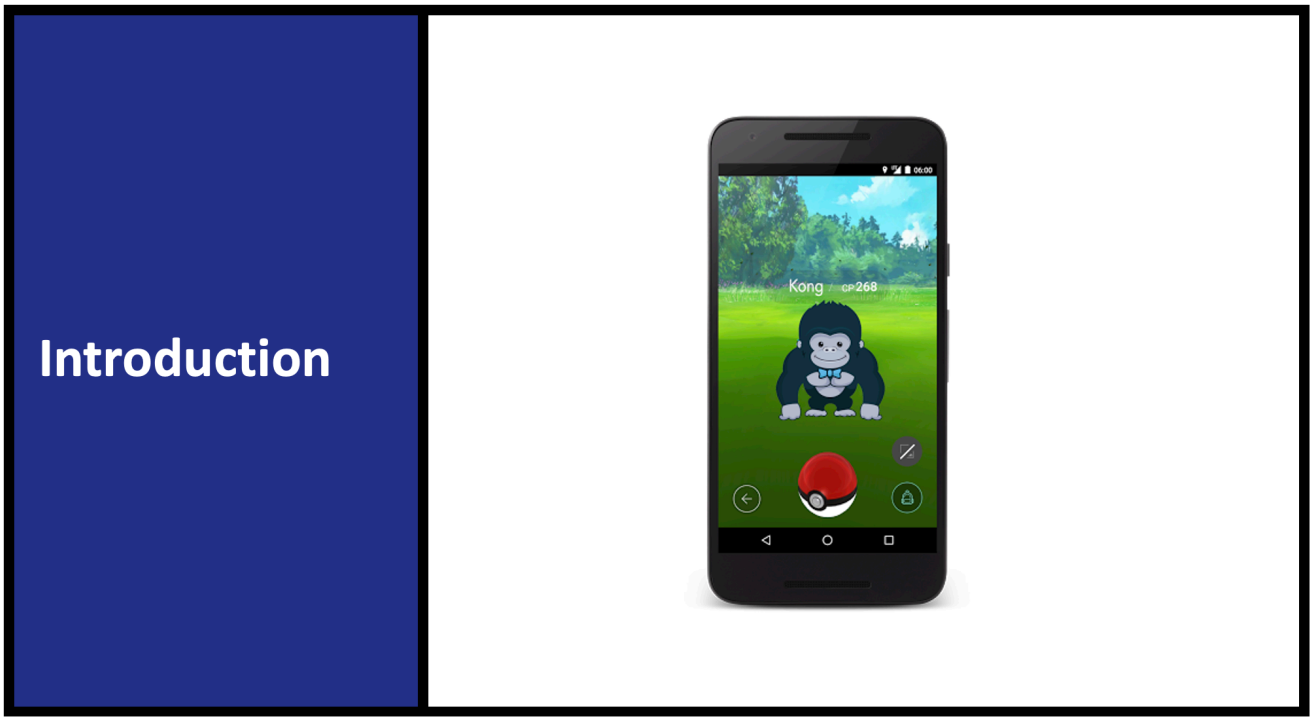
記事
Toshihiko Minamoto · 2021年4月26日 41m read

Open Exchange

ゼロから使いこなす IAM（InterSystems API Manager）



この記事には、IAM の基本概念を学習するための、教材、例、演習が含まれます。

すべてのリソースはこちらの git から入手できます: https://github.com/grongierisc/iam-training

ソリューションは training ブランチにあります。

この記事では、次の点について説明します。

- 1. 入門
  - 1.1. IAM とは？
  - 1.2. API 管理とは？
  - 1.3. IAM ポータル
  - 1.4. このトレーニングの流れ
- 2. インストール
  - 2.1. インストールには何が必要？
  - 2.2. IAM と IRIS の連携の仕組み
  - 2.3. セットアップ
  - 2.4. IAM のインストール
  - 2.4.1. Iris イメージ
  - 2.4.2. IAM イメージ
  - 2.4.3. Docker ファイルを更新する
  - 2.4.4. docker-compose を更新する
  - 2.4.5. オプション: .env として IRISPASSWORD を追加する
  - 2.4.6. テストしよう！
- 3. その 1: サービス/ルート

    - 3.1. サービスを作成する
    - 3.2. ルートを作成する
    - 3.3. テストしよう！
- 4. その 2: プラグインの使用
    - 4.1. プラグインをサービスに追加する
    - 4.2. テストしよう！
- 5. その 3: 独自認証の追加
    - 5.1. コンシューマを追加する
    - 5.2. Basic 認証プラグインを追加する
    - 5.3. ACL プラグインを追加する
    - 5.4. ACL とログイン情報で USER を構成する
    - 5.5. テストしよう！
- 6. 演習: レート制限
    - 6.1. ソリューション
- 7. 開発者ポータル
    - 7.1. 概要
    - 7.2. 有効にしよう！
    - 7.3. 最初の仕様を追加する
    - 7.4. テストしよう！
    - 7.5. 演習
    - 7.5.1. ソリューション
- 8. 開発者ポータル（その 2）：認証
    - 8.1. Basic 認証を有効にする
    - 8.2. アクセスを制限する
    - 8.2.1. ロールを作成する
    - 8.2.2. 仕様にロールを追加する
    - 8.2.3. テストしよう！
        - 8.2.3.1. 新しい開発者を登録する
        - 8.2.3.2. この開発者を承認する
        - 8.2.3.3. この開発者のロールを追加する
    - 8.3. 開発者の Oauth2 を追加する
    - 8.3.1. その 1: Basic 認証を削除する
    - 8.3.2. その 2: application-registration プラグインを追加する
    - 8.3.3. リンクサービスとドキュメント
        - 8.3.3.1. テストしよう！
- 9. 安全な管理ポータル
    - 9.1. 管理者を作成する
    - 9.2. Kong Manager の Basic 認証を有効にする
    - 9.3. RBAC で Kong Admin API を使用する
    - 9.3.1. トークンで管理者ユーザーを作成する
- 10. プラグイン
    - 10.1. コミュニティプラグインをインポートする
    - 10.1.1. コミュニティプラグインで新しい Kong/IAM docker イメージを構築する
    - 10.1.2. テストしよう！
        - 10.1.2.1. 使ってみよう！
    - 10.2. 新しいプラグインを作成する
    - 10.2.1. ファイル構造
        - 10.2.1.1. handler.lua
        - 10.2.1.1.1. 例
        - 10.2.1.2. schema.lua
        - 10.2.1.3. *.rockspec
    - 10.2.2. 構築しよう
    - 10.2.3. ヒント
- 11. CI/CD
    - 11.1. postman コレクションを作成する
    - 11.1.1. IAM が起動しているか
    - 11.1.2. 古いデータを削除する
    - 11.1.3. サービス/ルートを作成する
        - 11.1.3.1. ヒント

- 11.2. newman で実行する

1. 入門

1.1. IAM とは？

IAM は、InterSystems API Manager の略で、**Kong Enterprise Edition** が土台となっています。

このため、Kong Open Source エディションの他に、次の機能にアクセスすることができます。

- 管理者ポータル
- 開発者ポータル
- 高度プラグイン
  - Oauth2
  - キャッシング
  - ...

