**Debian and a LocalStack - Setting Up a Local AWS S3 Environment for Development**

[David Garcia](https://medium.com/@own3dh2so4?source=post_page---byline--573353200d32---------------------------------------) [David Garcia](https://medium.com/@own3dh2so4?source=post_page---byline--573353200d32---------------------------------------)

**Introduction**

These days, it’s very common for Software Engineers to create applications that require access to third-party services. Examples of such third-party services include APIs, databases, and cloud-based applications. Many applications and companies operate and deploy their services on Amazon Web Services (AWS). Additionally, one of the first and most renowned services offered by AWS is Simple Storage Service (S3).

Press enter or click to view image in full size

A person sitting at a table with a computer and cookies

AI-generated content may be incorrect.

When Docker entered the realm of development, setting up a local database for development became extremely easy. You could create a docker-compose.yml file where you had both the database and your service defined. This allowed for relatively simple end-to-end testing of certain services locally.

But what about other third-party services? Do I have to create mocks in my tests? Furthermore, in the case of certain services like S3, operations (both read and write) incur a cost. Would I have to pay every time I want to perform an end-to-end test?

Each type of third-party service has its way of being simulated. In this case, I’m here to explain how we can simulate S3 locally, enabling us to carry out all the operations that S3 allows without needing to access the AWS cloud.

**LocalStack**

[LocalStack](https://www.localstack.cloud/) is a powerful tool used in software development for creating local environments that replicate various AWS cloud services. It allows developers to emulate the behavior of AWS services, such as S3 (Simple Storage Service), SQS (Simple Queue Service), SNS (Simple Notification Service), DynamoDB (NoSQL database), and many others, on their local machines.

Essentially, LocalStack provides a local sandbox environment that mimics the AWS cloud infrastructure, enabling developers to test their applications locally without incurring any costs associated with using the actual AWS services. This is particularly useful for development and testing purposes, as it allows developers to experiment with AWS services, develop and debug their applications, and run end-to-end tests without relying on the internet or incurring expenses.

Moreover, this tool comes neatly packaged in a [Docker image](https://hub.docker.com/r/localstack/localstack), ready to be configured in our typical docker-compose.yml file.

In this article, we won’t delve deeply into all the options and what can or cannot be done, as LocalStack has very good documentation on its website.

**Example**

In this section, we’re going to perform an exercise as generic and simple as possible. For this reason, we’ll create a playground environment with the following requirements:

* We’ll have two containers: a Debian and a LocalStack.
* When we start the playground environment (docker compose up -d), the S3 bucket named ‘*david-garcia-medium*’ must exist.
* From the Debian container, using AWS CLI, I should be able to perform operations with S3 as if it were the actual AWS itself.

First of all, let’s display the file structure of our repository.

./  
 playground/  
 aws/  
 config  
 credentials  
 aws\_cli/  
 awscliv2-amd64.zip  
 awscliv2-arm64.zip  
 localstack/  
 s3.sh  
 .env.debian  
 .env.localstack  
 docker-compose.yml  
 Dockerfile

In the ‘playground’ directory, we have the necessary configuration to make our playground environment work. In this case, we have three subdirectories:

* **aws**: which contains the authentication configuration required by AWS CLI to access AWS.
* **aws\_cli**: where we have the downloaded AWS CLI binaries — in this case, we’ve downloaded both the AMD (for Linux in general) and ARM (for Mac with M1 in general) versions.
* **localstack**: with a script, which will be used to configure the startup of the LocalStack container.

The .env.debian and .env.localstack files are files with declared environment variables that will be mounted to the containers in the docker-compose.yml. They contain the following information

# .env.debian  
# AWS cli configuration  
AWS\_PROFILE=localstack

The .env.debian is very simple, and only contains [the AWS environment variable](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html#cli-configure-files-using-profiles) used by AWS CLI to know what profile must be used by default. If we review the files available in playground/aws path we find the following

# config  
[profile localstack]  
output = json  
endpoint\_url = http://localstack:4566  
region = us-east-1

# credentials  
[localstack]  
aws\_access\_key\_id=test  
aws\_secret\_access\_key=test

In the playground/aws/config file, we overwrite the endpoint\_url to use our LocalStack as AWS backend instead of the official AWS. In the playground/aws/credentials we configure the credentials to the localstack profile.

