LocalStack — Testando serviços AWS

Evandro F. Souza
Apr 3, 2018 · 5 min read

https://unsplash.com/photos/jTP3p3tAF-E

Originalmente criado pela *Atlassian*, porém agora desenvolvido e mantido de forma independente, o *LocalStack* permite que serviços da AWS sejam emulados diretamente no seu computador. Para aqueles que querem aprender e testar os serviços da AWS, isto é perfeito para praticar sem medo de vir aquele valor surpresa na fatura do cartão. Para aqueles que já utilizam AWS em seus projetos, podem utilizar esta ferramenta para melhorar a qualidade de sua aplicação, visto que é possível utilizar nos testes automatizados.

Serviços compatíveis

No momento que escrevo este post, os seguintes serviços são emulados pelo *LocalStack*, abaixo a lista com cada serviço e a porta no qual ele roda:

- API Gateway: http://localhost:4567
- Kinesis: http://localhost:4568
- DynamoDB: http://localhost:4569
- DynamoDB Streams: http://localhost:4570
- Elasticsearch: http://localhost:4571
- S3: http://localhost:4572
- Firehose: http://localhost:4573
- Lambda: http://localhost:4574
- SNS: http://localhost:4575
- SQS: http://localhost:4576
- Redshift: http://localhost:4577
- ES (Elasticsearch Service): http://localhost:4578
- SES: http://localhost:4579
- Route53: http://localhost:4580
- CloudFormation: http://localhost:4581
- CloudWatch: http://localhost:4582
- SSM: http://localhost:4583

> *Observação: Não mostramos neste post, porém é possível configurar cada serviço e porta independentemente.*

Como eu uso?

Para demonstrar como funciona, o tutorial esta dividido em três etapas:

1. Instalação do serviço

2. Inicialização do serviço
3. Exemplo de uso

> *Observação: Nas duas primeiras etapas, será abordado os problemas que ocorreram comigo e o que eu fiz para solucionar.*

1 — Instalação do serviço

Antes de instalar, tenha certeza que possui os seguintes requisitos:

- Python(ambos 2.x e 3.x)
- pip
- npm
- java/javac (Java 9)
- Mvn

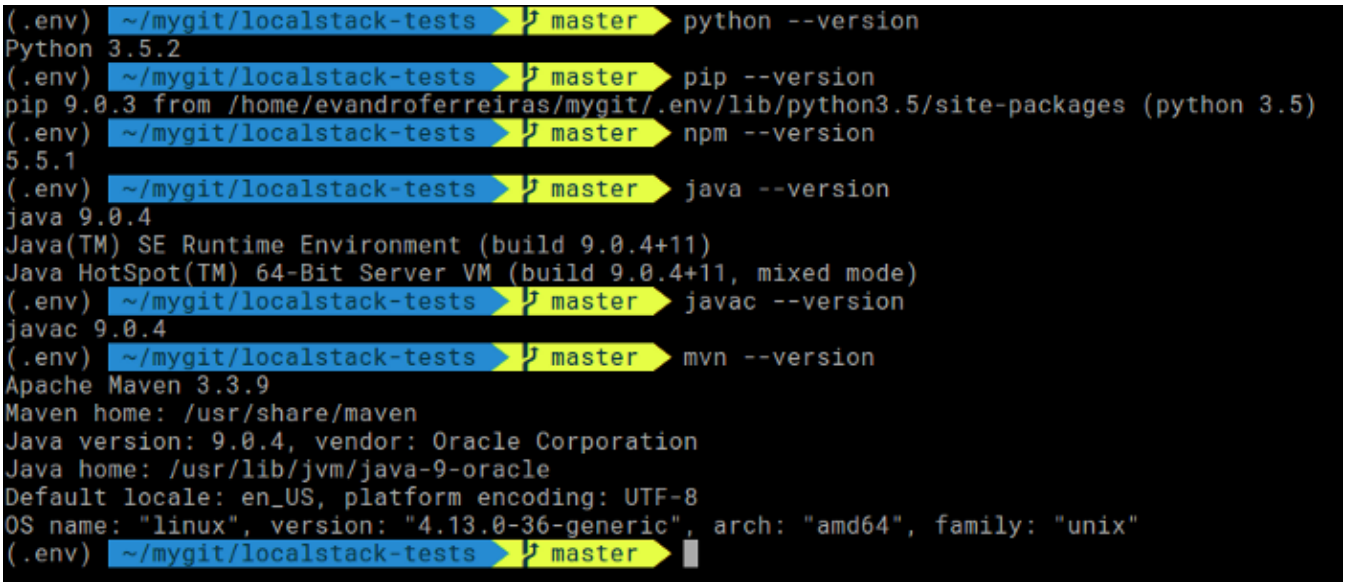


Figura 1 — Versão de cada um dos requisitos

Após confirmado, execute os seguintes comandos:

1 - Crie e ative um *virtual enviroment* com Python3.5:

```
1  virtualenv -p python3.5 .env
2  source .env/bin/activate
```

prepare_virtualenv.sh hosted with ❤ by **GitHub** **view raw**

2 - Com o *virtual enviroment* habilitado, execute o comando *pip* para instalar o serviço:

```
1  pip install localstack
```

install_localstack.sh hosted with ❤ by **GitHub** **view raw**

Possíveis problemas

Durante a minha instalação do serviço, ocorreram alguns problemas, abaixo vou descrever cada um deles e demonstrar como avancei.

Primeiramente, caso ocorra qualquer problema durante instalação do serviço, revise as versões dos requisitos, demonstrado na Figura 1. Durante a elaboração deste tutorial, eu instalei o *LocalStack* em dois computadores diferentes, ao tentar instalar no segundo, ocorreu um erro que só foi solucionado ao atualizar a versão do *pip* para a mesma do computador anterior.

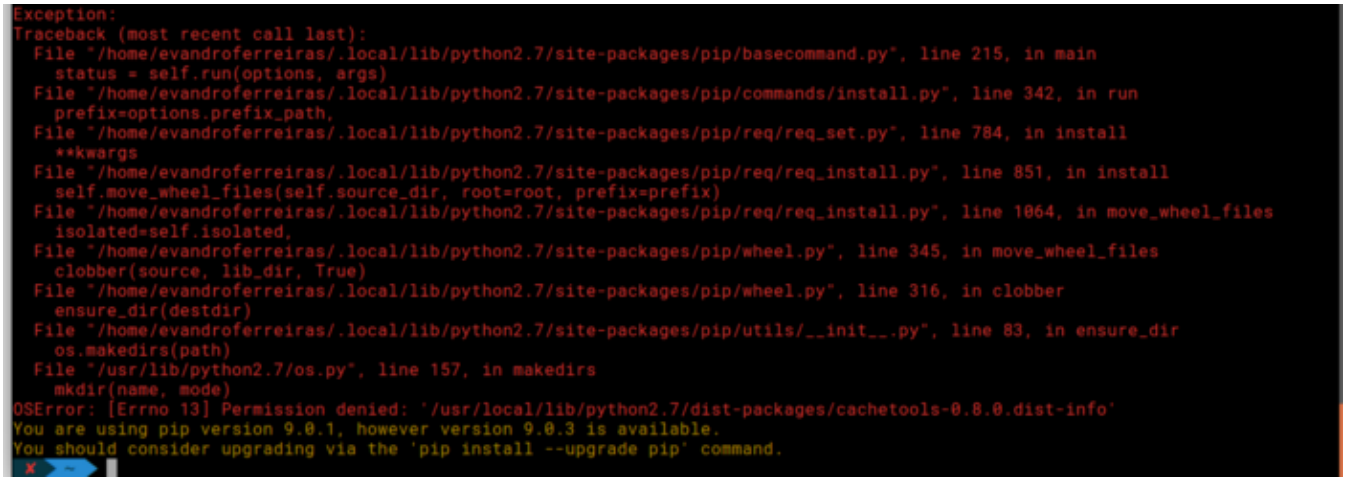
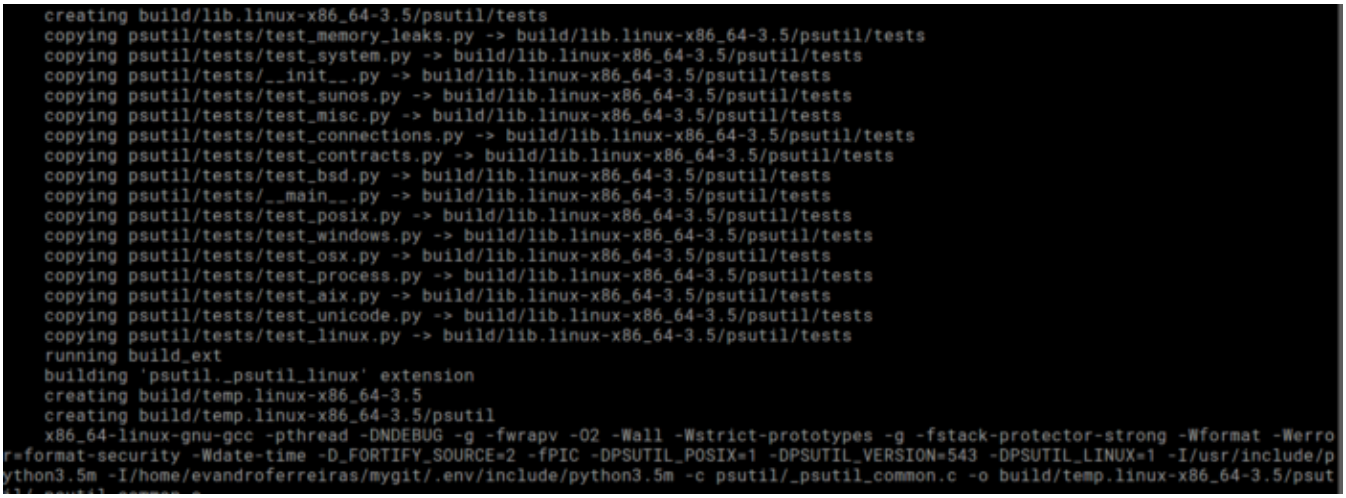