1.2. API 管理とは？

API 管理は、Web アプリケーションプログラミングインターフェース（API）の作成と公開、その使用ポリシーの適用、アクセスの制御、サブスクライバーコミュニティの育成、使用統計の収集と分析、およびパフォーマンスのレポート作成を行うプロセスです。 API 管理コンポーネントは、開発者とサブスクライバーのコミュニティをサポートする仕組みとツールをもたらします。

1.3. IAM ポータル

Kong と IAM は第一に API として設計されています。つまり、Kong/IAM で行われることすべては Rest 呼び出しまたは管理者ポータルで行えるということです。

この記事の中では、すべての例や演習は次のようにして提示されています。
IAM ポータル　　Rest API

1.4. このトレーニングの流れ

この記事では、IAM を IRIS Rest API のプロキシとして使用することを狙いとしています。

この Rest API の定義は、次の場所にあります。

```
http://localhost:52773/swagger-ui/index.html#/
```

または次の場所にあります。

```
https://github.com/grongierisc/iam-training/blob/training/misc/spec.yml
```

この記事は main ブランチから始めてください。

記事の最後に到達すると、training ブランチと同じ結果が得られるはずです。

2. インストール

2.1. インストールには何が必要？

- Git
- Docker（Windows をお使いの方は、Docker インストールで「Linux コンテナ」が使用されるように設定してください）。
- Docker Compose
- Visual Studio Code と InterSystems ObjectScript VSCode 拡張機能
- InterSystems IRIS IAM 対応のライセンスファイル
- IAM Docker イメージ

2.2. IAM と IRIS の連携の仕組み

Kong/IAM の起動時に、コンテナは curl 呼び出しによって、Kong/IAM のライセンスをチェックします。

この呼び出しのエンドポイントは、IRIS コンテナの Rest API です。

参考: Kong ライセンスは IRIS のライセンスに組み込まれています。

2.3. セットアップ

Git clone によって、次のリポジトリのクローンを作成します。

```
git clone https://github.com/grongierisc/iam-training
```

次のようにして、最初の Rest API を実行します。

```
docker-compose up
```

テストを行います。

```
http://localhost:52773/swagger-ui/index.html#/
```

ログイン/パスワード: SuperUser/SYS

2.4. IAM のインストール

2.4.1. Iris イメージ

まず、Community エディションをライセンスエディションに切り替える必要があります。

これを行うには、InterSystems Container Registry へのアクセスをセットアップして、アクセス制限付きの IRIS イメージをダウンロードする必要があります。

開発者コミュニティの「InterSystems Container Registry のご紹介」をご覧ください。

- WRC ログイン情報を使って https://containers.intersystems.com/ にログインし、トークンを取得します。
- コンピュータに Docker ログインをセットアップします。

```
docker login -u="user" -p="token" containers.intersystems.com
```

- InterSystems IRIS イメージを取得します。

```
docker pull containers.intersystems.com/intersystems/irishealth:2020.4.0.524.0
```

2.4.2. IAM イメージ

WRC Software Distribution を使用します。

- コンポーネント > ダウンロードに移動して *IAM-1.5.0.9-4.tar.gz* ファイルをダウンロードし、解凍（unzip & untar）してイメージを読み込みます。

```
docker load -i iam_image.tar
```

2.4.3. Docker ファイルを更新する

IRIS Community エディションをライセンスエディションに変更します。

- containers.intersystems.com/intersystems/irishealth:2020.4.0.524.0
- key フォルダにある iris.key を追加します。

dockerfile を編集して、その上に次の部分を追加します。

```
ARG IMAGE=containers.intersystems.com/intersystems/irishealth:2020.4.0.524.0
# ? 1 ????
FROM $IMAGE as iris-iam
COPY key/iris.key /usr/irissys/mgr/iris.key
COPY iris-iam.script /tmp/iris-iam.script
RUN iris start IRIS \
&& iris session IRIS < /tmp/iris-iam.script \
&& iris stop IRIS quietly

# ? 2 ????
FROM iris-iam
```

この部分では、マルチステージの dockerfile を作成します。

- 第 1 ステージでは、IRIS が IAM ライセンスを提供できるようにします。
- 第 2 ステージは、REST API ビルド用です。

新しい IRIS イメージを構築して IAM エンドポイントとユーザーを有効にする新しい iris-iam.script ファイルを作成します。

```
zn "%SYS"
write "Create web application ...",!
set webName = "/api/iam"
set webProperties("Enabled") = 1
set status = ##class(Security.Applications).Modify(webName, .webProperties)
write:'status $system.Status.DisplayError(status)
write "Web application "_webName_" was updated!",!
```

```
set userProperties("Enabled") = 1
set userName = "IAM"
Do ##class(Security.Users).Modify(userName,.userProperties)
write "User "_userName_" was updated!",!
halt
```

2.4.4. docker-compose を更新する

docker-compose ファイルを次のように更新します。

- db
  - IAM の postgres データベース
- iam-migration
  - データベースのブートストラップ
- iam
  - 実際の IAM インスタンス
- 永続データ用のボリューム

