LocalStack - A fully functional local AWS cloud stack

LocalStack provides an easy-to-use test/mocking framework for developing Cloud applications.

Currently, the focus is primarily on supporting the AWS cloud stack.

Announcements

- **2020-12-28**: Check out the LocalStack Pro **feature roadmap** here: https://roadmap.localstack.cloud - please help us prioritize our backlog by creating and upvoting feature requests. Looking forward to getting your feedback!
- **2020-09-15**: A major (breaking) change has been merged in PR #2905 - starting with releases after `v0.11.5`, all services are now exposed via the edge service (port 4566) only! Please update your client configurations to use this new endpoint.
- **2019-10-09**: **LocalStack Pro is out!** We're incredibly excited to announce the launch of LocalStack Pro - the enterprise version of LocalStack with additional APIs and advanced features. Check out the free trial at https://localstack.cloud
- **2018-01-10**: **Help wanted!** Please fill out this survey to support a research study on the usage of Serverless and Function-as-a-Service (FaaS) services, conducted at the Chalmers University of Technology. The survey only takes 5-10 minutes of your time. Many thanks for your participation!!
  - The result from this study can be found here
- **2017-08-27**: **We need your support!** LocalStack is growing fast, we now have thousands of developers using the platform regularly. Last month we have recorded a staggering 100k test runs, with 25k+ DynamoDB tables, 20k+ SQS queues, 15k+ Kinesis streams, 13k+ S3 buckets, and 10k+ Lambda functions created locally - for 0$ costs (more details to be published soon). Bug and feature requests are pouring in, and we now need some support from *you* to

keep the open-source version actively maintained. Please check out Open Collective and become a backer or supporter of the project today! Thanks, everybody for contributing. ❤️

- **2017-07-20**: Please note: Starting with version `0.7.0`, the Docker image will be pushed and kept up to date under the **new name** `localstack/localstack`. (This means that you may have to update your CI configurations.) Please refer to the updated **End-User License Agreement (EULA)** for the new versions. The old Docker image (`atlassianlabs/localstack`) is still available but will not be maintained any longer.

Overview

LocalStack spins up the following core Cloud APIs on your local machine.

Note: Starting with version `0.11.0`, all APIs are exposed via a single *edge service*, which is accessible on **http://localhost:4566** by default (customizable via `EDGE_PORT`, see further below).

- **ACM**
- **API Gateway**
- **CloudFormation**
- **CloudWatch**
- **CloudWatch Logs**
- **DynamoDB**
- **DynamoDB Streams**
- **EC2**
- **Elasticsearch Service**
- **EventBridge (CloudWatch Events)**
- **Firehose**
- **IAM**
- **Kinesis**
- **KMS**
- **Lambda**
- **Redshift**
- **Route53**
- **S3**
- **SecretsManager**
- **SES**
- **SNS**

- **SQS**
- **SSM**
- **StepFunctions**
- **STS**

In addition to the above, the **Pro version** of LocalStack supports additional APIs and advanced features, including:

- **Amplify**
- **API Gateway V2 (WebSockets support)**
- **AppSync**
- **Athena**
- **Batch**
- **CloudFront**
- **CloudTrail**
- **Cognito**
- **ECS/ECR/EKS**
- **ElastiCache**
- **EMR**
- **Glacier / S3 Select**
- **IAM Security Policy Enforcement**
- **IoT**
- **Kinesis Data Analytics**
- **Lambda Layers & Container Images**
- **Managed Streaming for Kafka (MSK)**
- **MediaStore**
- **Neptune Graph DB**
- **QLDB**
- **RDS / Aurora Serverless**
- **Timestream**
- **Transfer**
- **XRay**
- **Advanced persistence support for most services**
- **Interactive UIs to manage resources**
- **Test report dashboards**
- ...and much, much more to come! (Check out our **feature roadmap** here: https://roadmap.localstack.cloud)

Why LocalStack?

