Cognito

Get started with AWS Cognito on LocalStack

6 minute read

The **AWS Cognito** service enables you to manage authentication and access control for AWS-backed apps and resources.

LocalStack Pro contains basic support for authentication via Cognito. You can create Cognito user pools, sign up and confirm users, set up Lambda triggers, and use the `COGNITO_USER_POOLS` authorizer integration with API Gateway.

Note

By default, local Cognito does not send actual email messages. To enable this feature, you will require an email address and the corresponding SMTP settings. Please refer to the **Configuration** guide for instructions on how to configure the connection parameters of your SMTP server.

User pools and basic authentication flows

The following subsections illustrate how you can create a user pool and client, and then sign up and authenticate a new user in the pool.

Creating a User Pool

Just as with AWS, you can create a user pool in LocalStack with the following command:

```
$ awslocal cognito-idp create-user-pool --pool-name test
```

The response should look similar to this:

```
"UserPool": {
        "Id": "us-east-1_fd924693e9b04f549f989283123a29c2",
        "Name": "test",
        "Policies": {
            "PasswordPolicy": {
                "MinimumLength": 8,
                "RequireUppercase": true,
                "RequireLowercase": true,
                "RequireNumbers": true,
                "RequireSymbols": true,
                "TemporaryPasswordValidityDays": 7
            }
        },
        "LastModifiedDate": "2021-10-06T11:57:21.883Z",
        "CreationDate": "2021-10-06T11:57:21.883Z",
        "SchemaAttributes": [],
        "VerificationMessageTemplate": {
            "DefaultEmailOption": "CONFIRM_WITH_CODE"
        },
        "EmailConfiguration": {
            "EmailSendingAccount": "COGNITO_DEFAULT"
        },
        "AdminCreateUserConfig": {
            "AllowAdminCreateUserOnly": false
        },
        "Arn": "arn:aws:cognito-idp:us-east-1:000000000000:userpool/us-east-1_fd924
}
```

We will need the user pool's `id` for further operations, so save it in a `pool_id` variable:

```
$ pool_id=<your-pool-id>
```

Alternatively, you can also use a JSON processor like `jq` to directly extract the necessary information when creating a pool in the first place:

```
$ pool_id=$(awslocal cognito-idp create-user-pool --pool-name test | jq -rc ".UserP
```

Adding a Client

Now we add a client to our newly created pool. Again, we will also need the ID of the created client for the next step. The complete command for client creation with subsequent ID extraction is therefore:

```
$ client_id=$(awslocal cognito-idp create-user-pool-client --user-pool-id $pool_id
```

Using predefined IDs on pool creation

It is possible to use a predefined ID when creating Cognito user or identity pools, by setting the tag `_custom_id_`. This can be helpful when testing auth flows with LocalStack frequently being restarted and resourced re-created. For example:

```
$ awslocal cognito-idp create-user-pool --pool-name p1 --user-pool-tags "_custom_ic
{
    "UserPool": {
        "Id": "myid123",
        "Name": "p1",
    ...
```

Signing up and confirming a user

With these steps already taken, we can now sign up a user:

```
$ awslocal cognito-idp sign-up --client-id $client_id --username example_user --pas
```

The response should look similar to this:

```
{
    "UserConfirmed": false,
    "UserSub": "5fdbe1d5-7901-4fee-9d1d-518103789c94"
}
```

After the user is created, a confirmation code is generated. The code is printed in the LocalStack container logs (see below), and can optionally also be sent via email if you have **SMTP configured**.

```
INFO:localstack_ext.services.cognito.cognito_idp_api: Confirmation code for Cognito
DEBUG:localstack_ext.bootstrap.email_utils: Sending confirmation code via email to
```

We can confirm the user with the activation code, using the following command:

```
$ awslocal cognito-idp confirm-sign-up --client-id $client_id --username example_us
```

As the above command doesn't return an answer, we check the pool to see that the request was successful:

```
$ awslocal cognito-idp list-users --user-pool-id $pool_id
{
    "Users": [
        {
            "Username": "example_user",
            "Attributes": [
                {
                    "Name": "email",
                    "Value": "your.email@address.com"
                },
                {
                    "Name": "sub",
                    "Value": "5fdbe1d5-7901-4fee-9d1d-518103789c94"
                },
                {
                    "Name": "cognito:username",
                    "Value": "example_user"
                }
            ],
            "Enabled": true,
            "UserStatus": "CONFIRMED"
        }
    ]
}
```

JWT token issuer and JSON Web Key Sets (JWKS) endpoints

The JWT tokens created by Cognito contain an issuer (`iss`) attribute that represents the endpoint of the corresponding user pool. The issuer endpoint generally follows this pattern, where `<pool_id>` is the ID of the Cognito user pool:

```
http://localhost:4566/<pool_id>
```

Under certain circumstances (depending on your configurations), there may be slight nuances of the issuer URL, like:

```
https://cognito-idp.localhost.localstack.cloud/<pool_id>
```

You can access the JSON Web Key Sets (JWKS) configuration under the following standardized well-known URL for each user pool:

```
$ curl 'http://localhost:4566/<pool_id>/.well-known/jwks.json'
{"keys": [{"kty": "RSA", "alg": "RS256", "use": "sig", "kid": "test-key", "n": "k6l
```

Additionally, the global region-specific public keys for Cognito Identity Pools can be retrieved under this endpoint:

```
$ curl http://localhost:4566/.well-known/jwks_uri
{"keys": [{"kty": "RSA", "alg": "RS512", "use": "sig", "kid": "ap-northeast-11", "r
```

Cognito Lambda Triggers

Cognito provides a number of lifecycle hooks in the form of Cognito Lambda triggers. These triggers can be used to react to various lifecycle events and customize the behavior of user signup, confirmation, migration, etc.

