Disposable Cloud Environment (DCE) Documentation

Optum

Apr 30, 2021

Contents

1 Home **1**
1.1 Disposable Cloud Environment (DCE) . . . 1

2 Quickstart **3**
2.1 Quickstart . . . 3

3 How To **5**
3.1 How To . . . 5

4 Concepts **21**
4.1 Concepts . . . 21

5 DCE IAM Policies **25**
5.1 DCE IAM Policies . . . 25

6 Deploy with Terraform **31**
6.1 Deploy with Terraform . . . 31

7 API Reference **33**
7.1 API Reference . . . 33

8 API Auth **41**
8.1 API Auth . . . 41

9 SNS Lifecycle Events **49**
9.1 SNS Lifecycle Events . . . 49

10 Permissions and Policies **53**
10.1 Policies and Permissions . . . 53

11 Account Cleanup with AWS Nuke **57**
11.1 Account Cleanup with AWS Nuke . . . 57

12 Local Development **65**
12.1 Local Development . . . 65

CHAPTER 1

Home

1.1 Disposable Cloud Environment (DCE)

The Disposable Cloud Environment (DCE) provides temporary, limited Amazon Web Services (AWS) accounts. Accounts can be "leased" for a period of time or up to a pre-determined budget amount. When the period of time is reached or the maximum budgeted amount is exceeded, the lease is expired. The leased account is *reset* and returned to a pool of accounts to be leased again.

At a high-level, DCE consists of AWS Lambda functions (implemented in Go), Amazon DynamoDB tables, Amazon Simple Notification Service (SNS) topics, and APIs exposed with Amazon API Gateway. These resources are created in the AWS account using Hashicorp Terraform.

1.1.1 Why DCE?

DCE is your playground in the cloud

With a DCE account, you have a safe environment to experiment with in the cloud. With near-administrative access to an AWS account, you can click around the AWS web console or run AWS CLI commands from your terminal. As a developer in an organization, you don't need to worry about cost management or orphaned resources.

Budget limits

DCE can be configured to expire account *leases* (see *Concepts documentation*) based on a budgeted amount for usage.

Once the account hits a weekly spending limit, the account will be automatically wiped clean so there are no surprise bills at the end of the month.

Timed leases

DCE contains the concept of *expiring leases*. A *lease* represents temporary access to an account for a certain amount of time. Once the lease is expired, the AWS account is cleaned of all of the resources and returned to the pool for other people to lease.

1.1.2 Getting started

To get started using DCE, see the *quickstart*.

1.1.3 Viewing the source

The source code for DCE can be found on GitHub.

CHAPTER 2

Quickstart

2.1 Quickstart

Deploy DCE and lease an account quickly using the DCE CLI.

1. Download the appropriate executable for your OS from the latest release. e.g. for mac, you should download dce_darwin_amd64.zip
2. Unzip and move the executable to a directory on your PATH, e.g.

```
# Download the zip file
curl -L -o dce_darwin_amd64.zip https://github.com/Optum/dce-cli/releases/latest/
↪download/dce_darwin_amd64.zip

# Unzip to a directory on your path
unzip dce_darwin_amd64.zip -d /usr/local/bin
```

3. Type `dce init`. Leave all fields blank for now.
4. Deploy DCE using IAM Credentials that have AdministratorAccess

```
export AWS_ACCESS_KEY_ID=XXXXXXXXXX
export AWS_SECRET_ACCESS_KEY=XXXXXXXXXXXXXXXXXXXX
dce system deploy
```

5. Retrieve the DCE API url from API Gateway in your master account, and add it to the dce config file, e.g.

```
api:
    host: abcdefghij.execute-api.us-east-1.amazonaws.com
    basepath: /api
region: us-east-1
```

6. Prepare a second AWS account to be your first "DCE Child Account".
    - Create an IAM role with `AdministratorAccess` and a trust relationship to your DCE Master Accounts