LocalStack builds on existing best-of-breed mocking/testing tools, notably kinesalite/dynalite and moto, ElasticMQ, and others. While these tools are *awesome* (!), they lack functionality for certain use cases. LocalStack combines the tools, makes them interoperable, and adds important missing functionality on top of them:

- **Error injection:** LocalStack allows to inject errors frequently occurring in real Cloud environments, for instance `ProvisionedThroughputExceededException` which is thrown by Kinesis or DynamoDB if the amount of read/write throughput is exceeded.
- **Isolated processes**: Services in LocalStack can be run in separate processes. In moto, components are often hard-wired in memory (e.g., when forwarding a message on an SNS topic to an SQS queue, the queue endpoint is looked up in a local hash map). In contrast, LocalStack services live in isolation (separate processes/threads communicating via HTTP), which fosters true decoupling and more closely resembles the real cloud environment.
- **Pluggable services**: All services in LocalStack are easily pluggable (and replaceable), due to the fact that we are using isolated processes for each service. This allows us to keep the framework up-to-date and select best-of-breed mocks for each individual service.

Requirements

- `python` (both Python 2.x and 3.x supported)
- `pip` (python package manager)
- `Docker`

Installing

The easiest way to install LocalStack is via `pip`:

```
pip install localstack
```

Note: Please do **not** use `sudo` or the `root` user - LocalStack should be installed and started entirely under a local non-root user. If you have problems with permissions in MacOS X Sierra, install with `pip install --user localstack`

Running

By default, LocalStack gets started inside a Docker container using this command:

```
localstack start
```

(Note that on MacOS you may have to run `TMPDIR=/private$TMPDIR localstack start --docker` if `$TMPDIR` contains a symbolic link that cannot be mounted by Docker.)

Note: From 2020-07-11 onwards, the default image `localstack/localstack` in Docker Hub refers to the "light version", which has some large dependency files like Elasticsearch removed (and lazily downloads them, if required). (Note that the `localstack/localstack-light` image alias may get removed in the future). In case you need the full set of dependencies, the `localstack/localstack-full` image can be used instead. Please also refer to the `USE_LIGHT_IMAGE` configuration below.

Note: By default, LocalStack uses the image tagged `latest` that is cached on your machine, and will **not** pull the latest image automatically from Docker Hub (i.e., the image needs to be pulled manually if needed).

(**Note**: Although it is strongly recommended to use Docker, the infrastructure can also be spun up directly on the host machine using the `--host` startup flag. Note that this will require additional dependencies, and is not supported on some operating systems, including Windows.)

Using `docker`

You can also use docker directly and use the following command to get started with localstack

```
docker run --rm -p 4566:4566 -p 4571:4571 localstack/localstack
```

to run a throw-away container without any external volumes. To start a subset of services use `-e "SERVICES=dynamodb,s3"`.

Using `docker-compose`

You can also use the `docker-compose.yml` file from the repository and use this command (currently requires `docker-compose` version 1.9.0+):

```
docker-compose up
```

(Note that on MacOS you may have to run `TMPDIR=/private$TMPDIR docker-compose up` if `$TMPDIR` contains a symbolic link that cannot be mounted by Docker.)

To facilitate interoperability, configuration variables can be prefixed with `LOCALSTACK_` in docker. For instance, setting `LOCALSTACK_SERVICES=s3` is equivalent to `SERVICES=s3`.

Using Helm

You can use Helm to install LocalStack in a Kubernetes cluster by running these commands (the Helm charts are maintained in this repo):

```
helm repo add localstack-repo http://helm.localstack.cloud

helm upgrade --install localstack localstack-repo/localstack
```

Configurations

You can pass the following environment variables to LocalStack:

- `EDGE_PORT`: Port number for the edge service, the main entry point for all API invocations (default: `4566`).
- `SERVICES`: Comma-separated list of service names (APIs) to start up. Service names basically correspond to the service names of the AWS CLI (`kinesis`, `lambda`, `sqs`, etc), although LocalStack only supports a subset of them. Example value: `kinesis,lambda,sqs` to start Kinesis, Lambda, and SQS. In addition, the following shorthand values can be specified to run a predefined ensemble of services:
  - `serverless`: run services often used for Serverless apps (`iam`, `lambda`, `dynamodb`, `apigateway`, `s3`, `sns`)
- `DEFAULT_REGION`: AWS region to use when talking to the API (default: `us-east-1`).
- `HOSTNAME`: Name of the host to expose the services internally (default: `localhost`). Use this to customize the framework-internal communication, e.g., if services are started in different containers using docker-compose.
- `HOSTNAME_EXTERNAL`: Name of the host to expose the services externally (default: `localhost`). This host is used, e.g., when returning queue URLs from the SQS service to the client.