次の部分を docker-compose ファイルの最後に追加します。

```
  iam-migrations:
    image: intersystems/iam:1.5.0.9-4
    command: kong migrations bootstrap up
    depends_on:
      - db
    environment:
      KONG_DATABASE: postgres
      KONG_PG_DATABASE: ${KONG_PG_DATABASE:-iam}
      KONG_PG_HOST: db
      KONG_PG_PASSWORD: ${KONG_PG_PASSWORD:-iam}
      KONG_PG_USER: ${KONG_PG_USER:-iam}
      KONG_CASSANDRA_CONTACT_POINTS: db
      KONG_PLUGINS: bundled,jwt-crafter
      ISC_IRIS_URL: IAM:${IRIS_PASSWORD}@iris:52773/api/iam/license
    restart: on-failure
    links:
      - db:db
  iam:
    image: intersystems/iam:1.5.0.9-4
    depends_on:
      - db
    environment:
      KONG_ADMIN_ACCESS_LOG: /dev/stdout
      KONG_ADMIN_ERROR_LOG: /dev/stderr
      KONG_ADMIN_LISTEN: '0.0.0.0:8001'
      KONG_ANONYMOUS_REPORTS: 'off'
      KONG_CASSANDRA_CONTACT_POINTS: db
      KONG_DATABASE: postgres
      KONG_PG_DATABASE: ${KONG_PG_DATABASE:-iam}
      KONG_PG_HOST: db
      KONG_PG_PASSWORD: ${KONG_PG_PASSWORD:-iam}
      KONG_PG_USER: ${KONG_PG_USER:-iam}
      KONG_PROXY_ACCESS_LOG: /dev/stdout
      KONG_PROXY_ERROR_LOG: /dev/stderr
      KONG_PORTAL: 'on'
```

```
      KONG_PORTAL_GUI_PROTOCOL: http
      KONG_PORTAL_GUI_HOST: '127.0.0.1:8003'
      KONG_ADMIN_GUI_URL: http://localhost:8002
      KONG_PLUGINS: bundled
      ISC_IRIS_URL: IAM:${IRIS_PASSWORD}@iris:52773/api/iam/license
    volumes:
      - ./iam:/iam
    links:
      - db:db
    ports:
      - target: 8000
        published: 8000
        protocol: tcp
      - target: 8001
        published: 8001
        protocol: tcp
      - target: 8002
        published: 8002
        protocol: tcp
      - target: 8003
        published: 8003
        protocol: tcp
      - target: 8004
        published: 8004
        protocol: tcp
      - target: 8443
        published: 8443
        protocol: tcp
      - target: 8444
        published: 8444
        protocol: tcp
      - target: 8445
        published: 8445
        protocol: tcp
    restart: on-failure
  db:
    image: postgres:9.6
    environment:
      POSTGRES_DB: ${KONG_PG_DATABASE:-iam}
      POSTGRES_PASSWORD: ${KONG_PG_PASSWORD:-iam}
      POSTGRES_USER: ${KONG_PG_USER:-iam}
    volumes:
      - 'pgdata:/var/lib/postgresql/data'
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "${KONG_PG_USER:-iam}"]
      interval: 30s
      timeout: 30s
      retries: 3
    restart: on-failure
    stdin_open: true
    tty: true
volumes:
  pgdata:
```

ルートフォルダに .env ファイルを追加します。

```
IRIS_PASSWORD=SYS
```

ちなみに、Kong ポートの定義は次のようになっています。

ポート	プロトコル	説明
:8000	HTTP	コンシューマからの HTTP 着信トラフィックを取り、上流のサービスに転送します。
:8443	HTTPS	コンシューマからの HTTPS 着信トラフィックを取り、上流のサービスに転送します。
:8001	HTTP	管理 API。 HTTP によるコマンドラインからの呼び出しをリスンします。
:8444	HTTPS	管理 API。 HTTPS によるコマンドラインからの呼び出しをリスンします。
:8002	HTTP	Kong Manager（GUI）。 HTTP トラフィックをリスンします。
:8445	HTTPS	Kong Manager（GUI）。 HTTPS トラフィックをリスンします。
:8003	HTTP	開発者ポータル。 開発者ポータルが有効になっていることを前提に、HTTP トラフィックをリスンします。
:8446	HTTPS	開発者ポータル。 開発者ポータルが有効になっていることを前提に、HTTPS トラフィックをリスンします。
:8004	HTTP	HTTP による開発者ポータルの /files トラフィック。開発者ポータルが有効になっていることが前提です。
:8447	HTTPS	HTTPS による開発者ポータルの /files トラフィック。開発者ポータルが有効になっていることが前提です。

2.4.5. オプション: .env として IRISPASSWORD を追加する

使いやすいよう（またセキュリティの理由により）、IRIS dockerfile の .env ファイルを使用できます。

使用するには、docker-compose の iris サービスの部分を次のように編集します。

```
    build:
      context: .
      dockerfile: dockerfile
      args:
        - IRIS_PASSWORD=${IRIS_PASSWORD}
```

そして、dockerfile を編集します（ビルドの第 2 または第 1 ステージ）。

```
ARG IRIS_PASSWORD
RUN echo "${IRIS_PASSWORD}" > /tmp/password.txt && /usr/irissys/dev/Container/changeP
assword.sh /tmp/password.txt
```

2.4.6. テスト

```
docker-compose -f "docker-compose.yml" up -d --build
```

3. その 1: サービス/ルート

Kong/IAM の連携の仕組みを覚えていますか？

ここでは、次の項目を構築します。

- サービス
  - crud API 用
- ルート
  - このサービスにアクセスするためのルート

3.1. サービスを作成する

IAM ポータル

Rest API

```
# ?????????curl -i -X POST \--url http://localhost:8001/services/ \--data 'name=crud' \--data 'url=http://iris:52773/crud/'
```

ここで確認したこと: サービスの作成するには、その URL のみが必要です。

3.2. ルートを作成する

IAM ポータル

Rest API

```
# ????????curl -i -X POST \--url http://localhost:8001/services/crud/routes \--data 'name=crud-route' \--data 'paths=/persons/*' \--data 'strip_path=false'
```

ルートの作成には、次の項目が必要です。

- サービス名
- RegEx が許可されているパス

3.3. テスト

元の API

```
# ????curl –i --location --request GET 'http://localhost:52773/crud/persons/all' \--header 'Authorization: Basic U3VwZXJVc2VyOlNZUw=='
```

プロキシ API

```
# KONGcurl –i --location --request GET 'http://localhost:8000/persons/all' \--header 'Authorization: Basic U3VwZXJVc2VyOlNZUw==='
```

ここで確認したこと:

- レガシー側は何も変わらない
- Kong 側:
  - ポートを変更する
  - パスはルートに対応する
  - 依然として認証が必要

4. その 2: プラグインの使用

次に、IRIS エンドポイントに対して Kong の自動認証を行ってみましょう。

これを行うために、resquest-transformer というプラグインを使用します。

4.1. プラグインをサービスに追加する

IAM ポータル

Rest API

```
# ??????????curl -i -X POST \--url http://localhost:8001/services/crud/plugins \--data 'name=request-transformer' \--data 'config.add.headers=Authorization:Basic U3VwZXJVc2VyOlNZUw==' \--data 'config.replace.headers=Authorization:Basic U3VwZXJVc2VyOlNZUw==='
```

4.2. テスト

元の API

```
# ????curl –i --location --request GET 'http://localhost:52773/crud/persons/all'
```