- Create an account alias in the IAM dashboard or using the AWS CLI command

```
aws iam create-account-alias --account-alias examplealias
```

7. Add your child account to the accounts pool

```
dce accounts add --account-id <child-account-id> --admin-role-arn <child-account-cross-account-r
```

8. Wait until the child account `accountStatus` is `Ready`

```
dce accounts list
[
    {
        "accountStatus": "Ready",
        "adminRoleArn": "arn:aws:iam::555555555555:role/DCEMasterAccess",
        "createdOn": 1575485630,
        "id": "775788068104",
        "lastModifiedOn": 1575485630,
        "principalPolicyHash": "\"bc5872b50475b186afea67ff47516a8f\"",
        "principalRoleArn": "arn:aws:iam::775788768154:role/DCEPrincipal-quickstart"
    }
]
```

9. Lease your child account

```
dce leases create --budget-amount 100.0 --budget-currency USD --email jane.doe@email.com --princ
```

10. Log in to your leased account using the `--open-browser` flag to open the AWS Console in your default web browser. See the howto guide for more login options.

```
dce leases login --open-browser <lease-id>
```

CHAPTER 3

How To

3.1 How To

A practical guide to common operations and customizations for DCE.

3.1.1 Use the DCE CLI

The DCE CLI is the easiest way to quickly deploy and use DCE. For more advanced usage, refer to the *DCE API* section.

Installing the DCE CLI

1. Download the appropriate executable for your OS from the latest release. e.g. for mac, you should download dce_darwin_amd64.zip
2. Unzip the artifact and move the executable to a directory on your PATH, e.g.

```
# Download the zip file
wget https://github.com/Optum/dce-cli/releases/download/<VERSION>/dce_darwin_
↪amd64.zip

# Unzip to a directory on your path
unzip dce_darwin_amd64.zip -d /usr/local/bin
```

3. Test the dce command by typing `dce`

```
$ dce
Disposable Cloud Environment (DCE)

The DCE cli allows:

- Admins to provision DCE to a master account and administer said account
```

(continues on next page)

(continued from previous page)

```
- Users to lease accounts and execute commands against them

Usage:
dce [command]

Available Commands:
accounts    Manage dce accounts
auth        Login to dce
help        Help about any command
init        First time DCE cli setup. Creates config file at "$HOME/.dce/config.
↪yaml" (by default) or at the location specifief by "--config"
leases      Manage dce leases
system      Deploy and configure the DCE system
usage       View lease budget information

Flags:
    --config string   config file (default is "$HOME/.dce/config.yaml")
-h, --help            help for dce

Use "dce [command] --help" for more information about a command.
```

4. Type `dce init` to generate a new configuration file. Leave everything blank for now.

Configuring AWS Credentials

The DCE CLI needs AWS IAM credentials any time it interacts with an AWS account. Below is a list of places where the DCE CLI will look for credentials, ordered by precedence.

1. An API Token in the `api.token` field in the configuration file. You may obtain an API Token by:
   - Running the `dce auth` command
   - Base64 encoding the following JSON string. Note that `expireTime` is a Unix epoch timestamp and the string should not contain spaces or newline characters.

```
{
   "accessKeyId":"xxx",
   "secretAccessKey":"xxx",
   "sessionToken":"xxx",
   "expireTime":"xxx"
}
```

2. The Environment Variables: `AWS_ACCESS_KEY_ID`, `AWS_ACCESS_KEY`, and `AWS_SESSION_TOKEN`
3. Stored in the AWS CLI credentials file under the `default` profile. This is located at `$HOME/.aws/credentials` on Linux/OSX and `%USERPROFILE%\.aws\credentials` on Windows.

.

Deploying DCE from the CLI

You can build and deploy DCE from *source* or by using the CLI. This section will cover deployment using the DCE CLI with credentilas configured by the AWS CLI. See *Configuring AWS Credentials* for alternatives.