- `<SERVICE>_PORT_EXTERNAL`: Port number to expose a specific service externally (defaults to service ports above). `SQS_PORT_EXTERNAL`, for example, is used when returning queue URLs from the SQS service to the client.
- `IMAGE_NAME`: Specific name and tag of LocalStack Docker image to use, e.g., `localstack/localstack:0.11.0` (default: `localstack/localstack`).
- `USE_LIGHT_IMAGE`: Whether to use the light-weight Docker image (default: `1`). Overwritten by `IMAGE_NAME`.
- `KINESIS_ERROR_PROBABILITY`: Decimal value between 0.0 (default) and 1.0 to randomly inject `ProvisionedThroughputExceededException` errors into Kinesis API responses.
- `KINESIS_SHARD_LIMIT`: Integer value (default: `100`) or `Infinity` (to disable), causing the Kinesis API to start throwing exceptions to mimick the default shard limit.
- `KINESIS_LATENCY`: Integer value (default: `500`) or `0` (to disable), causing the Kinesis API to delay returning a response in order to mimick latency from a live AWS call.
- `DYNAMODB_ERROR_PROBABILITY`: Decimal value between 0.0 (default) and 1.0 to randomly inject `ProvisionedThroughputExceededException` errors into DynamoDB API responses.
- `DYNAMODB_HEAP_SIZE`: Sets the JAVA EE maximum memory size for dynamodb values are (integer)m for MB, (integer)G for GB default(256m), full table scans require more memory
- `STEPFUNCTIONS_LAMBDA_ENDPOINT`: URL to use as the Lambda service endpoint in Step Functions. By default this is the LocalStack Lambda endpoint. Use `default` to select the original AWS Lambda endpoint.
- `LAMBDA_EXECUTOR`: Method to use for executing Lambda functions. Possible values are:
  - `local`: run Lambda functions in a temporary directory on the local machine
  - `docker`: run each function invocation in a separate Docker container
  - `docker-reuse`: create one Docker container per function and reuse it across invocations

For `docker` and `docker-reuse`, if LocalStack itself is started inside Docker, then the `docker` command needs to be available inside the container (usually requires to run the container in privileged mode). Default is `docker`, fallback to `local` if Docker is not available.

- `LAMBDA_REMOTE_DOCKER` determines whether Lambda code is copied or mounted into containers. Possible values are:
    - `true` (default): your Lambda function definitions will be passed to the container by copying the zip file (potentially slower). It allows for remote execution, where the host and the client are not on the same machine.
    - `false`: your Lambda function definitions will be passed to the container by mounting a volume (potentially faster). This requires to have the Docker client and the Docker host on the same machine. Also, `HOST_TMP_FOLDER` must be set properly, and a volume mount like `${HOST_TMP_FOLDER}:/tmp/localstack` needs to be configured if you're using docker-compose.
- `LAMBDA_DOCKER_NETWORK`: Optional Docker network for the container running your lambda function.
- `LAMBDA_DOCKER_DNS`: Optional DNS server for the container running your lambda function.
- `LAMBDA_CONTAINER_REGISTRY` Use an alternative docker registry to pull lambda execution containers (default: `lambci/lambda`).
- `LAMBDA_REMOVE_CONTAINERS`: Whether to remove containers after Lambdas finished executing (default: `true`).
- `TMPDIR`: Temporary folder inside the LocalStack container (default: `/tmp`).
- `HOST_TMP_FOLDER`: Temporary folder on the host that gets mounted as `$TMPDIR/localstack` into the LocalStack container. Required only for Lambda volume mounts when using `LAMBDA_REMOTE_DOCKER=false`.
- `DATA_DIR`: Local directory for saving persistent data (currently only supported for these services: Kinesis, DynamoDB, Elasticsearch, S3, Secretsmanager, SSM, SQS, SNS). Set it to `/tmp/localstack/data` to enable persistence (`/tmp/localstack` is mounted into the Docker container), leave blank to disable persistence (default).
- `PORT_WEB_UI`: Port for the Web user interface / dashboard (default: `8080`). Note that the Web UI is now deprecated, and requires to use the `localstack/localstack-full` Docker image.
-

`<SERVICE>_BACKEND`: Custom endpoint URL to use for a specific service, where `<SERVICE>` is the uppercase service name (currently works for: `APIGATEWAY`, `CLOUDFORMATION`, `DYNAMODB`, `ELASTICSEARCH`, `KINESIS`, `S3`, `SNS`, `SQS`). This allows to easily integrate third-party services into LocalStack.