Figura 2 — [Errno 13] Permission Denied

Caso ocorra erro de permissão, como o demonstrado na Figura 2. Tenha certeza que foi habilitado o seu VirtualEnv com Python 3.5.



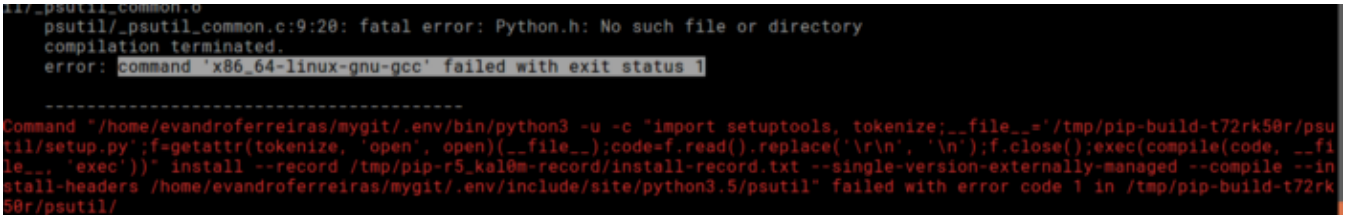


Figura 3 — Python.h: No such file or directory

Caso ocorra um erro referenciando a falta do arquivo Python.h, como demonstrado na Figura 3. Rode o seguinte comando:

```
sudo apt-get install python3-dev
```