1. Download and install the AWS CLI

2. Choose an AWS account to be your new "DCE Master Account" and configure the AWS CLI with user credentials that have AdministratorAccess in that account.

```
aws configure set aws_access_key_id default_access_key
aws configure set aws_secret_access_key default_secret_key
```

3. Type `dce system deploy` to deploy dce to the AWS account specified in the previous step.

4. Edit your dce config file with the host and base url from the api gateway that was just deployed to your master account. This can be found in the master account under `API Gateway > (The API with "dce" in the title) > Stages > "Invoke URL: https://<host>/<baseurl>"`. Your config file should look something like this:

```
api:
  host: abcdefghij.execute-api.us-east-1.amazonaws.com
  basepath: /api
region: us-east-1
```

Using advanced deployment options

The DCE CLI uses terraform to provision the infrastructure into the AWS account. You can use the `--tf-init-options` and `--tf-apply-options` to supply options directly to `terraform init` and `terraform apply` (respectively) in the same format in which you would supply them to the `terraform` command.

> Note: if you are an advanced terraform user, you should consider using the **'DCE terraform module directly<terraform.html>'_**.

The `--save-options` flag, if supplied, saves the values supplied to `--tf-init-options` and `--tf-apply-options` in the configuration file in the following locations:

```
terraform:
  initOptions: "-lock=true"
  applyOptions: "-compact-warnings -lock=true"
```

The DCE CLI stores its configuration by default in the `$HOME/.dce/config.yaml` location. This can by overridden using the `--config` command line option. The file is as shown:

```
# The API configutation. This is the DCE API that has been deployed to
# an AWS account.
api:
  # This is the host name only, in the format of
  # {restapi_id}.execute-api.{region}.amazonaws.com
  # For more information, see
  # https://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-call-api.html
  host: api-gateway-id.execute-api.us-east-1.amazonaws.com
  # The stage name of the API Gateway
  # Default: /api
  basepath: /api
# The AWS region. It must match the region configured in the
# api.host. Must be one of:
# "us-east-1", "us-east-2", "us-west-1", "us-west-2"
region: us-east-1
# Terraform configuration
terraform:
  # The full path to the locally-cached terraform binary used
```

(continues on next page)

(continued from previous page)

```
  # by DCE to provision resources. Default value is
  # $HOME/.dce/.cache/terraform/0.12.18/terraform
  bin: /path/to/terraform
  # The source from which terraform was downloaded. This
  # is reserved for future use.
  source: https://download.url.example.com/terraform.zip
  # The options passed to the underlying terraform init command.
  # This value is read if the --tf-init-options command option
  # is not specified or if the DCE_TF_INIT_OPTIONS environment
  # variable is empty, in that order.
  # The format of the value should be just as you would pass
  # them to terraform, as a quoted string.
  initOptions: ""
  # The options passed to the underlying terraform apply command.
  # Like the --tf-init-options flag, the command option is read
  # first, then the DCE_TF_APPLY_OPTIONS environment variable,
  # and lastly this value here. Use the --save-options flag to
  # easily save the values you supply on the CLI to this file.
  # The format of the value should be just as you would pass
  # them to terraform, as a quoted string.
  applyOptions: ""
```

Authenticating with DCE

There are two ways to authenticate with DCE.

1. Use custom IAM credentials for quick access to your individual DCE deployment
2. Use Cognito to set up admin and user profiles

Adding a child account

1. Prepare a second AWS account to be your first "DCE Child Account".
   - Create an IAM role with `AdministratorAccess` and a trust relationship to your DCE Master Accounts
   - Create an account alias by clicking the 'customize' link in the IAM dashboard of the child account. This must not include the terms "prod" or "production".
2. Authenticate as an admin using the `dce auth` command if you are using DCE with AWS Cognito
3. Use the `dce accounts add` command to add your child account to the "DCE Accounts Pool".