- `FORCE_NONINTERACTIVE`: when running with Docker, disables the `--interactive` and `--tty` flags. Useful when running headless.
- `DOCKER_FLAGS`: Allows to pass custom flags (e.g., volume mounts) to "docker run" when running LocalStack in Docker.
- `DOCKER_CMD`: Shell command used to run Docker containers, e.g., set to `"sudo docker"` to run as sudo (default: `docker`).
- `SKIP_INFRA_DOWNLOADS`: Whether to skip downloading additional infrastructure components (e.g., specific Elasticsearch versions).
- `START_WEB`: Flag to control whether the Web UI should be started in Docker (values: `0` / `1`; default: `1`).
- `LAMBDA_FALLBACK_URL`: Fallback URL to use when a non-existing Lambda is invoked. Either records invocations in DynamoDB (value `dynamodb://<table_name>`) or forwards invocations as a POST request (value `http(s)://...`).
- `LAMBDA_FORWARD_URL`: URL used to forward all Lambda invocations (useful to run Lambdas via an external service).
- `EXTRA_CORS_ALLOWED_HEADERS`: Comma-separated list of header names to be be added to `Access-Control-Allow-Headers` CORS header
- `EXTRA_CORS_EXPOSE_HEADERS`: Comma-separated list of header names to be be added to `Access-Control-Expose-Headers` CORS header
- `LAMBDA_JAVA_OPTS`: Allow passing custom JVM options (e.g., `-Xmx512M`) to Java Lambdas executed in Docker. Use `_debug_port_` placeholder to configure the debug port (e.g., `-agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=_debug_port_`).
- `MAIN_CONTAINER_NAME`: Specify the main docker container name (default: `localstack_main`).
- `INIT_SCRIPTS_PATH`: Specify the path to the initializing files with extensions .sh that are found default in `/docker-entrypoint-initaws.d`.

- `DEBUG`: For troubleshooting LocalStack start issues
- `LS_LOG`: Specify the log level('debug', 'info', 'warn', 'error', 'warning') currently overrides the `DEBUG` configuration.

The following environment configurations are *deprecated*:

- `USE_SSL`: Whether to use `https://...` URLs with SSL encryption (default: `false`). Deprecated as of version 0.11.3 - each service endpoint now supports multiplexing HTTP/HTTPS traffic over the same port.

Additionally, the following *read-only* environment variables are available:

- `LOCALSTACK_HOSTNAME`: Name of the host where LocalStack services are available. Use this hostname as endpoint (e.g., `http://${LOCALSTACK_HOSTNAME}:4566`) in order to **access the services from within your Lambda functions** (e.g., to store an item to DynamoDB or S3 from a Lambda).

An example passing the above environment variables to LocalStack to start Kinesis, Lambda, Dynamodb and SQS:

```
SERVICES=kinesis,lambda,sqs,dynamodb localstack start
```

Verifying your docker-compose configuration using the command line

You can use `localstack config validate` command to validate for common mis-configuration.

By `default` it validates `docker-compose.yml`. You can override it using `--file` argument.

For example

```
localstack config validate --file=localstack-docker-compose.yml
```

Dynamically updating configuration at runtime

Each of the service APIs listed above defines a backdoor API under the path `/?_config_` which allows to dynamically update configuration variables defined in `config.py`.

For example, to dynamically set `KINESIS_ERROR_PROBABILITY=1` at runtime, use the following command:

```
curl -v -d '{"variable":"KINESIS_ERROR_PROBABILITY","value":1}'
'http://localhost:4566/?_config_'
```

Service health checks

The service `/health` check endpoint on the edge port (`http://localhost:4566/health` by default) provides basic information about the status of each service (e.g., `{"s3":"running","es":"starting"}`). By default, the endpoint returns cached values that are determined during startup - the status values can be refreshed by adding the `reload` query parameter: `http://localhost:4566/health?reload` .