プロキシ API

```
# KONGcurl –i --location --request GET 'http://localhost:8000/persons/all'
```

ここで確認したこと:

- 元の API にエラー 401 が発生する
- 認証を行わずにデータに到達した

5. その 3: 独自認証の追加

ここでは、元の API を損なうことなく、独自の認証を追加してみましょう。

5.1. コンシューマを追加する

IAM ポータル

Rest API

```
# ?????? anonymous ?????curl -i -X POST \--url http://localhost:8001/consumers/ \--data "username=anonymous" \--data "custom_id=anonymous"
# ?????? user ?????curl -i -X POST \--url http://localhost:8001/consumers/ \--data "username=user" \--data "custom_id=user"
```

5.2. Basic 認証プラグインを追加する

IAM ポータル

Rest API

```
# ????? Basic ????????curl -i -X POST http://localhost:8001/routes/crud-route/plugins \--data "name=basic-auth" \--data "config.anonymous=5cc8dee4-066d-492e-b2f8-bd77eb0a4c86" \--data "config.hide_credentials=false"
```

説明:

- config.anonymous = anonymous コンシューマの uuid

5.3. ACL プラグインを追加する

IAM ポータル

Rest API

```
# ACL ??????curl -i -X POST http://localhost:8001/routes/crud-route/plugins \--data "name=acl" \--data "config.whitelist=user"
```

5.4. ACL とログイン情報で USER を構成する

IAM ポータル

Rest API

```
# ??????????????curl -i -X POST \--url http://localhost:8001/consumers/user/acls \--data "group=user"# ?????????????????curl -i -X POST http://localhost:8001/consumers/user/basic-auth \--data "username=user" \--data "password=user"
```

5.5. テスト

元の API

```
# ????curl –i --location --request GET 'http://localhost:52773/crud/persons/all' \--header 'Authorization:Basic dXNlcjp1c2Vy'
```

プロキシ API

```
# KONGcurl –i --location --request GET 'http://localhost:8000/persons/all' \--header 'Authorization:Basic dXNlcjp1c2Vy'
```

6. 演習: レート制限

1. 認証されていないユーザーを有効にする
2. 認証されていないユーザーのレートを 1 分あたり 2 回の呼び出しに制限する

6.1. ソリューション

1. 認証されていないユーザーを有効にする

IAM ポータル

Rest API

```
# ??????????????curl -i -X POST \--url http://localhost:8001/consumers/anonymous/acls \--data "group=user"
```

2. 認証されていないユーザーのレートを 1 分あたり 2 回の呼び出しに制限する

IAM ポータル

Rest API

```
# ?????????????????curl -i -X POST \--url http://localhost:8001/consumers/anonymous/plugins \--data "name=rate-limiting" \--data "config.limit_by=consumer" \--data "config.minute=2"
```

7. 開発者ポータル

7.1. 概要

Kong Developer Portal（Kong 開発者ポータル）では以下が提供されます。

- すべての開発者が信頼できる唯一の情報源
- 直感的なドキュメントのコンテンツ管理
- 合理化された開発者オンボーディング
- ロールベースのアクセス制御（RBAC）

7.2. 有効化

IAM ポータル

Rest API
```
curl -X PATCH http://localhost:8001/worksp
aces/default --data "config.portal=true"
```

7.3. 最初の仕様を追加する

IAM ポータル

Rest API
```
curl -X POST http://localhost:8001/default
/files -F "path=specs/iam-training.yml" -F
 "contents=@misc/spec.yml"
```

7.4. テスト

```
http://localhost:8003/default/documentation/iam-training
```

何が起こりましたか？

これを解決するには？

7.5. 演習

1. CORS プラグインをルートに追加する

7.5.1. ソリューション

IAM ポータル

Rest API
```
# CORS ??????curl -i -X POST http://localh
ost:8001/routes/crud-
route/plugins \--data "name=cors"
```

8. 開発者ポータル（その 2）: 認証

8.1. Basic 認証を有効にする

IAM ポータル

セッション構成（JSON）
```
{    "cookie_secure": false,    "cookie_na
me": "portal_session",    "secret": "SYS",
    "storage": "kong"}
```

これで、開発者ポータルの認証が有効になりました。

8.2. アクセスを制限する

デフォルトでは、認証されていないユーザーがすべてにアクセスできるようになっています。

ロールを作成することで、アクセスを制限することができます。

例として、CRUD API ドキュメントへのアクセスを制限してみましょう。

8.2.1. ロールを作成する

IAM ポータル

Rest API

```
# ?????????curl -i -X POST http://localhos
t:8001/default/developers/roles \--data "n
ame=dev"
```