install_python_dev.sh hosted with ❤ by **GitHub** **view raw**

2 — Iniciando serviço

Após instalado, inicie o serviços com o comando:

```
localstack start
```

start_localstack.sh hosted with ❤ by **GitHub** **view raw**

Possíveis problemas

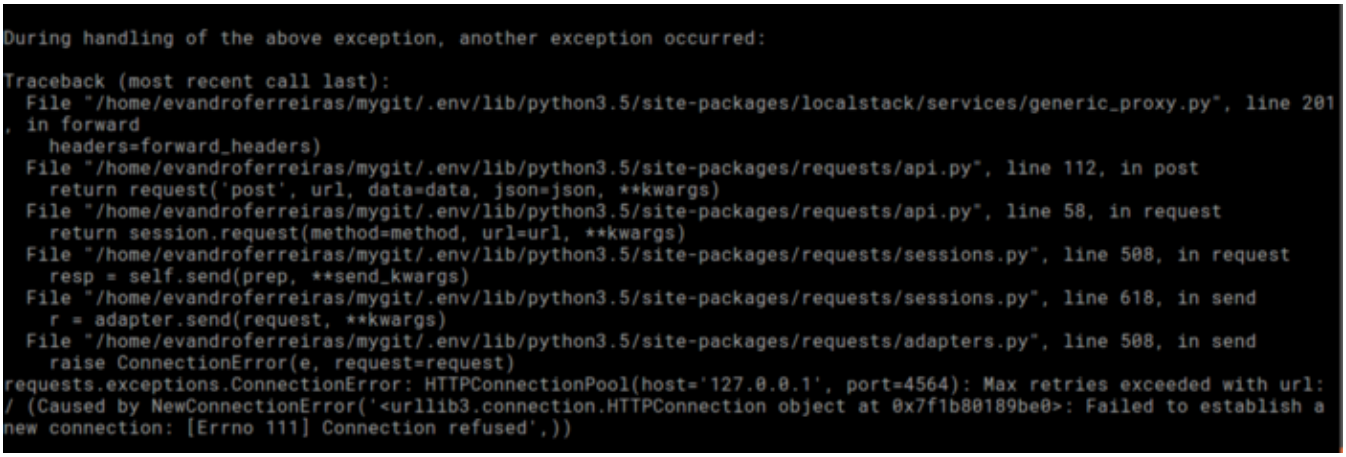


Figura 4 — Connection refused

A inicialização do serviço deve demorar dependendo da velocidade da sua conexão, pois serão baixados diversos pacotes para cada serviço. Caso ocorra algum erro, como o ilustrado na Figura 4, pare a execução (Ctrl + C) e inicie novamente em modo *Debug*, com o comando abaixo:

```
DEBUG=1 localstack start
```

start_localstack_debug.sh hosted with ❤ by **GitHub** **view raw**

```
Starting mock Lambda service (http port 4574)...
Starting mock SES (http port 4579)...
Starting mock SNS (http port 4575)...
Starting mock Redshift (http port 4577)...
Starting mock API Gateway (http port 4567)...
Starting local Elasticsearch (http port 4571)...
Starting mock SSM (http port 4583)...
Starting mock ES service (http port 4578)...
Starting mock SQS (http port 4576)...
Listening at http://:::4565
Error: Unable to access jarfile /home/evandroferreiras/mygit/.env/lib/python3.5/
site-packages/localstack/infra/elasticmq/elasticmq-server.jar
Starting mock DynamoDB (http port 4569)...
Starting mock CloudWatch (http port 4582)...
Starting mock DynamoDB Streams service (http port 4570)...
Error: Invalid or corrupt jarfile DynamoDBLocal.jar
```

Figura 5 — Invalid or corrupt jarfile DynamoDBLocal.jar

Uma vez executado o comando, o erro deve ocorrer novamente, porém dessa vez é possível identificar o problema, conforme é ilustrado na Figura 5. No meu caso, o arquivo *DynamoDBLocal.jar* estava corrompido. Para resolver, eu exclui os pacotes baixados para forçar uma nova tentativa:

```
1  rm -rf .env/lib/python3.5/site-packages/localstack/infra
```

delete_packages_localstack.sh hosted with ❤ by **GitHub** **view raw**

Exemplo de uso

Agora com o *LocalStack* instalado e rodando, vamos testar ele com um exemplo simples, se quiser pode acompanhar pelo repositório no *GitHub*.

No exemplo, utilizei a SDK de Python oficial da Amazon(boto3) para:

1. Criar uma fila no SQS
2. Criar uma tabela no DynamoDB
3. Enviar mensagens para fila SQS
4. Ler as mensagens da fila SQS
5. Gravar os dados no DynamoDB
6. Ler os dados do DynamoDB

No arquivo ***main.py*** é possível verificar cada um dos passos:

```python
from sqs_wrapper import send_json, return_all, create_queue
from dynamodb_wrapper import create_table, send_data, return_data

# 1) Criar uma fila no SQS
# 2) Criar uma tabela no DynamoDB
create_table()
create_queue()

# 3) Enviar mensagens para fila SQS
send_json({
    'first_number' : 14,
    'second_number' : 20
})
send_json({
    'first_number' : 15,
    'second_number' : 20
})

# 4) Ler as mensagens da fila SQS
result = return_all()
for r in result:
    # 5) Gravar os dados no DynamoDB
    send_data(r)

# 6) Ler os dados do DynamoDB
res_dynamo = return_data()
for r in res_dynamo:
    print(r)
```

main.py hosted with ❤ by **GitHub** **view raw**

O código para consumir os serviços emulados é exatamente o mesmo para utilizar os da própria AWS. A unica diferença é o ***endpoint_url***. Isto é possível notar no arquivo ***sqs_wrapper.py***, lá é apontado para o *endpoint* do SQS do *LocalStack* (http://localhost:4576/).