Initializing a fresh instance

When a container is started for the first time, it will execute files with extensions .sh that are found in `/docker-entrypoint-initaws.d` or an alternate path defined in `INIT_SCRIPTS_PATH` . Files will be executed in alphabetical order. You can easily create aws resources on localstack using `awslocal` (or `aws`) cli tool in the initialization scripts.

Using custom SSL certificates

To use your own SSL certificate instead of the randomly generated certificate, you can place a file `server.test.pem` into the LocalStack temporary directory (`$TMPDIR/localstack` , or `/tmp/localstack` by default). The file `server.test.pem` must contain the key file, as well as the certificate file content:

```
-----BEGIN PRIVATE KEY-----
...
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----
```

Using custom SSL certificates with docker-compose

Typically with docker-compose you can add into docker-compose.yml this volume to the LocalStack services :

```
  volumes:
    - "${PWD}/ls_tmp:/tmp/localstack"
    - "/var/run/docker.sock:/var/run/docker.sock"
```

The local directory `/ls_tmp` must contains the three files (server.test.pem, server.test.pem.crt, server.test.pem.key)

Accessing the infrastructure via CLI or code

You can point your `aws` CLI to use the local infrastructure, for example:

```
aws --endpoint-url=http://localhost:4566 kinesis list-streams
{
    "StreamNames": []
}
```

Use the below command to install `aws CLI`, if not installed already.

```
pip install awscli
```

Setting up local region and credentials to run LocalStack

aws requires the region and the credentials to be set in order to run the aws commands. Create the default configuration & the credentials. Below key will ask for the Access key id, secret Access Key, region & output format.

```
aws configure --profile default

# Config & credential file will be created under ~/.aws folder
```

NOTE: Please use `test` as Access key id and secret Access Key to make S3 presign url work. We have added presign url signature verification algorithm to validate the presign url and its expiration. You can configure credentials into the system environment using `export` command in the linux/Mac system. You also can add credentials in `~/.aws/credentials` file directly.

```
export AWS_ACCESS_KEY_ID=test
export AWS_SECRET_ACCESS_KEY=test
```

NEW: Check out awslocal, a thin CLI wrapper that runs commands directly against LocalStack (no need to specify `--endpoint-url` anymore). Install it via `pip install awscli-local`, and then use it as follows:

```
awslocal kinesis list-streams
{
    "StreamNames": []
}
```

UPDATE: Use the environment variable `$LOCALSTACK_HOSTNAME` to determine the target host inside your Lambda function. See Configurations section for more details.

Using the official AWS CLI version 2 Docker image with Localstack Docker container

By default the container running amazon/aws-cli is isolated from `0.0.0.0:4566` on the host machine, that means that aws-cli cannot reach localstack through your shell.

To ensure that the two docker containers can communicate create a network on the docker engine:

```
$ ▶ docker network create localstack
0c9cb3d37b0ea1bfeb6b77ade0ce5525e33c7929d69f49c3e5ed0af457bdf123
```

Then modify the `docker-compose.yml` specifying the network to use:

```
networks:
  default:
    external:
      name: "localstack"
```

Run AWS Cli v2 docker container using this network (example):

```
$ ▶ docker run --network localstack --rm -it amazon/aws-cli --endpoint-u
{
    "Functions": []
}
```

If you use AWS CLI v2 from a docker container often, create an alias:

```
$ ▶ alias laws='docker run --network localstack --rm -it amazon/aws-cli
```

So you can type:

```
$ ▶ laws lambda list-functions
{
    "Functions": []
}
```

Client Libraries

- Python: https://github.com/localstack/localstack-python-client
  - alternatively, you can also use `boto3` and use the `endpoint_url` parameter when creating a connection
- .NET: https://github.com/localstack-dotnet/localstack-dotnet-client
  - alternatively, you can also use `AWS SDK for .NET` and change `ClientConfig` properties when creating a service client.
- (more coming soon...)

Invoking API Gateway

While API Gateway endpoints on AWS use a custom DNS name to identify the API ID (e.g., `https://nmafetnwf6.execute-api.us-east-1.amazonaws.com/prod/my/path`), LocalStack uses the special URL path indicator `.../_user_request_/...` to indicate the execution of a REST API method.