8.2.2. 仕様にロールを追加する

IAM ポータル

Rest API

```
# ?????????curl 'http://localhost:8001/def
ault/files/specs/iam-training.yml' -X PATC
H -H 'Accept: application/json, text/plain
, */*'  --compressed -H 'Content-Type: app
lication/json;charset=utf-8' -H 'Origin: h
ttp://localhost:8002' -H 'Referer: http://
localhost:8002/default/portal/permissions/
' --data-raw $'{"contents":"x-headmatter:\
\n  readable_by:\\n    - dev\\nswagger: \'
2.0\'\\ninfo:\\n  title: InterSystems IRIS
 REST CRUD demo\\n  description: Demo of a
 simple rest API on IRIS\\n  version: \'0.
1\'\\n  contact:\\n    email: apiteam@swag
ger.io\\n  license:\\n    name: Apache 2.0
\\n    url: \'http://www.apache.org/licens
es/LICENSE-2.0.html\'\\nhost: \'localhost:
8000\'\\nbasePath: /\\nschemes:\\n  - http
\\nsecurityDefinitions:\\n  basicAuth:\\n
   type: basic\\nsecurity:\\n  - basicAuth
: []\\npaths:\\n  /:\\n    get:\\n      de
scription: \' PersonsREST general informat
ion \'\\n      summary: \' Server Info \'\
\n      operationId: GetInfo\\n      x-ISC
_CORS: true\\n      x-ISC_ServiceMethod: G
etInfo\\n      responses:\\n        \'200\
':\\n          description: (Expected Resu
lt)\\n          schema:\\n            type
: object\\n            properties:\\n
         version:\\n                type:
string\\n        default:\\n          desc
ription: (Unexpected Error)\\n  /persons/a
ll:\\n    get:\\n      description: \' Ret
reive all the records of Sample.Person \'\
\n      summary: \' Get all records of Per
son class \'\\n      operationId: GetAllPe
rsons\\n      x-ISC_ServiceMethod: GetAllP
ersons\\n      responses:\\n        \'200\
':\\n          description: (Expected Resu
lt)\\n          schema:\\n            type
: array\\n            items:\\n
   $ref: \'#/definitions/Person\'\\n
  default:\\n          description: (Unexp
ected Error)\\n  \'/persons/{id}\':\\n
get:\\n      description: \' Return one re
cord fo Sample.Person \'\\n      summary:
```

```
\' GET method to return JSON for a given p
erson id\'\\n      operationId: GetPerson\
\n      x-ISC_ServiceMethod: GetPerson\\n
     parameters:\\n        - name: id\\n
        in: path\\n          required: tru
e\\n          type: string\\n      respons
es:\\n        \'200\':\\n          descrip
tion: (Expected Result)\\n          schema
:\\n            $ref: \'#/definitions/Pers
on\'\\n        default:\\n          descri
ption: (Unexpected Error)\\n    put:\\n
   description: \' Update a record in Samp
le.Person with id \'\\n      summary: \' U
pdate a person with id\'\\n      operation
Id: UpdatePerson\\n      x-ISC_ServiceMeth
od: UpdatePerson\\n      parameters:\\n
     - name: id\\n          in: path\\n
       required: true\\n          type: st
ring\\n        - name: payloadBody\\n
     in: body\\n          description: Req
uest body contents\\n          required: f
alse\\n          schema:\\n            typ
e: string\\n      responses:\\n        \'2
00\':\\n          description: (Expected R
esult)\\n        default:\\n          desc
ription: (Unexpected Error)\\n    delete:\
\n      description: \' Delete a record wi
th id in Sample.Person \'\\n      summary:
 \' Delete a person with id\'\\n      oper
ationId: DeletePerson\\n      x-ISC_Servic
eMethod: DeletePerson\\n      parameters:\
\n        - name: id\\n          in: path\
\n          required: true\\n          typ
e: string\\n      responses:\\n        \'2
00\':\\n          description: (Expected R
esult)\\n        default:\\n          desc
ription: (Unexpected Error)\\n  /persons/:
\\n    post:\\n      description: \' Creat
es a new Sample.Person record \'\\n      s
ummary: \' Create a person\'\\n      opera
tionId: CreatePerson\\n      x-ISC_Service
Method: CreatePerson\\n      parameters:\\
n        - name: payloadBody\\n          i
n: body\\n          description: Request b
ody contents\\n          required: false\\
n          schema:\\n            type: str
ing\\n      responses:\\n        \'200\':\
\n          description: (Expected Result)
\\n        default:\\n          descriptio
n: (Unexpected Error)\\ndefinitions:\\n  P
erson:\\n    type: object\\n    properties
:\\n      Name:\\n        type: string\\n
     Title:\\n        type: string\\n
 Company:\\n        type: string\\n      P
hone:\\n        type: string\\n      DOB:\
\n        type: string\\n        format: d
ate-time\\n"}'
```

ここでは、次の箇所が重要です。

```
x-headmatter:
  readable_by:
    - dev
```

次のドキュメントを参照してください: 「readableby attribute 」（readableby 属性）

8.2.3. テスト

8.2.3.1. 新しい開発者を登録する

8.2.3.2. この開発者を承認する

8.2.3.3. この開発者のロールを追加する

```
curl 'http://localhost:8001/default/developers/dev@dev.com' -X PATCH --compressed -H
'Content-Type: application/json;charset=utf-8' -H 'Cache-Control: no-cache' -H 'Origi
n: http://localhost:8002' -H 'DNT: 1' -H 'Connection: keep-alive' -H 'Referer: http:/
/localhost:8002/default/portal/permissions/dev/update' -H 'Pragma: no-cache' --data-
raw '{"roles":["dev"]}'
```

8.3. 開発者向け Oauth2 を追加する

ここでは、開発者が安全に crud API を使用できるよう、Oauth2 認証を追加します。

このフローでは、開発者自身が登録を行い、crud API へのアクセス権を付与します。

8.3.1. その 1: Basic 認証を削除する

これを行うには、Basic 認証を BearToken 認証に置き換えます。

まず、Basic 認証/ACL を無効にします。

IAM ポータル

Rest API

```
# ACL ??????????curl 'http://localhost:80
01/default/routes/afefe836-b9be-49a8-927a-
1324a8597a9c/plugins/3f2e605e-9cb6-454a-83
ec-d1b1929b1d30' -X PATCH --compressed -H
'Content-Type: application/json;charset=ut
f-8' -H 'Cache-Control: no-cache' -H 'Orig
in: http://localhost:8002' -H 'DNT: 1' -H
'Connection: keep-alive' -H 'Referer: http
://localhost:8002/default/plugins/acl/3f2e
605e-9cb6-454a-83ec-d1b1929b1d30/update' -
H 'Pragma: no-cache' --data-raw '{"enabled
":false,"name":"acl","route":{"id":"afefe8
36-b9be-49a8-927a-1324a8597a9c"},"config":
{"hide_groups_header":false,"whitelist":["
user","dev","crud"]}}'
```

8.3.2. その 2: application-registration プラグインを追加する

IAM ポータル

Rest API

```
# application-registration ?????????curl
```

```
-i -X POST \--url http://localhost:8001/se
rvices/crud/plugins \--data 'name=applicat
ion-registration' \--data 'config.auth_hea
der_name=authorization' \--data 'config.au
to_approve=true' \--data 'config.display_n
ame=auth' \--data 'config.enable_client_cr
edentials=true'
```

8.3.3. リンクサービスとドキュメント

IAM ポータル

Rest API

```
curl 'http://localhost:8001/default/docume
nt_objects' --compressed -H 'Content-Type:
 application/json;charset=utf-8' -H 'Cache
-Control: no-cache' -H 'Origin: http://loc
alhost:8002' -H 'DNT: 1' -H 'Connection: k
eep-alive' -H 'Referer: http://localhost:8
002/default/services/create-documentation'
 -H 'Pragma: no-cache' --data-raw '{"servi
ce":{"id":"7bcef2e6-117c-487a-aab2-c7e57a0
bf61a"},"path":"specs/iam-training.yml"}'
```

8.3.3.1. テスト

dev@dev.com としてログインした開発者ポータルで、新しいアプリケーションを作成します。

clientid と clientsecret を提供されます。

これらの情報は、Swagger 開発者ポータルで使用できます。

このアプリケーションを crud サービスに登録します。

トークンを取得する:

```
curl --insecure -X POST https://localhost:8443/persons/oauth2/token \
  --data "grant_type=client_credentials" \
  --data "client_id=2TXNvDqjeVMHydJbjv9t96lWTXOKAtU8" \
  --data "client_secret=V6Vma6AtIvl04UYssz6gAxPc92eCF4KR"
```

