Python podría ser el camino a seguir. No es necesario que invierta demasiado tiempo en la implementación de una producción hasta que deba acercarse y obtener algo o establecer algo en IRIS para hacer que su aplicación ejecute su característica más sobresaliente, así que esperamos este artículo sea útil y desarrolle algo aunque solamente usted pueda usarlo, ya que eso también es importante.



Si necesita algunas excusas para implementarla, utilice las siguientes:

- Debe cargar el Activador que generará previamente los tokens en Cognito para buscar y completar el contexto de la identificación del paciente en el token. Esto con la finalidad de utilizar una solución basada en SMART on FHIR(R), la cual implementará un flujo de trabajo en OAUTH2.
- Quiere publicar el suministro de la configuración para los parámetros en IRIS que se basa en el tipo de instancia, grupo de nodo, Toaster o el clúster ECS que inició para ejecutar IRIS en el anillo cero.
- Quiere impresionar a su familia y amigos en Zoom con sus habilidades para administrar IRIS sin que su shell tenga que abandonar la CLI de AWS.

Lo más importante

Aquí proporcionamos una función lambda de AWS que se comunica con IRIS y brindaremos algunos ejemplos sobre cómo utilizarla, además indicaremos cómo podemos interactuar con ella en varias funciones con la esperanza de que todos podamos discutir sobre ella y **publicarla en pip para facilitar las cosas.**

¿¿¿¿Tiene problemas????

Consulte el Stream

https://www.youtube.com/embed/mA0VzKOYhBk
*[Este es un enlace integrado, pero no puede ver el contenido integrado directamente en el sitio porque rechazó las cookies que se necesitan para acceder a él. Para ver el contenido integrado, debe aceptar todas las cookies desde la **Configuración de cookies**]*

En todos los casos...

Para participar en la diversión, necesitará eliminar algunas cosas de sus planes.

Interconexión

Tiene a IRIS ejecutándose en cualquier lugar que desee, funcionando en una AWS VPC con exactamente dos subredes y un grupo de seguridad que permite el acceso al super servidor que inicia IRIS... utilizamos 1972 por razones nostálgicas y por el simple hecho de que InterSystems se tomó el tiempo para registrar ese puerto con IANA, y si agrega el nuevo puerto en /etc/services sin registrarlo, sufrirá las mismas consecuencias que si arrancara la etiqueta de un colchón nuevo. En nuestro caso, es un conjunto de réplicas de las instancias EC2 con las comprobaciones de estado adecuadas en torno a un balanceador de carga para la red AWS v2.

Ejemplo de una clase importada

De alguna forma consiguió crear e importar la clase a en la raíz de este repositorio con el namespace %SYS en su instancia de IRIS. A continuación, se muestra el ejemplo de una clase que impulsa la salida anterior. Si se pregunta por qué necesitamos una clase para importar aquí, consulte la siguiente nota donde el enfoque recomendado es proporcionar algunas clases envolventes para utilizarlas con Python.

> Nota de los documentos: Aunque estos métodos también pueden utilizarse con las clases que se definieron en la Biblioteca de clases de InterSystems, una de las prácticas recomendadas consiste en llamarlos indirectamente, desde el interior de una clase o rutina definida por el usuario. Muchos métodos de clase devuelven solo un código de estado, y transmiten los resultados reales a un argumento (al que no es posible acceder por la API nativa). Las funciones definidas por el sistema (incluidas como parte de las Funciones de ObjectScript en la Referencia de ObjectScript) no pueden llamarse directamente.

Ejemplo de la clase:

```
Class ZDEMO.IRIS.Lambda.Operations Extends %Persistent
{
  ClassMethod Version() As %String
  {
   Set tSC = 0

   Set tVersion = $ZV
   if ( tVersion '="" ) { set tSC = $$$OK }

   Set jsonret = {}
   Set jsonret.status = tSC
   Set jsonret.payload = tVersion

   Quit jsonret.%ToJSON()
  }
}
```

Tenga en cuenta que decidí trabajar según lo establecido en los métodos, de modo que siempre emite un objeto JSON como respuesta, lo cual también me permite enviar el estado y posiblemente subsanar las deficiencias que implica devolver algunas cosas como referencia.

AWS Access

Obtenga algunas claves de acceso IAM que le permitirán suministrar e invocar la función Lambda con la que cambiaremos el mundo.

Comprobación previa:

```
IRIS            [ $$$OK ]
VPC             [ $$$OK ]
Subnets         [ $$$OK ]
Security Group  [ $$$OK ]
IAM Access      [ $$$OK ]
Imported Class  [ $$$OK ]
```

$$$OK, **Lesgo**.

