Configurando um API Requester em OAS 2 - Parte 1

Passo a passo - Gabriel Paiva e Pedro Alonso

IBM

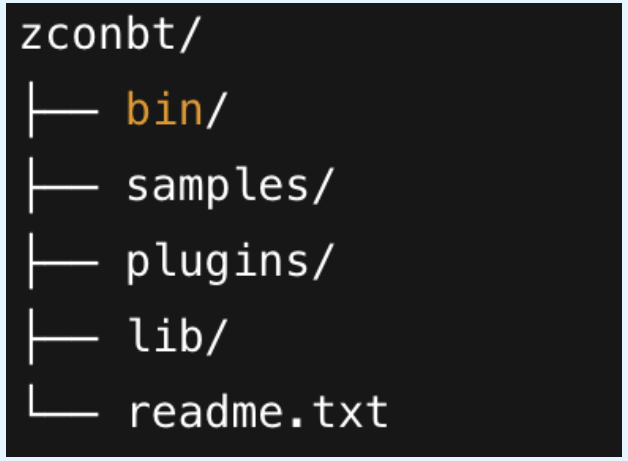
Objetivos

Utilizando uma API Watson como **exemplo**, nesse passo a passo temos como objetivo:

- **Aprender** a configurar o zconbt, utilitário necessário para gerar os artefatos do requester.
- E **gerar os artefatos** propriamente ditos.
- **Deploy** do requester no servidor.

Instalando o z/OS Connect Build Toolkit.

Extraindo os arquivos do zconbt.zip



O Build Toolkit pode rodar tanto no z/OS como na sua estação de trabalho local. Ele funciona em qualquer ambiente contanto que o Java 8 esteja disponível.

1 – Primeiro precisamos localizar o arquivo “zconbt.zip”. Esse arquivo está disponível no diretório de instalação do z/OS Connect.

Dica: Para achar o arquivo .zip, o comando de buscar recursivamente (em todas as subpastas) a partir da pasta atual pode ser útil.

```
find . -name "zconbt.zip"
```

2 – Depois crie uma pasta própria para o Build Toolkit

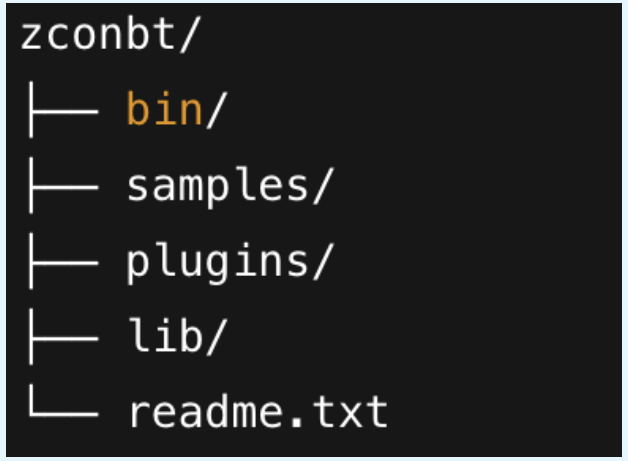
3 – Copie o .zip para a pasta criada.

4 – E extraia o .zip nessa pasta.

O comando para extração é:

```
jar -xf zconbt.zip
```

Testando o funcionamento do Build Toolkit



1 – Primeiro vamos garantir que temos permissão para executar esse comando. No z/OS ou sistemas Unix-like execute:

```
chmod +x bin/zconbt*
```

2 – Agora entre no diretório /bin.

3 – E execute um comando de ajuda para testar se o comando zconbt está funcionando, se estiver o Built Toolkit está funcionando.