> *Observação: No início do post é listado todos os serviços e suas respectivas portas default.*

```python
import boto3
import json

def __init_sqs():
    return boto3.resource('sqs', endpoint_url="http://localhost:4576")

def create_queue():
    sqs = __init_sqs()
```

```
    sqs.create_queue(QueueName='teste')

def __get_queue_by_name(name):
    sqs = __init_sqs()
    return sqs.get_queue_by_name(QueueName=name)

def send_json(msg):
    queue = __get_queue_by_name('teste')
    queue.send_message(MessageBody= json.dumps(msg))

def return_all():
    queue = __get_queue_by_name('teste')
    messages = queue.receive_messages()
    result = []
    while len(messages) > 0:
        message = messages[0]
        json_returned = json.loads(message.body)
        result.append(json_returned)
        message.delete()
        messages = queue.receive_messages()
    return result
```

sqs_wrapper.py hosted with ❤ by **GitHub** **view raw**

Caso você tenha clonado o repositório do *GitHub* e quer ver o resultado, rode o comando abaixo:

```
python main.py
```

exec_python.sh hosted with ❤ by **GitHub** **view raw**

Após rodado, será criado uma tabela *DynamoDB* e uma fila SQS no seu *LocalStack*. É possível visualizar tais recursos em um *Dashboard*. Para isto abra outra aba no terminal e execute o seguinte comando:

```
localstack web
```

localstack_web.sh hosted with ❤ by **GitHub** **view raw**

Após executado, você pode acessar a url http://localhost:8080 no seu browser.

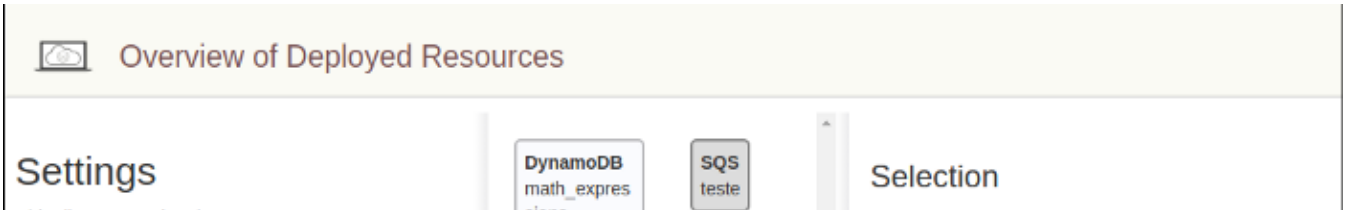


Figura 6 — Dashboard do LocalStack

Conforme é possível observar na Figura 6, o *Dashboard* apresenta uma visão geral dos serviços criados no *LocalStack*. Note que esta selecionado a fila “teste” do SQS e o painel à direita apresenta informações importantes, como por exemplo o ARN(Amazon Resource Name). O ARN é o identificador de cada recurso criado, é útil para os casos de um serviço que se relaciona com outros (como por exemplo um tópico SNS e uma fila SQS).

E o que mais?

O *LocalStack* abre um leque de opções de uso. Ficou mais fácil e barato desenvolver e testar suas aplicações, agora não precisa mais criar recursos na *Amazon* somente para testar funcionalidades ainda em desenvolvimento. Ele também auxilia nos testes automatizados, visto que é possível utilizar como um *mocking framework*. Por fim, o *LocalStack* também auxilia aqueles que querem estudar AWS e ficar livre do receio de vir aquela conta surpresa no final do mês.

Uma coisa importante para se notar é que o LocalStack não persiste os dados. Assim que o serviço para de rodar, os recursos criados são perdidos. Atualmente, o *LocalStack* tem somente versão *Free*, porém já temos um *spoiler* ao acessar o site oficial e notar que está previsto uma versão “Pro Edition”. Dentre as diversas diferenças entre as versões, a “Pro Edition” possui como um dos diferenciais a persistência de dados.

É interessante, aparentemente a proposta deles é você poder ter uma nuvem privada. Fiquei curioso para estudar os possíveis casos de uso quando esta nova versão sair.

. . .

Se quiser trocar uma ideia ou entrar em contato comigo, pode me achar no Twitter (@e_ferreirasouza) ou Linkedin.

Grande abraço e até a próxima!

AWS Software Development Serverless Development Technology