トークンを使用する:

```
curl --insecure -X GET https://localhost:8443/persons/all \
  --header "authorization: Bearer u5guWaYR3BjZ1KdwuBSC6C7udCYxj5vK"
```

9. セキュアな管理ポータル

9.1. 管理者を作成する

シードパスワードなしでKong をブートしました。

そのため、RBAC を適用する前に、管理者を作成する必要があります。

これを行うために、次を行ってください。

- ［チーム］に移動します。
- 管理者を招待します。
  - メールを設定します。
  - ユーザー名を設定します。
  - ロールをスーパー管理者に設定します。
  - 招待します。
- ［招待された管理者］に移動します。
- 表示します。
- リンクを生成します。

9.2. Kong Manager の Basic 認証を有効にする

この機能を有効にするには、docker-compose ファイルに変更を加える必要があります。

次のコードを iam サービスの環境に追加します。

```
      KONG_ENFORCE_RBAC: 'on'
      KONG_ADMIN_GUI_AUTH: 'basic-auth'
      KONG_ADMIN_GUI_SESSION_CONF: '{"secret":"${IRIS_PASSWORD}","storage":"kong","co
okie_secure":false}'
```

コンテナを再起動します。

```
docker-compose down && docker-compose up -d
```

招待された管理者のリンクに移動します。

```
http://localhost:8002/register?email=test.test%40gmail.com&username=admin&token=eyJhb
GciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2MTYzMzYzNzEsImlkIjoiY2JiZGE5Y2UtODQ3NS00M
mM2LTk4ZjItNDgwZTI4MjQ4NWNkIn0.sFeOc_5UPIr3MdlQrgyGvmvIjRFvSn3nQjo2ph8GrJA
```

9.3. RBAC で Kong Admin API を使用する

RBAC が設定されたため、Kong Admin API は使用できません。

```
curl -s -X GET \
  --url http://localhost:8001/routes
```

このエラーが発生します。

```
{"message":"Invalid credentials. Token or User credentials required"}
```

9.3.1. トークンで管理者ユーザーを作成する

- ［チーム］に移動します。
- RBAC ユーザー
- 新しいユーザーを追加

```
curl -s -X GET \
  --url http://localhost:8001/routes \
  --header "Kong-Admin-Token: SYS"
```

10. プラグイン

Kong には品質の高いプラグインが付属しています。

しかし、組み込まれていないプラグインが必要となった場合は、どうすればよいのでしょうか。
コミュニティプラグインを使用する場合はどうすればよいのでしょうか。

この章では、コミュニティプラグインとそのインポート方法について説明します。

その後で、独自のプラグインの構築方法を確認しましょう。

10.1. コミュニティプラグインをインポートする

このステップでは、jwt-crafter プラグインを使用することにします。

これは、Kong 自体で JWT
トークンを生成できるようにするプラグインで、トーク生成を行う上流のサービスに頼る必要がなくなります。

このプラグインは次の場所にあります。

```
https://github.com/grongierisc/kong-plugin-jwt-crafter
```

docker バージョンを使用しているため、このプラグインをインストールするには、プラグインを組み込む新しいイメージを構築する必要があります。

10.1.1. コミュニティプラグインで新しい Kong/IAM docker イメージを構築する

1. この Git のルートに「iam」というフォルダを作成します。
2. この新しいフォルダに dockerfile を作成します。
3. 「plugins」というフォルダを作成します。
    1. すべてのコミュニティプラグインを追加する場所です。
4. docker-compose ファイルを更新して、新しいプラグインを有効にします。

plugins フォルダで、コミュニティプラグインの git clone を実行します。

```
git clone https://github.com/grongierisc/kong-plugin-jwt-crafter
```

dockerfile は次のようになります。

```
FROM intersystems/iam:1.5.0.9-4

USER root
```

```
COPY ./plugins /custom/plugins

RUN cd /custom/plugins/kong-plugin-jwt-crafter && luarocks make

USER kong
```

この dockerfile で何を確認できますか？

コミュニティプラグインをインストールするには、ルートフォルダ（rockspec のある場所）に移動して、luarocks make を呼び出しています。 それだけです。 これでプラグインをインストールできました。

docker-compose の方では次を行います。

1. iam イメージタグを編集します。
    1. intersystems/iam:1.5.0.9-4 -> intersystems/iam-custom:1.5.0.9-4
2. ビルドコンテキストを追加します。

```
build:
  context: iam
  dockerfile: dockerfile
```

3. 環境変数でプラグインを有効にします。

```
KONG_PLUGINS: 'bundled,jwt-crafter'
```

これで新しい IAM イメージをビルドします。

```
docker-compose build iam
```

10.1.2. テスト

```
docker-compose up -d
```

［プラグイン］->［新規］に移動すると、リストの最後に jwt-crafter プラグインが表示されているはずです。

10.1.2.1. 使ってみる

1. 新しいサービスを作成します。

IAM ポータル

Rest API

```
# ?????????curl -i -X POST \--url http://l
ocalhost:8001/services/ \--data 'name=crud
-persons' \--data 'url=http://iris:52773/c
rud/persons/'
```

2. ルートを作成します。

IAM ポータル

Rest API

```
# ????????curl -i -X POST \--url http://lo
```

```
calhost:8001/services/crud-persons/routes
\--data 'name=crud-route-jwt' \--data 'pat
hs=/crud/persons/*' \--data 'strip_path=tr
ue'
```

3. 自動認証を再利用します。

IAM ポータル

Rest API

```
# ??????????curl -i -X POST \--url http://
localhost:8001/services/crud-persons/plugi
ns \--data 'name=request-transformer' \--d
ata 'config.add.headers=Authorization:Basi
c U3VwZXJVc2VyOlNZUw==' \--data 'config.re
place.headers=Authorization:Basic U3VwZXJV
c2VyOlNZUw=='
```

