

LocalStack

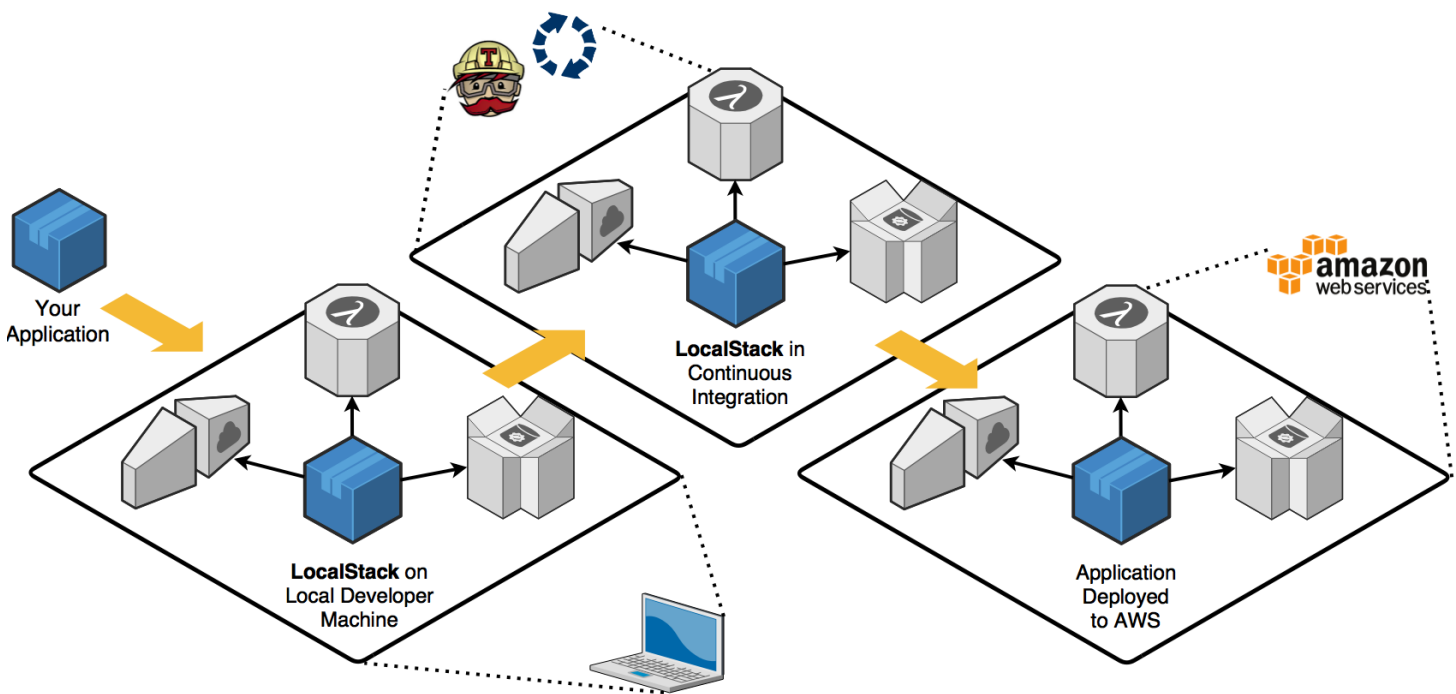
January 26, 2021

Como simular os recursos da AWS localmente com LocalStack

Uma abordagem para desenvolvimento e testes de aplicações *cloud* em ambiente local: rápida, sem custo e offline.


Neste artigo vamos estudar uma abordagem para o desenvolvimento de aplicações que utilizem AWS, para desenvolvimento e testes em Integração Contínua. Ao fim, queremos poder executar os principais serviços da AWS (com suas APIs oficiais) utilizando Docker.

Isso nos permite um ciclo de desenvolvimento mais rápido e eficiente, além de conseguir incluir os serviços da AWS em nossa pipeline.