   WARNING: This will delete any resources in the account.

```
dce accounts add --account-id 555555555555 --admin-role-arn arn:aws:iam::555555555555:role/DCEMa
```

4. Type `dce accounts list` to verify that your account has been added.

```
dce accounts list
[
    {
        "accountStatus": "NotReady",
        "adminRoleArn": "arn:aws:iam::555555555555:role/DCEMasterAccess",
        "createdOn": 1575485630,
```

(continues on next page)

(continued from previous page)

```
            "id": "775788068104",
            "lastModifiedOn": 1575485630,
            "principalPolicyHash": "\"bc5872b50475b186afea67ff47516a8f\"",
            "principalRoleArn": "arn:aws:iam::775788768154:role/DCEPrincipal-
↪quickstart"
        }
    ]

The account status will initially say ``NotReady``. It may take up to 5 minutes␣
↪for the new account to be processed. Once the account status is ``Ready``\ ,␣
↪you may proceed with creating a lease.
```

Leasing a DCE Account

1. Now that your accounts pool isn't empty, you can create your first lease using the `dce leases create` command.

```
dce leases create --budget-amount 100.0 --budget-currency USD --email jane.doe@email.com --prin
Lease created: {
  "accountId": "555555555555",
  "budgetAmount": 100,
  "budgetCurrency": "USD",
  "budgetNotificationEmails": [
      "jane.doe@email.com"
  ],
  "createdOn": 1575509206,
  "expiresOn": 1576114006,
  "id": "19a742a0-149f-41e5-813a-6d3be101058b",
  "lastModifiedOn": 1575509206,
  "leaseStatus": "Active",
  "leaseStatusModifiedOn": 1575509206,
  "leaseStatusReason": "Active",
  "principalId": "quickstartuser"
}
```

2. Type `dce leases list` to verify that a lease has been created

```
dce leases list
[
  {
      "accountId": "555555555555",
      "budgetAmount": 100,
      "budgetCurrency": "USD",
      "budgetNotificationEmails": [
          "jane.doe@email.com"
      ],
      "createdOn": 1575490207,
      "expiresOn": 1576095007,
      "id": "e501cb86-8317-458b-bdce-d47ab92f86a8",
      "lastModifiedOn": 1575490207,
      "leaseStatus": "Active",
      "leaseStatusModifiedOn": 1575490207,
      "leaseStatusReason": "Active",
      "principalId": "quickstartuser"
```

(continues on next page)

(continued from previous page)

```
  }
]
```

Logging into a leased account

There are three ways to "log in" to a leased account.

1. To use the *AWS CLI* with your leased account, type `dce leases login <lease-id>`. The `default` profile will be used unless you specify a different one with the `--profile` flag.

```
dce leases login --profile quickstart 19a742a0-149f-41e5-813a-6d3be101058b
Adding credentials to .aws/credentials using AWS CLI
cat ~/.aws/credentials
[default]
aws_access_key_id = xxxxxxxxxxxxxxxxxxx
aws_secret_access_key = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

[quickstart]
aws_access_key_id = xxxxxMAJKITANQZPFFXY
aws_secret_access_key = xxxxxDEiaAvZ0OeqO5qxNBcJVrFGzNLxz6tgKWTF
aws_session_token =␣
↪xxxxxXIvYXdzEC0aDFEgMqpsBg4dtUS1qSKyAa3ktoh0SBPbwJv3S5B5NXdG8OdOVCQsya5b943mFfJnxX2reFw1a/
↪r+LKa7G6CKj2NnWbkVWXdzWEVtsjy5Y32po2kVDp1lt74C7V6H8xbOk4HjgiXLOQl5faXpjmi80yaFI/
↪yBrvnBbQVOq9QkbpeHcSyEkoouSkagCtkPicjLjq6omrAGR2xDXrrFYvYRIMevj2mZoBkk/
↪5jGB3FpNycuWz6weqF4Z6qlCZLSalfetEAow7ml7wUyLf4OrtDvPgTPBjg6PClxC6BZgUMZaQM9ePQR0ZgMynNvm7JHbQz
```