Let’s review the .env.localstack

# .env.localstack  
  
SERVICES=s3  
  
# Checker  
TRIES=30  
  
# Base  
HOSTNAME\_EXTERNAL=localstack  
  
LOCALSTACK\_API=http://localstack:4566  
LOCALSTACK\_HEALTH\_ENDPOINT=http://localstack:4566/health  
  
# S3  
BUCKET\_NAMES=david-garcia-medium  
  
# AWS cli configuration  
AWS\_DEFAULT\_PROFILE=localstack

These environment variables are typically utilized by the script located at playground/localstack/s3.sh. This script serves as a fundamental tool that executes once the LocalStack container is up and running, facilitating the creation of the pre-existing S3 bucket that we require. This script will be mounted in the path /etc/localstack/initi/ready.d and the localstack container will execute when it is ready. More info [here](https://docs.localstack.cloud/references/init-hooks/)

Let’s review the docker image with debian, the Dockerfile

FROM docker.io/debian:bookworm-slim  
  
# Added TARGETARCH to differenciate between amd64 (Linux and Windows) and arm64 (Mac) when install awscliv2.zip  
ARG TARGETARCH  
  
# Install zip  
RUN apt update && \  
 apt install -y zip=3.0-13 && \  
 rm -rf /var/lib/apt/lists/\*  
  
# Install awscli  
COPY playground/aws\_cli/awscliv2-${TARGETARCH}.zip .  
RUN unzip awscliv2-${TARGETARCH}.zip && \  
 ./aws/install && \  
 rm -rf awscliv2-${TARGETARCH}.zip aws  
  
# By default, this container does not execute anything; it simply sleeps indefinitely.  
CMD ["tail", "-f", "/dev/null"]

In this scenario, the image is quite straightforward, with the only sophisticated aspect being our readiness to build the image for both amd64 and arm64 architectures, utilizing the TARGETARCH Docker ARG. We use the binaries stored in playground/aws\_cli instead of downloading them via curl in the Dockerfile to ensure consistent installation of the same version of AWS CLI. This practice guarantees that each time we rebuild the image, we maintain uniformity in the awscli version.

And finally, we see the docker-compose.yml file with the definition of the services before running a test and verifying that everything works.

version: '3.7'  
  
  
services:  
 debian:  
 build:  
 context: .  
 dockerfile: Dockerfile  
 depends\_on:  
 - localstack  
 env\_file:  
 - .env.debian  
 volumes:  
 - ./playground/aws:/root/.aws:ro  
  
  
 localstack:  
 image: docker.io/localstack/localstack:3.1  
 ports:  
 - "4566:4566"  
 env\_file:  
 - .env.localstack  
 volumes:  
 - ./playground/localstack:/etc/localstack/init/ready.d/:ro  
 - ./playground/aws:/root/.aws:ro

As you can see, there’s nothing unfamiliar here that we haven’t commented on previously. Let’s test it:

$ docker compose down -v && docker compose up -d && docker compose exec debian bash  
...  
root@9698486ac759:/# # List the aws s3 buckets availables  
root@9698486ac759:/# aws s3 ls   
2024-04-02 17:57:15 david-garcia-medium  
root@9698486ac759:/# # Create a file and upload to S3  
root@9698486ac759:/# echo "hello medium" > my\_file.txt  
root@9698486ac759:/# aws s3 cp my\_file.txt s3://david-garcia-medium  
upload: ./my\_file.txt to s3://david-garcia-medium/my\_file.txt   
root@9698486ac759:/# aws s3 ls s3://david-garcia-medium  
2024-04-02 18:02:06 13 my\_file.txt  
root@9698486ac759:/# # Create a new S3 bucket  
root@9698486ac759:/# aws s3 mb "s3://my-new-bucket"  
make\_bucket: my-new-bucket  
root@9698486ac759:/# aws s3 ls   
2024-04-02 17:57:15 david-garcia-medium  
2024-04-02 18:04:20 my-new-bucket  
root@9698486ac759:/# # you can test other commands!

**Extra example**

Furthermore, LocalStack provides the files we have uploaded to S3 at the following path: http://localhost:4566/<bucket-name>, and you can download them using the S3 key.

$ curl http://localhost:4566/david-garcia-medium  
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">  
 <IsTruncated>false</IsTruncated>  
 <Marker/>  
 <Name>david-garcia-medium</Name>  
 <Prefix/>  
 <MaxKeys>1000</MaxKeys>  
 <Contents>  
 <Key>my\_file.txt</Key>  
 <ETag>"5047cdd6613d55aa1a3143639a81cf78"</ETag>  
 <Owner>  
 <DisplayName>webfile</DisplayName>  
 <ID>75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a</ID>  
 </Owner>  
 <Size>13</Size>  
 <LastModified>2024-04-02T18:02:06.000Z</LastModified>  
 <StorageClass>STANDARD</StorageClass>  
 </Contents>  
</ListBucketResult>  
  
$ curl http://localhost:4566/david-garcia-medium/my\_file.txt  
hello medium

To conclude, you have all the code available in the following [GitHub repository](https://github.com/own3dh2so4/localstack_s3_example).