No Windows execute:

```
zconbt.bat --help
```

No z/OS execute:

```
./zconbt.zos --help
```

Em sistemas Unix-like:

```
./zconbt --help
```

Adicionando o zconbt aos utilitários padrão do OMVS (Recomendado)

```
000111 PATH=/bin
000112 PATH=$PATH:.
000113 PATH=$PATH:/usr/sbin
000114 PATH=$PATH:/usr/lpp/java/J8.0_64/bin
000115 PATH=$PATH:/usr/bin
000116 PATH=$PATH:/zfsprd/nodejs/node-v14.16.0-os390-s390x-202103142315/bin
000117 PATH=$PATH:/u/pedroa/zconbt/bin
000118 export PATH
```

Material: Gabriel Paiva | Apresentação: Pedro Alonso

Caso o utilitário **zconbt (z/OS Connect Build Toolkit)** tenha sido instalado em um diretório fora dos caminhos padrões definidos no ambiente OMVS, ele só pode ser executado informando o caminho absoluto (como /u/pedroa/zconbt/bin/zconbt.zos).

Nesse slide vamos incorporar o zconbt ao conjunto de **utilitários padrão do sistema**, permitindo sua execução direta via terminal OMVS (somente zconb.zos), sem necessidade de especificar o path completo.

Para isso precisamos adicionar o path da pasta /bin do utilitário aos paths padrões do OMVS. Portanto vamos alterar o arquivo etc/profile, que é executado ao inicializar o OMVS.

1 – Abra o arquivo profile da pasta /etc no modo de edição.

2 – Procure a sessão do arquivo em que são adicionados os PATHS padrões do sistema. (Observe que são adicionados vários PATHS na mesma variável, e eles são separados por “:”)

3 – Por fim, adicione uma linha referente ao path da pasta bin do utilitário, antes da linha que exporta a variável. Algo como:

`PATH=$PATH:<pasta-pai-de-zconbt>/zconbt/bin`

(vide o exemplo da imagem)

4 – Salve o arquivo. Agora toda vez que você abrir o OMVS, esse PATH será adicionado aos PATHS padrões, portanto você pode usar o utilitário digitando somente zconbt.zos. Teste executando esse comando em qualquer pasta:

```
zconbt.zos  --help
```

Gerando os artefatos do API Requester no z/OS.

Criando a estrutura de pastas necessária para gerar os artefatos

```
<nome-da-api>/
├── apiInfoFile/
├── archives/
├── logs/
├── structures/
├── <arquivo-properties> (UTF-8)
└── <arquivo-swagger> (UTF-8)
```

Para fazermos o deploy de um API Requester no servidor z/OS Connect, precisamos antes gerar alguns artefatos como o arquivo.ara e os copybooks de request, response e informações sobre endpoints. Para isso precisamos que criar uma estrutura de pastas como a da imagem, portanto vamos seguir os seguintes passos:

1 - Criar a pasta principal do projeto.

`mkdir <nome-da-api>`

2 - Entrar na pasta principal.

`cd <nome-da-api>`

3 - Criar as subpastas seguindo a convenção oficial IBM.

`mkdir archives` => Para armazenar o .ara e o summary report
`mkdir apiInfoFile` => Para os arquivos de informação de cada endpoint (I)
`mkdir structures` => Para os copybooks de request (Q) e response (P)
`mkdir logs` => Para armazenar logs de build

4 - Agora precisamos colocar também nessa pasta o arquivo Swagger da API (.json ou .ymal) e o arquivo .properties, ambos em UTF-8. (Veremos como preencher o arquivo .properties no próximo slide)

Podemos fazer isso de 2 formas: criar um arquivo UTF-8 vazio no z/OS e depois preenche-lo [I] ou passar um arquivo em UTF-8 já preenchido via SFTP (ou similares) [II].

I - Para criar um arquivo UTF-8 vazio utilizamos os comandos:

`touch <nome-do-arquivo>`
`chtag -t -c 1208 <nome-do-arquivo>` => Só recomendamos a utilização desse comando caso o arquivo esteja vazio, já que ele não converte o conteúdo.

Para verificarmos o tipo de encoding de um arquivo execute:

`chtag -p <nome-do-arquivo>`

Agora é só preencher o arquivo. Lembre-se de selecionar o modo de edição UTF-8.

II – E caso você passe um arquivo UTF-8 já preenchido via SFTP, irá funcionar mesmo se ele estiver marcado como untagged.