2. Access your leased account in a *web browser* via the `dce leases login` command with the `--open-browser` flag

```
dce leases login --open-browser 19a742a0-149f-41e5-813a-6d3be101058b
Opening AWS Console in Web Browser
```

3. To *print your credentials*, type `dce leases login` command with the `--print-creds` flag

```
dce leases login --print-creds 19a742a0-149f-41e5-813a-6d3be101058b
export AWS_ACCESS_KEY_ID=xxxxxMAJKITANQZPFFXY
export AWS_SECRET_ACCESS_KEY=xxxxxDEiaAvZ0OeqO5qxNBcJVrFGzNLxz6tgKWTF
export AWS_SESSION_
↪TOKEN=xxxxxXIvYXdzEC0aDFEgMqpsBg4dtUS1qSKyAa3ktoh0SBPbwJv3S5B5NXdG8OdOVCQsya5b943mFfJnxX2reFw1
↪r+LKa7G6CKj2NnWbkVWXdzWEVtsjy5Y32po2kVDp1lt74C7V6H8xbOk4HjgiXLOQl5faXpjmi80yaFI/
↪yBrvnBbQVOq9QkbpeHcSyEkoouSkagCtkPicjLjq6omrAGR2xDXrrFYvYRIMevj2mZoBkk/
↪5jGB3FpNycuWz6weqF4Z6qlCZLSalfetEAow7ml7wUyLf4OrtDvPgTPBjg6PClxC6BZgUMZaQM9ePQR0ZgMynNvm7JHbQz
```

Ending a Lease

1. End a lease using the `dce leases end` command with the `--account-id` and `--principal-id` flags

```
dce leases end --account-id 555555555555 --principal-id jdoe99
Lease ended
```

2. Type `dce leases list` to verify that the lease has been ended. The `leaseStatus` should now be marked as `Inactive`.

```
dce leases list
[
  {
    "accountId": "555555555555",
    "budgetAmount": 100,
    "budgetCurrency": "USD",
    "budgetNotificationEmails": [
      "jane.doe@email.com"
    ],
    "createdOn": 1575490207,
    "expiresOn": 1576095007,
    "id": "e501cb86-8317-458b-bdce-d47ab92f86a8",
    "lastModifiedOn": 1575490207,
    "leaseStatus": "Inactive",
    "leaseStatusModifiedOn": 1575490207,
    "leaseStatusReason": "Destroyed",
    "principalId": "quickstartuser"
  }
]
```

Removing a Child Account

1. Authenticate as an admin using the `dce auth` command if you are using DCE with AWS Cognito
2. You can remove an account from the accounts pool using the `dce accounts remove` command

```
dce accounts remove 555555555555
```

3.1.2 Use the DCE API

DCE provides a set of endpoints for managing account pools and leases, and for monitoring account usage.

See API Reference Documentation for details.

See *API Auth Documentation* for details on authenticating and authorizing requests.

Prerequisites

Before you can deploy and use DCE, you will need the following:

1. An AWS account to use as the master account, and **sufficient credentials** for deploying DCE into the account.
2. One or more AWS accounts to add as *child accounts* in the account pool. DCE does not *create* any AWS accounts for you. You will need to bring your own AWS accounts for adding to the account pool.
3. In each account you add to the account pool, you will create an IAM role that allows DCE to control the child account.

Deploying DCE from Source

You can build and deploy DCE from source or by using *the CLI*. This section will cover deployment from source. Please ensure you have the following:

1. GNU Make 3.81+

2. Go 1.12.x+
3. Hashicorp Terraform 0.12+
4. The AWS CLI 1.16+

Once you have the requirements installed, you can deploy DCE into your account by following these steps:

1. Clone the Github repository by using the command as shown here:

```
$ git clone https://github.com/Optum/dce.git dce
```

2. Verify that the AWS CLI is configured. with an IAM user that has admin-level permissions in your AWS master account.
3. Make sure that the AWS region is set to *us-east-1* by using the command as shown:

```
$ aws configure list
    Name                    Value             Type    Location
    ----                    -----             ----    --------
profile                <not set>             None    None
access_key     ****************NXAW shared-credentials-file
secret_key     ****************ymwP shared-credentials-file
region                 us-east-1      config-file    ~/.aws/config
```

4. Change into the base directory and use `make` to deploy the code as shown here:

```
$ cd dce
$ make deploy_local
```

When the last command is complete, you will have DCE deployed into your master account.