The URL pattern for API Gateway executions is `http://localhost:4566/restapis/<apiId>/<stage>/_user_request_/<methodPath>`. The example URL above would map to the following `localhost` URL:

```
$ curl http://localhost:4566/restapis/nmafetnwf6/prod/_user_request_/my/path
```

Integration with nosetests

If you want to use LocalStack in your integration tests (e.g., nosetests), simply fire up the infrastructure in your test setup method and then clean up everything in your teardown method:

```
from localstack.services import infra
```

```
def setup():
    infra.start_infra(asynchronous=True)

def teardown():
    infra.stop_infra()

def my_app_test():
    # here goes your test logic
```

See the example test file `tests/integration/test_integration.py` for more details.

Integration with Serverless

You can use the `serverless-localstack` plugin to easily run Serverless applications on LocalStack. For more information, please check out the plugin repository here: https://github.com/localstack/serverless-localstack

Integration with Terraform

You can use Terraform to provision your resources locally. Please refer to the Terraform AWS Provider docs here on how to configure the API endpoints on `localhost`.

Using local code with Lambda

In order to mount a local folder, ensure that `LAMBDA_REMOTE_DOCKER` is set to `false` then set the S3 bucket name to `__local__` and the S3 key to your local path:

```
awslocal lambda create-function --function-name myLambda \
    --code S3Bucket="__local__",S3Key="/my/local/lambda/folder" \
    --handler index.myHandler \
    --runtime nodejs8.10 \
    --role whatever
```

Note: When using `LAMBDA_REMOTE_DOCKER=false`, make sure to properly set the `HOST_TMP_FOLDER` environment variable for the LocalStack container (see Configuration section above).

Integration with Java/JUnit

In order to use LocalStack with Java, the project ships with a simple JUnit runner, see sample below.

```
...
import cloud.localstack.LocalstackTestRunner;
import cloud.localstack.TestUtils;
import cloud.localstack.docker.annotation.LocalstackDockerProperties;

@RunWith(LocalstackTestRunner.class)
@LocalstackDockerProperties(services = { "s3", "sqs", "kinesis:77077"
})
public class MyCloudAppTest {

  @Test
  public void testLocalS3API() {
    AmazonS3 s3 = TestUtils.getClientS3()
    List<Bucket> buckets = s3.listBuckets();
    ...
  }
}
```

For more details and a complete list of configuration parameters, please refer to the LocalStack Java Utils repository.

Troubleshooting

- If you're using AWS Java libraries with Kinesis, please, refer to CBOR protocol issues with the Java SDK guide how to disable CBOR protocol which is not supported by kinesalite.
- Accessing local S3: To avoid domain name resolution issues, you need to enable **path style access** on your S3 SDK client. Most AWS SDKs provide a config to achieve that, e.g., for Java:

```
s3.setS3ClientOptions(S3ClientOptions.builder().setPathStyleAccess(true)
// There is also an option to do this if you're using any of the
client builder classes:
AmazonS3ClientBuilder builder = AmazonS3ClientBuilder.standard();
builder.withPathStyleAccessEnabled(true);
...
```

- Mounting the temp. directory: Note that on MacOS you may have to run `TMPDIR=/private$TMPDIR docker-compose up` if `$TMPDIR` contains a symbolic link that cannot be mounted by Docker. (See details here: https://bitbucket.org/atlassian/localstack/issues/40/getting-mounts-failed-on-docker-compose-up)
-

If you're seeing Lambda errors like `Cannot find module ...` when using `LAMBDA_REMOTE_DOCKER=false`, make sure to properly set the `HOST_TMP_FOLDER` environment variable and mount the temporary folder from the host into the LocalStack container.

- If you run into file permission issues on `pip install` under Mac OS (e.g., `Permission denied: '/Library/Python/2.7/site-packages/six.py'`), then you may have to re-install `pip` via Homebrew (see this discussion thread). Alternatively, try installing with the `--user` flag: `pip install --user localstack`
- If you are deploying within OpenShift, please be aware: the pod must run as `root`, and the user must have capabilities added to the running pod, in order to allow Elasticsearch to be run as the non-root `localstack` user.
- If you are experiencing slow performance with Lambdas in Mac OS, you could either (1) try mounting local code directly into the Lambda container, or (2) disable mounting the temporary directory into the LocalStack container in docker-compose. (See also https://github.com/localstack/localstack/issues/2515)
- The environment variable `no_proxy` is rewritten by LocalStack. (Internal requests will go straight via localhost, bypassing any proxy configuration).
- For troubleshooting LocalStack start issues, you can check debug logs by running `DEBUG=1 localstack start`
- In case you get errors related to node/nodejs, you may find (this issue comment: https://github.com/localstack/localstack/issues/227#issuecomment-319938530) helpful.
- If you are using AWS Java libraries and need to disable SSL certificate checking, add `-Dcom.amazonaws.sdk.disableCertChecking` to the java invocation.