これで完了です。 jwt-crafter の実際の使用は次のようになります。

```
# acl ?????????
curl -i -X POST http://localhost:8001/routes/crud-route-jwt/plugins \
    --data "name=acl"  \
    --data "config.whitelist=test" \
    --data "config.hide_groups_header=false"

# ?????????
curl -i -X POST \
  --url http://localhost:8001/services/ \
  --data 'name=jwt-login' \
  --data 'url=http://neverinvoked/'

# ????????
curl -i -X POST \
  --url http://localhost:8001/services/jwt-login/routes \
  --data 'name=jwt-login-route' \
  --data 'paths=/jwt/log-in'

# ????? Basic ????????
curl -i -X POST http://localhost:8001/routes/jwt-login-route/plugins \
    --data "name=basic-auth"  \
    --data "config.hide_credentials=false"

# ????? Basic ????????
curl -i -X POST http://localhost:8001/routes/jwt-login-route/plugins \
    --data "name=jwt-crafter"  \
    --data "config.expires_in=86400"

# ??????????
curl -i -X POST \
   --url http://localhost:8001/consumers/ \
   --data "username=test"

# ??????????????
curl -i -X POST \
   --url http://localhost:8001/consumers/test/acls \
   --data "group=test"
```

```
# ??????????????????
curl -i -X POST http://localhost:8001/consumers/test/basic-auth \
    --data "username=test" \
    --data "password=test"
curl -i -X POST http://localhost:8001/consumers/test/jwt \
    --data "key=test" \
    --data "algorithm=HS256"

# JWT ?????
curl -i -X POST http://localhost:8001/routes/crud-route-jwt/plugins \
    --data "name=jwt"
```

テストしよう！

```
# test:test ? base64 ?????????????
curl -H 'Authorization: basic dGVzdDp0ZXN0' localhost:8000/jwt/log-in
```

```
curl --location --request GET 'http://localhost:8000/crud/persons/all' \
--header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW0iOiJ0ZXN0
Iiwic3ViIjoiODJiNjcwZDgtNmY2OC00NDE5LWJiMmMtMmYxZjMxNTViN2E2Iiwicm9sIjpbInRlc3QiXSwiZ
XhwIjoxNjE2MjUyMTIwLCJpc3MiOiJ0ZXN0In0.g2jFqe0hDPumy8_gG7J3nYsuZ8KUz9SgZOecdBDhfns'
```

10.2. 新しいプラグインを作成する

これは lua を学習する場所ではありませんが、IAM
を素早く再起動して新しい開発をテストする方法などのヒントをいくつか紹介します。

10.2.1. ファイル構造

```
kong-plugin-helloworld
??? kong
?   ??? plugins
?       ??? helloworld
?           ??? handler.lua
?           ??? schema.lua
??? kong-plugin-helloworld-0.1.0-1.rockspec
```

表記法により、Kong プラグインには kong-plugin のプレフィックスを付ける必要があります。

この例では、プラグインの名前を helloworld とします。

次の 3 つのファイルは必須ファイルです。

- handler.lua: プラグインのコアです。 実装のインターフェースで、各関数は、リクエスト/接続のライフサイクルにおいて希望する時点で実行されます。
- schema.lua: プラグインはおそらく、ユーザーが入力した構成を維持する必要があります。 このモジュールは、その構成のスキーマを保持し、ユーザーが有効な構成値のみを入力できるようにルールを定義します。
- *.rockspec: Rockspec: 仕様ファイルのパッケージです。宣言的な Lua スクリプトで、ロック *.rockspec（テーブルが含まれる Lua ファイル）を構築してパッケージ化する方法に関するルールが含まれます。

10.2.1.1. handler.lua

plugins インターフェースでは、handler.lua ファイル内の次のいずれかのメソッドをオーバーライドして、Kong の実行ライフサイクルのさまざまなエントリポイントにカスタムロジックを実装できます。

関数名	フェーズ	説明
:initworker()	initworker	Nginx ワーカープロセスが起動するたびに実行されます。
:certificate()	sslcertificate	SSL ハンドシェイクのSSL 証明書の配信フェーズで実行されます。
:rewrite()	rewrite	rewrite フェーズハンドラーとしてクライアントからリクエストを受信するたびに、そのリクエストに対して実行されます。 注意: このフェーズでは、サービスもコンシューマも識別されていないため、このハンドラーはプラグインがグローバルプラグインとして構成されている場合にのみ実行されます。
:access()	access	クライアントからのすべてのリクエストに対して、上流サービスにプロキシされる前に実行されます。
:response()	access	headerfilter() と bodyfilter() の両方を置き換えます。 上流サービスからレスポンス全体を受信した後、その一部をクライアントに送信する前に実行されます。
:headerfilter()	headerfilter	上流サービスからすべてのレスポンスヘッダーのバイトが受信されると実行されます。
:bodyfilter()	bodyfilter	上流サービスからレスポンス本文のチャンクを受信するごとに実行されます。 レスポンスはクライアントにストリーミングされるため、バッファーサイズを超過し、チャンクごとにストリーミングされる場合があります。 そのため、レスポンスが大きければ、このメソッドが何度も呼び出されることがあります。 詳細は、lua-nginx-module のドキュメントをご覧ください。
:log()	log	レスポンスの最後のバイトがクライアントに送信されると実行されます。

10.2.1.1.1. 例

```
local BasePlugin = require "kong.plugins.base_plugin"

local HelloWorldHandler = BasePlugin:extend()

function HelloWorldHandler:new()
  HelloWorldHandler.super.new(self, "helloworld")
end

function HelloWorldHandler:access(conf)
  HelloWorldHandler.super.access(self)
  if conf.say_hello then
    ngx.log(ngx.ERR, "============ Hello World! ============")
    ngx.header["Hello-World"] = "Hello World!!!"
  else
    ngx.log(ngx.ERR, "============ Bye World! ============")
```

```
    ngx.header["Hello-World"] = "Bye World!!!"
  end
end

return HelloWorldHandler
```

10.2.1.2. schema.lua

ポータルで表示される構成ファイルです。

```
return {
    no_consumer = true,
    fields = {
      say_hello = { type = "boolean", default = true },
      say_hello_body = { type = "boolean", default = true }
    }
  }
```

10.2.1.3. *.rockspec

```
package = "kong-plugin-
helloworld"  -- hint: rename, must match the info in the filename of this rockspec!
                                    -- as a convention; stick to the prefix: `kong-
plugin-`
version = "0.1.0-1"                 -- hint: renumber, must match the info in the filen
ame of this rockspec!
-- The version '0.1.0' is the source code version, the trailing '1' is the version of
 this rockspec.
-- whenever the source version changes, the rockspec should be reset to 1. The rocksp
ec version is only
-- updated (incremented) when this file changes, but the source remains the same.

-- TODO: This is the name to set in the Kong configuration `plugins` setting.
-- Here we extract it from the package name.
local pluginName = package:match("^kong%-plugin%-(.+)$")  -- "myPlugin"

supported_platforms = {"linux", "macosx"}
source = {
  url = "https://github.com/grongierisc/iam-training",
  branch = "master",
--  tag = "0.1.0"
-- hint: "tag" could be used to match tag in the repository
}

description = {
  summary = "This a demo helloworld for Kong plugin",
  homepage = "https://github.com/grongierisc/iam-training",
  license = "Apache 2.0"
}

dependencies = {
   "lua >= 5.1"
   -- other dependencies should appear here
}

build = {
```

```
  type = "builtin",
  modules = {
    ["kong.plugins."..pluginName..".handler"] = "kong/plugins/"..pluginName.."/handle
r.lua",
    ["kong.plugins."..pluginName..".schema"] = "kong/plugins/"..pluginName.."/schema.
lua",
  }
}
```