Configurando o arquivo properties

```
000001    apiDescriptionFile=./watson/watson_translator_v2.json
000002    dataStructuresLocation=./watson/structures
000003    apiInfoFileLocation=./watson/apiInfoFile
000004    requesterPrefix=API
000005    connectionRef=BluemixAPIConnect
000006    logFileDirectory=./watson
000007    language=COBOL
```

Agora vamos preencher o arquivo .properties. Ele é responsável por definir o Swagger que será analisado para gerar os artefatos, e onde cada tipo de artefato será armazenado.

Se você cumpriu o último passo corretamente, configure da seguinte forma:

```
apiDescriptionFile=./<arquivo-swagger>
dataStructuresLocation=./structures
apiInfoFileLocation=./apiInfoFile
requesterPrefix=<prefixo-de-3-letras-para-os-copybooks>
connectionRef=<nome-da-conexão-a-api>
logFileDirectory=./logs
language=COBOL (ou PLI-ENTERPRISE)
```

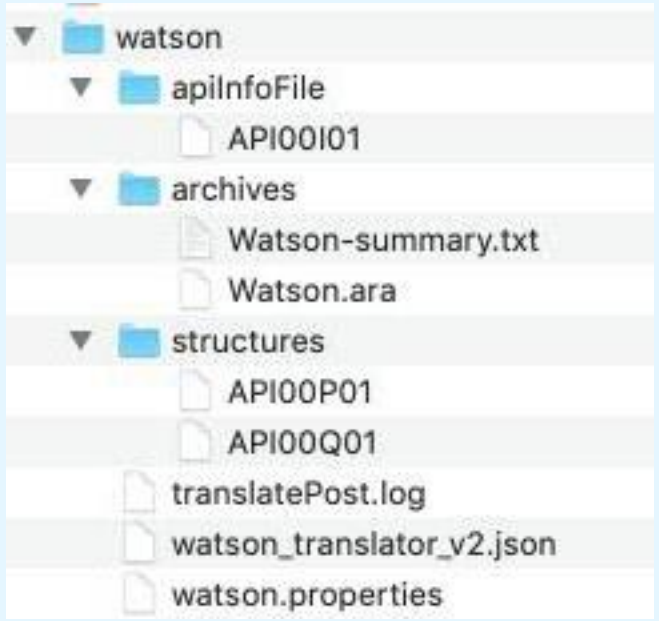
Para mais informações sobre como preencher o .properties acesse:

The build toolkit properties file

Aqui você pode ver quais são os atributos obrigatórios, quais são os opcionais e a explicação de cada atributo.

Gerando os artefatos

```
GABRIEL:/u/pedroa/watson: >zconbt --properties=./watson.properties --file=./archives/Watson.ara
```



Depois de configurar as pastas e os arquivos em UTF-8, vamos executar o comando responsável por gerar os artefatos. Dentre os artefatos gerados estão o .ara e os copybooks que são necessários para o funcionamento do requester.

O comando para gerar os artefatos é:

```
zconbt.zos --properties=<path-properties> --file=<path-onde-sera-gerado-o-ara>
```

Se você cumpriu o últimos passos corretamente, siga o passo a passo:

1 – Certifique-se que você está na pasta <nome-da-api>.

2 - Execute:

```
zconbt.zos --properties=./<arquivo-properties> --file=./archives/<arquivo-.ara>
```

Obs: Caso as propriedades consumes ou produces acusem erro, recomendamos a deleção das mesmas. Essas propriedades são ignoradas pelo z/OS Connect já que elas são definidas sempre como JSON.

Deploy do requester no servidor.

Configurando o servidor para o requester

```
GABRIEL:/u/pedroa/watson/archives: >ls
Watson.ara
GABRIEL:/u/pedroa/watson/archives: >cp Watson.ara /var/zosconnect/servers/myServ
er/resources/zosconnect/apirs/Watson.ara
GABRIEL:/u/pedroa/watson/archives: >ls /var/zosconnect/servers/myServer/resource
s/zosconnect/apirs
Watson.ara    petstore.ara
```

Para que o deploy do requester funcione, precisamos que o arquivo do requester (.ara) esteja na pasta correta. Para garantir essa etapa:

1 - Primeiro verifique se existe uma pasta reservada para os requesters (arquivos .ara). Caso não exista crie com esse comando (ou equivalente):

```
mkdir /var/zosconnect/servers/<nome-do-servidor>/resources/zosconnect/apirs
```