For example, to define a *user migration* Lambda trigger, we can first create a Lambda function (say, named `"f1"`) capable of performing the migration, and then define the corresponding `--lambda-config` on the user pool creation:

```
$ awslocal cognito-idp create-user-pool --pool-name test2 --lambda-config '{"UserMi
```

Upon authentication of a non-registered user, Cognito will then automatically call the migration Lambda function and finally add the migrated user to the pool.

More details on Cognito Lambda triggers can be found in the **AWS documentation**.

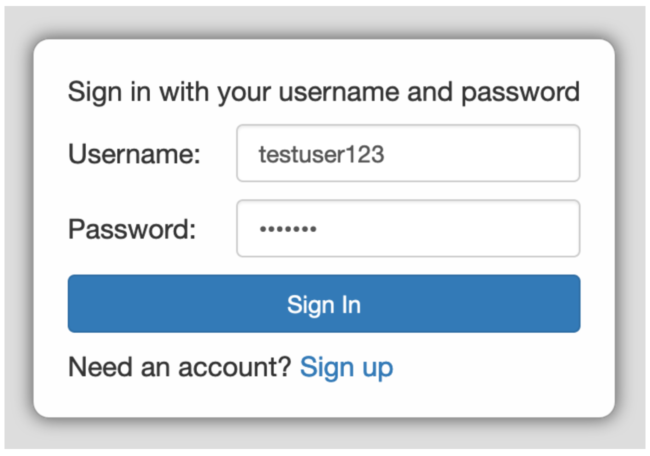
OAuth Flows via Cognito Login Form

You can also access the local **Cognito login form** by entering the following URL in your browser:

```
https://localhost.localstack.cloud/_aws/cognito-idp/login?response_type=code&client
```

Please replace `<client_id>` with the ID of an existing user pool client ID (in this case, `example_user`), and `<redirect_uri>` with the redirect URI of your application (e.g., `http://example.com`).

The login form should look similar to the screenshot below:


Sign in with your username and password

Username: testuser123

Password: •••••••

After successful login, the page will redirect to the specified `<redirect_uri>`, with a path parameter `?code=<code>` appended, e.g., `http://example.com?code=test123`. Obtain a token by submitting that code with `grant_type=authorization_code` the LocalStack's implementation of the Cognito OAuth2 TOKEN Endpoint documented **here**. Note that the value of the `redirect_uri` parameter must match the value provided during login.

```
% curl \
  --data-urlencode 'grant_type=authorization_code' \
  --data-urlencode 'redirect_uri=http://example.com' \
  --data-urlencode "client_id=${client_id}" \
  --data-urlencode 'code=test123' \
  'http://localhost:4566/_aws/cognito-idp/oauth2/token'
{"access_token": "eyJ0eXAi…lKaHx44Q", "expires_in": 86400, "token_type": "Bearer",
```

Serverless and Cognito

You can also use Cognito and LocalStack in conjunction with the **Serverless framework**.

For example, take this snippet of a `serverless.yml` configuration:

```
service: test

plugins:
  - serverless-deployment-bucket
  - serverless-pseudo-parameters
  - serverless-localstack

custom:
  localstack:
    stages: [local]

functions:
  http_request:
    handler: http.request
    events:
      - http:
          path: v1/request
          authorizer:
            arn: arn:aws:cognito-idp:us-east-1:#{AWS::AccountId}:userpool/ExampleUs

resources:
  Resources:
    UserPool:
      Type: AWS::Cognito::UserPool
      Properties:
        ...
```

The serverless configuration can then be deployed using `serverless deploy --stage local`. The example contains a Lambda function `http_request` which is connected to an API Gateway endpoint. Once deployed, the `v1/request` API Gateway endpoint will be secured against the Cognito user pool "`ExampleUserPool`". You can then register users against that local pool, using the same API calls as for AWS.

In order to make requests against the secured API Gateway endpoint, use the local Cognito API to retrieve identity credentials which can be sent along as `Authentication` HTTP headers (where `test-1234567` is the name of the access key ID generated by Cognito):

```
Authentication: AWS4-HMAC-SHA256 Credential=test-1234567/20190821/us-east-1/cognitc
```

Further reading

For a more detailed example, please check out our **sample repository**.

Last modified April 26, 2023: **add docs for `EKS_K3S_IMAGE_TAG` and `_custom_id_` in Cognito (#600) (03ccd91f3)**