10.2.2. 構築

ここでは、次の章と同じことを行います: 11.1.1. コミュニティプラグインで新しい Kong/IAM docker イメージを構築する

ただし、このプラグインに適応させています。

Dockerfile :

```
FROM intersystems/iam:1.5.0.9-4

USER root
COPY ./plugins /custom/plugins

RUN cd /custom/plugins/kong-plugin-jwt-crafter && luarocks make
RUN cd /custom/plugins/kong-plugin-helloworld && luarocks make

#USER kong #Stay with root use, we will see why later
```

環境変数でプラグインを有効にします。

```
KONG_PLUGINS: 'bundled,jwt-crafter,helloworld'
```

これで新しい IAM イメージをビルドします。

```
docker-compose build iam
```

そして docker-compose を起動してテストします。

10.2.3. ヒント

IAM コンテナを「デバッグモード」で実行し、コンテナの停止/再開を簡単に行えるようにするには、dockerfile を変更して、プラグインの追加/削除など行います。

iam サービスを次のようにして停止できます。

```
docker-compose stop iam
```

そして、シェルを使用して実行モードで起動します。

```
docker-compose run -p 8000:8000 -p 8001:8001 -p 8002:8002 iam sh
```

コンテナでは次のように行います。

```
./docker-entrypoint.sh kong
```

コーディングをお楽しみください :)

11. CI/CD

この記事も残り少なくなりました。

最後に、DevOps/CI/CD について話しましょう。 この章は、IAM/Kong 向けの CI/CD を実装/スクリプト化する方法について、いくつかのアイデアを提供することを目的としています。

Kong は第一に API であるため、すべての Rest 呼び出しをスクリプト化して、各環境で実行させるという考え方です。

postman とその親友の newman（postman のコマンドラインバージョン）を使用すると、Rest 呼び出しのスクリプト化を最も簡単に行うことができます。

11.1. postman コレクションを作成する

postman が便利な理由の 1 つは、Rest 呼び出しの前後でスクリプトを実行できるところにあります。

ほとんどの場合、この機能を使用します。

11.1.1. IAM が起動しているか

最初のスクリプトでは、IAM が稼働しているかどうかを確認します。

```
var iam_url = pm.environment.get("iam_url");
var iam_config_port = pm.environment.get("iam_config_port");
var url = "http://" + iam_url + ":" + iam_config_port + "/";
SenReq(20);
async function SenReq(maxRequest) {
    var next_request = "end request";
    const result = await SendRequest(maxRequest);
    console.log("result:",result);
    if(result == -1)
    {
        console.error("IAM starting .... failed !!!!");
    }
}

function SendRequest(maxRequest) {
    return new Promise(resolve => {
        pm.sendRequest(url,
            function (err) {
                if (err) {
                    if (maxRequest > 1) {
                        setTimeout(function () {}, 5000);
```

```
                        console.warn("IAM not started...retry..next retry in 5 sec");
                        SendRequest(maxRequest - 1);
                    } else {
                        console.error("IAM starting .... failed");
                        resolve(-1);
                    }
                } else {
                    console.log("IAM starting .... ok");
                    resolve(1);
                }

            }
        );
    });
}
```

11.1.2. 古いデータを削除する

```
var iam_url=pm.environment.get("iam_url");
var iam_config_port=pm.environment.get("iam_config_port");

pm.sendRequest("http://"+iam_url+":"+iam_config_port+"/plugins", function (err, res)
{
    if (err) {
        console.log("ERROR : ",err);
    }
    else {

        var body_json=res.json();
        if(body_json.data)
        {
            for( i=0; i < body_json.data.length; i++)
            {
                // Example with a full fledged SDK Request
                route_id = body_json.data[i].id;
                const delete_route = {
                  url: "http://"+iam_url+":"+iam_config_port+"/plugins/" + route_id,
                  method: 'DELETE',
                };
                pm.sendRequest(delete_route, function(err, res){
                    console.log(err ? err : res);
                });
            }
        }
    }
});
```

ルート、サービス、およびコンシューマでも同じように行ってください。

ルートのあるサービスは削除できないため、この実行順序は重要です。

11.1.3. サービス/ルートを作成する

ルートはサービスに依存しています。 この種のケースでは、postman の Test
関数を使用してデータを取得することができます。

画面

スクリプト

```
var id = pm.response.json().id;var name =
pm.response.json().name;pm.globals.set("se
rvice_crud_id", id);pm.globals.set("servic
e_crud_name", name);
```

ここで、レスポンスから、新しいサービスの ID と名前を保存します。

次に、次のルートの作成でそれらを使用します。

画面

スクリプト

```
service_crud_name = pm.globals.get("servic
e_crud_name");
```

ここで、グローバル変数「servicecrudname」を取得します。

次に、それを実際の呼び出しで使用します。

画面

スクリプト

```
{    "paths": [        "/persons/*"    ],
   "protocols": [        "http"    ],    "
name": "crud-persons",    "strip_path": fa
lse,    "service": {        "name": "{{ser
vice_crud_name}}"        }}
```

11.1.3.1. ヒント

- ペイロードは json または form-data のいずれかです。
  - form-data:
- json:

json 形式を簡単に取得するには、管理者ポータルに移動し、json を表示してコピーします。

11.2. newman で実行する

```
docker run  --rm -v "`pwd`/ci/":"/etc/newman" \
 --network="iam-training_default" \
 -t postman/newman run "DevOps_IAM.postman_collection.json" \
 --environment="DevOps_IAM.postman_environment.json"
```

#InterSystems IRIS
InterSystems Open Exchangeで関連アプリケーションを確認してください

ソースURL:
https://jp.community.intersystems.com/post/%E3%82%BC%E3%83%AD%E3%81%8B%E3%82%89%E4%BD%BF%E3%81%84%E3%81%93%E3%81%AA%E3%81%99-iam%EF%BC%88intersystems-api-manager%EF%BC%89