Empaquetar la API nativa de IRIS para Python con el fin de utilizarla en la función Lambda

Esta parte sería fantástica si fuera un paquete pip, especialmente si solo es para Linux, ya que las funciones Lambda de AWS se ejecutan en Linux Boxen. Cuando realizamos esta asignación la API no estaba disponible por medio de pip, pero somos hábiles y podemos lanzar nuestro propio paquete.

```
mkdir iris_native_lambda
cd iris_native_lambda
wget https://github.com/intersystems/quickstarts-python/raw/master/Solutions/nativeAP
I_wheel/irisnative-1.0.0-cp34-abi3-linux_x86_64.whl
unzip nativeAPI_wheel/irisnative-1.0.0-cp34-abi3-linux_x86_64.whl
```

A continuación, cree connection.config

Por ejemplo: connection.config

Cree su controlador, index.py o utilice el que se encuentra en la carpeta de ejemplos, en el repositorio con demostraciones de GitHub. Tenga en cuenta que la versión de demostración utiliza tanto variables de entorno como un archivo externo para guardar la información de conectividad de IRIS.

Por ejemplo:
index.py

Ahora comprímalo para utilizarlo:

```
zip -r9 ../iris_native_lambda.zip *
```

Cree un bucket S3, y cargue la función zip en él.

```
cd ..
aws s3 mb s3://iris-native-bucket
s3 sync iris_native_lambda.zip s3://iris-native-bucket
```

> Con esto concluye el empaquetado de la API y el controlador para utilizarlos como una función Lambda de AWS.

Ahora, haga clic hasta que la consola termine para crear la función, o utilice algo como Cloudformation para realizar el trabajo:

```
  IRISAPIFunction:
    Type: "AWS::Lambda::Function"
    DependsOn:
      - IRISSG
      - VPC
    Properties:
      Environment:
        Variables:
          IRISHOST: "172.31.0.10"
          IRISPORT: "1972"
          NAMESPACE: "%SYS"
          USERNAME: "intersystems"
          PASSWORD: "lovetheyneighbor"
      Code:
        S3Bucket: iris-native-bucket
        S3Key: iris_native_lambda.zip
      Description: "IRIS Native Python API Function"
      FunctionName: iris-native-lambda
      Handler: "index.lambda_handler"
      MemorySize: 128
      Role: "arn:aws:iam::8675309:role/BeKindtoOneAnother"
      Runtime: "python3.7"
      Timeout: 30
      VpcConfig:
        SubnetIds:
          - !GetAtt
            - SubnetPrivate1
            - Outputs.SubnetId
          - !GetAtt
            - SubnetPrivate2
            - Outputs.SubnetId
        SecurityGroupIds:
          - !Ref IRISSG
```

Eso fue MUCHO, pero ahora puede volverse loco, llamar a IRIS mediante la función lambda con Python y cambiar el mundo.

¡Vamos a iniciarlo!

De la forma en que se implementó lo anterior, se espera que la función sea aprobada en un evento del objeto que esté menos estructurado para reutilizarlo, puede ver la idea en el ejemplo para el evento del objeto que se muestra a continuación:

```
  {
    "method": "Version",
    # important, if method requires no args, enforce "none"
    "args": "none"
    # example method with args, comma seperated
    # "args": "thing1, thing2"
  }
```

ahora puede hacerlo, si tiene cierta tolerancia para los ejemplos de la línea de comandos, eche un vistazo en la ejecución que se muestra a continuación utilizando la CLI de AWS:

```
(base) sween @ basenube-pop-os ~/Desktop/BASENUBE
?? $ &#x25b6; aws lambda invoke --function-name iris-native-lambda --payload '{"metho
d":"Version","args":"none"}' --invocation-type RequestResponse --cli-binary-
format raw-in-base64-out --region us-east-2 --profile default /dev/stdout
{{\"status\":1,\"payload\":\"IRIS for UNIX (Red Hat Enterprise Linux for x86-64) 2020
.2 (Build 210U) Thu Jun 4 2020 15:48:46 EDT\"}"
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

Ahora bien, si vamos un poco más lejos, la CLI de AWS admite alias, así que cree uno y podrá jugar integrando completamente su increíble comando con AWS. Este es un ejemplo del alias para la CLI:

```
?? $ &#x25b6; cat ~/.aws/cli/alias
[toplevel]
whoami = sts get-caller-identity

iris =
  !f() {
   aws lambda invoke \
     --function-name iris-native-lambda \
     --payload \
     "{\"method\":\""${1}"\",\"args\":\"none\"}" \
     --invocation-type RequestResponse \
     --log-type None \
     --cli-binary-format raw-in-base64-out \
     gar.json > /dev/null
     cat gar.json
     echo
     echo
  }; f
```

....y ahora puede hacerlo...

¡Manténgase a salvo!¬¡Todos los argumentos técnicos son bienvenidos!

#InterSystems IRIS #InterSystems IRIS for Health

URL de fuente:https://es.community.intersystems.com/post/la-api-nativa-de-iris-para-python-en-aws-lambda