Developing

Requirements for developing or starting locally

To develop new features, or to start the stack locally (outside of Docker), the following additional tools are required:

- `make`
- `npm` (node.js package manager)

- `java` / `javac` (Java 8 runtime environment and compiler)
- `mvn` (Maven, the build system for Java)
- `moto` (for testing)
- `docker-compose` (for running the localstack using docker-compose)
- `mock` (for unit testing)
- `pytest` (for unit testing)
- `pytest-cov` (to check the unit-testing coverage)

Development Environment

If you pull the repo in order to extend/modify LocalStack, run this command to install all the dependencies:

```
make install
```

This will install the required pip dependencies in a local Python virtualenv directory `.venv` (your global python packages will remain untouched), as well as some node modules in `./localstack/node_modules/`. Depending on your system, some pip/npm modules may require additional native libs installed.

The Makefile contains a target to conveniently run the local infrastructure for development:

```
make infra
```

Starting LocalStack using Vagrant (Centos 8)

This is similar to `make docker-mount-run`, but instead of docker centos VM will be started and source code will be mounted inside.

Pre-requirements

- Vagrant
- `vagrant plugin install vagrant-vbguest`

Starting Vagrant

- `make vagrant-start` (be ready to provide system password)

Using Vagrant

- `vagrant ssh`

- `sudo -s`
- `cd /localstack`
- `SERVICES=dynamodb DEBUG=1 make docker-mount-run`

Stopping Vagrant

- `make vagrant-stop` or `vagrant halt`

Deleting Vagrant VM

- `vagrant destroy`

Check out the developer guide which contains a few instructions on how to get started with developing (and debugging) features for LocalStack.

Testing

The project contains a set of unit and integration tests that can be kicked off via a make target:

```
make test
```

To check the Code Coverage

Once the new feature / bug fix is done, run the unit testing and check for the coverage.

```
# To run the particular test file (sample)
pytest --cov=localstack tests/unit/test_common.py

# To check the coverage in the console
coverage report

# To check the coverage as html (output will be redirected to the
html folder)
coverage html
```

Web Dashboard (deprecated)

The projects also comes with a simple Web dashboard that allows to view the deployed AWS components and the relationship between them.

```
localstack web
```

Please note that the Web UI requires using the extended version of the Docker image (`localstack/localstack-full`).

Note: The Web dashboard is not actively maintained anymore and may get removed in an upcoming release.

Other UI Clients

- Commandeer desktop app
- DynamoDB Admin Web UI

Change Log

Please refer to `CHANGELOG.md` to see the complete list of changes for each release.

Contributing

We welcome feedback, bug reports, and pull requests!

For pull requests, please stick to the following guidelines:

- Add tests for any new features and bug fixes. Ideally, each PR should increase the test coverage.
- Follow the existing code style (e.g., indents). A PEP8 code linting target is included in the Makefile.
- Put a reasonable amount of comments into the code.
- Fork localstack on your github user account, do your changes there and then create a PR against main localstack repository.
- Separate unrelated changes into multiple pull requests.
- 1 commit per PR: Please squash/rebase multiple commits into one single commit (to keep the history clean).

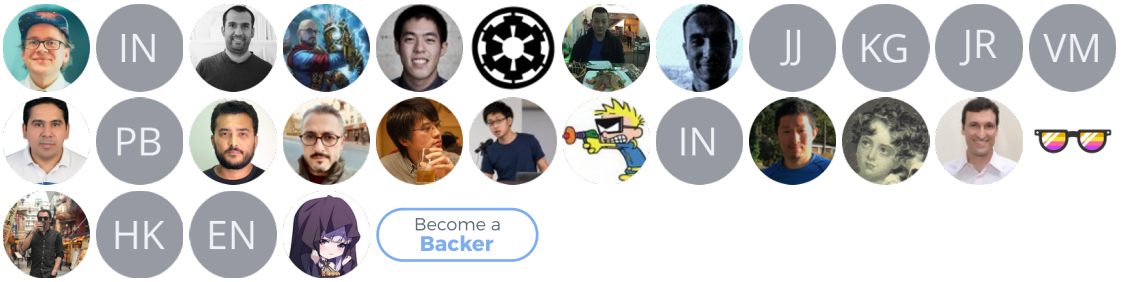
Please note that by contributing any code or documentation to this repository (by raising pull requests, or otherwise) you explicitly agree to the **Contributor License Agreement**.