2 – Depois disso copie o arquivo .ara criado para esse pasta. Se você ainda está na pasta root para a geração dos artefatos (como /u/pedroa/watson no exemplo), execute os comandos:

```
cd archives
cp <arquivo-.ara> /var/zosconnect/servers/<nome-do-servidor>/resources/zosconnect/apirs/<arquivo-.ara>
```

Alterações do server.xml

```
<!-- Enable features -->
<featureManager>
    <feature>zosconnect:zosConnect-2.0</feature>
    <feature>zosconnect:zosConnectCommands-1.0</feature>
    <feature>apiDiscovery-1.0</feature>
    <feature>zosconnect:apiRequester-1.0</feature>
</featureManager>
```

```
<zosconnect_endpointConnection id="BluemixAPIConnect"
    host="https://watson-api-explorer.myblumix.net"
    port="80"/>
```

Agora precisamos atualizar o servidor para que ele utilize o requester, para isso precisamos:

1 – Adicionar a feature de requester ao server.xml. Para isso, dentro da tag <featureManager> **adicione** a feature “zosconnect:apiRequester-1.0” da seguinte forma:

```
<featureManager>
  <feature>zosconnect:apiRequester-1.0</feature>
</featureManager>
```

Obs.: não é necessário remover as demais features.

2 – Criar uma conexão a URL base aonde vamos fazer as requisições:

```
<zosconnect_endpointConnection id=”<nome-da-conexão-a-api>"
  host=”<url-raiz-das-requests>"
  port=”<porta-para-requisições>"/>
```

Para mais opções de configuração da conexão, clique aqui.

Alterações do server.xml (continuação)

```
<zosconnect_apiRequesters
    location="/var/zosconnect/servers/myServer/resources/zosconnect/apirs">
    <apiRequester name="Watson" connectionRef="BluemixAPIConnect"/>
</zosconnect_apiRequesters>
```

Por fim, vamos adicionar a tag que adiciona a lista de requesters (zosconnect_apiRequesters), e vamos adicionar a essa lista a tag da API que criamos agora (um apiRequester):

1 – Vai ficar algo como:

```
<zosconnect_apiRequesters
  location="<path-para-pasta-dos-requesters>">
  <apiRequester name="<nome-do-.ara>" connectionRef="<nome-da-conexão-a-api>"/>
</zosconnect_apiRequesters>
```

Obs. 1: caso você tenha configurado o atributo connectionRef do .properties com a referência simbólica para a conexão antes do build, o arquivo .ara já contem essa referência. Mas para garantir o funcionamento, vamos passar a connectionRef no server.xml também.

Obs. 2: caso você queira configurar outro requester, é só adicionar outra tag apiRequester também dentro de zosconnect_apiRequesters.

2 – Salve o arquivo.

Refresh no servidor

```
SDSF DA S0W1      S0W1      PAG   0   CPU   23
COMMAND INPUT ===> /F ZCONSRV,ZCON,REFRESH
```

```
<!-- Enable features -->
<featureManager>
    <feature>zosconnect:zosConnect-2.0</feature>
    <feature>zosconnect:zosConnectCommands-1.0</feature>
    <feature>apiDiscovery-1.0</feature>
    <feature>zosconnect:apiRequester-1.0</feature>
</featureManager>
```

```
API requester Swagger-Petstore_1.0.7 is deployed successfully.
```

Agora vamos recarregar o servidor para que ele funcione com a nova configuração.

1 – Para isso execute o comando de console:

/F <nome-da-stc>,ZCON,REFRESH

Para que esse comando funcione é necessário que a seguinte feature esteja configurada no server.xml:

<feature>zosconnect:zosConnectCommands-1.0</feature>

2 – Verifique o messages.log para ver se o deploy foi bem sucedido.

Criando, gerando e executando uma requisição.

Parte 2

Nesse link, temos orientações e arquivos sample para criar um programa COBOL que faz uma requisição a uma API, é a parte 2 desse guia:

https://github.com/gabriel-paiva17/Compilar-Linkeditar-E-Executar-COBOL