Requisitos

Não precisa de muita coisa para seguir este tutorial além de pouca familiaridade com docker e AWS, mas são necessárias algumas instalações com, no mínimo, as seguintes versões:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It displays the output of three commands: 'docker --version' showing '20.10.1, build 831e1bea', 'python --version' showing '2.7.17', and 'pip --version' showing '20.3.3'.

```
→ docker --version
Docker version 20.10.1, build 831e1bea

→ python --version
Python 2.7.17

→ pip --version
pip 20.3.3
```

As dores de cabeça que surgiram com a computação em nuvem

A computação em nuvem trouxe muitas mudanças e benefícios para o ciclo de desenvolvimento - menos preocupação com infraestrutura e mais preocupação com código. Mas esses benefícios chegaram a um custo para os desenvolvedores: agora não é mais possível ter um ambiente de desenvolvimento completo e isolado.

Enquanto cloud-based development não é popularizado, desenvolvedores de aplicações em nuvem trabalham em um ambiente híbrido - utilizando recursos *cloud* da AWS, por exemplo, é comum alocar uma infraestrutura completa de "desenvolvimento" para experimentar os serviços e permitir que os devs trabalhem.

Essa abordagem tem diversos pontos negativos, mas vou destacar dois principais: custos e limitações.

Por parte do custo, não existe uma versão gratuita dos serviços cloud para desenvolvimento, toda a utilização de recursos é cobrada e isso inclui testes, desenvolvimento, pipeline, experimentação, POCs e etc. Diferente de trabalhar em um cenário *on-premise* em que tudo é executado de forma local. Isso significa que testes de integração ficam prejudicados e limitados as bibliotecas de testes que tentam simular aquele ambiente.

Já na parte das limitações - além da latência e configurações de rede no caso de VPCs - muita liberdade do time de desenvolvimento é perdida, visto que eles estão atuando em um ambiente real que tem impacto imediato na conta e nos outros desenvolvedores. Em um time diverso, com pessoas com muita ou pouca experiência em *cloud* o ambiente cloud se torna uma dor de cabeça para quem utiliza e para quem paga a conta.

1. Setup
2. Provisionar/Utilizar recursos de forma transparente com aws-cli
3. Utilização (desenvolvimento e testes de integração em CI)

LocalStack FTW: trazendo a nuvem para o ambiente de desenvolvimento


LocalStack surgiu para resolver justamente este problema: executando de forma local (Docker), podemos ter todos os serviços da AWS disponíveis e fazer a bagunça que quisermos - offline, sem custos, sem limitações e sem atrapalhar os colegas. Por ser eficiente e portátil permite que o desenvolvimento utilize todo o poder da nuvem em um container.



O conceito é simples: ter um ambiente completamente funcional da AWS sendo executado de forma local, respeitando as APIs oficiais e sendo o mais transparente possível. Além disso, LocalStack simula erros reais da AWS e executa os serviços de forma totalmente desacoplada.

Instalando e Executando LocalStack

Dado todo esse contexto, agora veremos como instalar e executar os serviços AWS de fato com LocalStack. E é bem simples, instala o pacote `localstack` pelo pip e então executa com o comando `start`:




```
→ pip install localstack
...
Successfully built localstack

→ localstack start
...
Waiting for all LocalStack services to be ready
Ready.
```

Se você digitar `docker ps` vai ver que temos o container executando a partir da imagem `localstack/localstack` expondo a porta 4566.

E com isso, temos a AWS executando em nossa máquina.


Para utilizar os recursos, podemos utilizar as próprias ferramentas da AWS para interagir, alterando apenas as configurações de perfil. Eu criei um perfil "localstack", mas pode ser o global, ou qualquer outra nomenclatura.



```
→ ~ aws configure --profile localstack
AWS Access Key ID [None]: test
AWS Secret Access Key [None]: test
Default region name [None]:
Default output format [None]:
```

E agora, se você está acostumado com a AWS e digitar algum comando, por exemplo `aws lambda list-functions`, verá um erro porque não temos credenciais válidas da AWS.