Contributors

This project exists thanks to all the people who contribute.

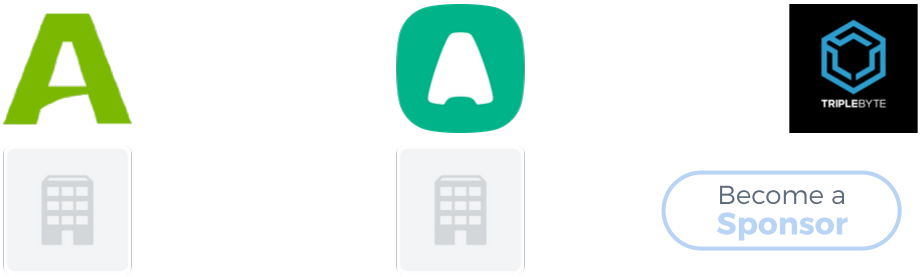
Backers

Thank you to all our backers! 🙏 [Become a backer]

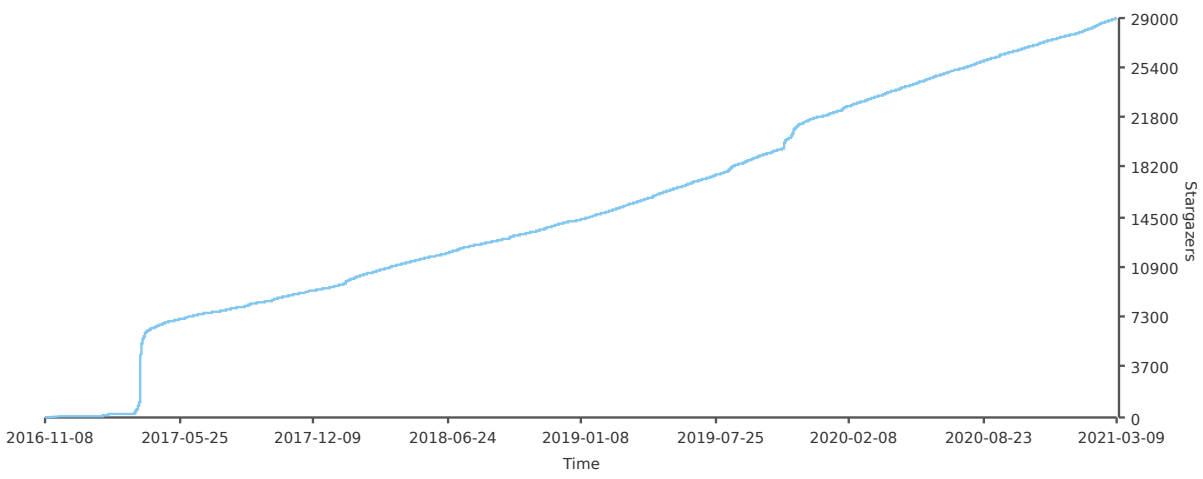


Sponsors

Support this project by becoming a sponsor. Your logo will show up here with a link to your website. [Become a sponsor]



Stargazers over time



License

This version of LocalStack is released under the Apache License, Version 2.0 (see LICENSE.txt). By downloading and using this software you agree to the End-User License Agreement (EULA).

We build on a number of third-party software tools, including the following:

Third-Party software	License
Python/pip modules:	
airspeed	BSD License
amazon_kclpy	Amazon Software License
boto3	Apache License 2.0
coverage	Apache License 2.0
docopt	MIT License
elasticsearch	Apache License 2.0
flask	BSD License
flask_swagger	MIT License
jsonpath-rw	Apache License 2.0
moto	Apache License 2.0
requests	Apache License 2.0
subprocess32	PSF License
Node.js/npm modules:	
kinesalite	MIT License
Other tools:	
Elasticsearch	Apache License 2.0
local-kms	MIT License