Então, para conseguir executar os serviços, precisamos sobrescrever o endpoint padrão da AWS, utilizando o nosso local:



```
→ aws lambda list-functions --profile localstack | json_pp
An error occurred (UnrecognizedClientException) when calling
the ListFunctions operation: The security token included in
the request is invalid.

→ aws lambda list-functions --endpoint-url
http://localhost:4566 --profile localstack | json_pp
{
  "Functions" : []
}
```

Exemplo de provisionamento e utilização de uma fila com SQS

Agora podemos utilizar todos os recursos da AWS pelo CLI ou pelas SDKs. Para o nosso caso de exemplo, vamos criar uma fila de pedidos a serem atendidos.


A primeira etapa é criar e provisionar a fila e, assim como ocorre em um ambiente real da AWS, podemos fazer dessa forma:



```
→ aws sqs create-queue --queue-name orders \  
> --endpoint-url http://localhost:4566 \  
> --profile localstack  
  
{  
  "QueueUrl": "http://localhost:4566/000000000000/orders"  
}
```

Agora, vamos simular que dois pedidos diferentes foram feitos e vamos enviar um por um para a fila:

```
[  
  {  
    "id": "ORDER#0001",  
    "customer": "Myreli",  
    "items": ["Fries", "Chocolate Shake"]  
  },  
  {  
    "id": "ORDER#0002",  
    "customer": "William",  
    "items": ["Cheeseburger Combo Meal"]  
  }  
]
```



```
→ aws sqs send-message \  
--queue-url http://localhost:4566/000000000000/orders \  
--message-body "  
{\"id\": \"ORDER#0001\", \"customer\": \"Myreli\", \"items\":  
[\"Fries\", \"Chocolate Shake\"]}" \  
--endpoint-url http://localhost:4566 \  
--profile localstack  
  
{  
  "MD5ofMessageBody" : "5b911b1e3bbea61a97612829e92304f9",  
  "MessageId" : "d1b1475a-6aee-2bcb-49ee-7d498a3fa96e"  
}
```

E por último, vamos consumir essas mensagens da fila, assim como faríamos de uma fila na infraestrutura da AWS:


```
→ aws sqs receive-message \
--queue-url http://localhost:4566/000000000000/orders \
--endpoint-url http://localhost:4566 \
--profile localstack

{
  "Messages" : [
    {
      "MessageId" : "d1b1475a-6aee-2bcb-49ee-7d498a3fa96e",
      "MD5OfBody" : "5b911b1e3bbea61a97612829e92304f9",
      "Body" : "
{\\"id\\":\\"ORDER#0001\\",\\"customer\\":\\"Myreli\\",\\"items\\":
[\\"Fries\\",\\"Chocolate Shake\\"]}"}
    ]
  }
}
```

Isso encerra o exemplo do uso de SQS com LocalStack, utilizando a ferramenta oficial da AWS para tal.

Para expandir o exemplo, o mesmo pode ser feito com uma SDK oficial em Node ou Kotlin, por exemplo. E aí basta configurar quando sobreescrever a URL original - na pipeline de CI e no ambiente de desenvolvimento.

Em Java, por exemplo, a implementação seria bem simples:

```
// Builder em ambiente Cloud
SqsClient.builder()
    .region(@Region)
    .build();
// Builder em ambiente Local
SqsClient.builder()
    .region(@Region)
```

```
.endpointOverride("http://localhost:4576")  
.build();
```

Próximos Passos

Isso conclui a introdução ao LocalStack e já permite que seja implantado em projetos reais, mas é só uma ponta da ferramenta. Recomendo explorar o [repositório oficial](#) para otimizar o fluxo de trabalho.

Além disso, é um bom exercício para praticar implementar exatamente o mesmo exemplo apresentado aqui com alguma SDK oficial.