Finding the DCE API URL

The API is hosted by AWS API Gateway. The base URL is exposed as a Terraform output. API Gateway generates a unique ID as part of the API URL. To retrieve the base url of the API, run the following command from the Terraform modules directory:

```
terraform output api_url
```

All endpoints use this value as the base url. For example, to view accounts:

```
GET https://asdfghjkl.execute-api.us-east-1.amazonaws.com/api/accounts
```

Authenticating with DCE

There are two ways to authenticate with DCE.

1. Use custom IAM credentials for quick access to your individual DCE deployment
2. Use Cognito to set up admin and user profiles

Adding Accounts to the DCE Account Pool

DCE manages its collection of AWS accounts in an account pool. Each account in the pool is made available for leasing by DCE users.

DCE *does not* create AWS accounts. These must be added to the account pool by a DCE administrator. You can create accounts using the AWS CreateAccount API.

The child account must have an administrative IAM Role with a trust relationship to allow the master account to assume the role. For example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::MASTER_ACCOUNT_ID:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Use the `/accounts` endpoint to add an account to the DCE accounts pool.

Request

`POST ${api_url}/accounts`

```
{
    "adminRoleArn": "arn:aws:iam::123456789012:role/DCEAdmin",
    "id": "123456789012"
}
```

Response

```
{
    "accountStatus": "NotReady",
    "adminRoleArn": "arn:aws:iam::123456789012:role/DCEAdmin",
    "createdOn": 1572379783,
    "id": "123456789012",
    "lastModifiedOn": 1572379783,
    "metadata": null,
    "principalPolicyHash": "\"852ee9abbf1220a111c435a8c0e65490\"",
    "principalRoleArn": "arn:aws:iam::123456789012:role/DCEPrincipal"
}
```

You can verify the account has been added with the following:

Request

`GET ${api_url}/accounts`

Response

```
[
    {
        "accountStatus": "Ready",
        "adminRoleArn": "arn:aws:iam::123456789012:role/DCEAdmin",
        "createdOn": 1572379783,
        "id": "123456789012",
        "lastModifiedOn": 1572379888,
        "metadata": null,
        "principalPolicyHash": "\"852ee9abbf1220a111c435a8c0e65490\"",
```

(continues on next page)

(continued from previous page)

```
        "principalRoleArn": "arn:aws:iam::123456789012:role/DCEPrincipal"
    }
]
```

Leasing a child account

Now that the child account has been added to the account pool, you can create a lease on the account.

Request

`POST ${api_url}/leases`

```
{
    "principalId": "DCEPrincipal",
    "accountId": "123456789012",
    "budgetAmount": 20,
    "budgetCurrency": "USD",
    "budgetNotificationEmails": [
        "myuser@example.com"
    ],
    "expiresOn": 1572382800
}
```

Response

```
{
    "accountId": "123456789012",
    "budgetAmount": 20,
    "budgetCurrency": "USD",
    "budgetNotificationEmails": [
        "myuser@example.com"
    ],
    "createdOn": 1572381585,
    "expiresOn": 1572382800,
    "id": "94503268-426b-4892-9b53-3c73ab38aeff",
    "lastModifiedOn": 1572381585,
    "leaseStatus": "Active",
    "leaseStatusModifiedOn": 1572381585,
    "leaseStatusReason": "",
    "principalId": "DCEPrincipal"
}
```

After getting the response, call the `/accounts` endpoint again to see that the account status has been changed to `Leased`:

Request

`GET ${api_url}/accounts`

Response

```
[
    {
        "accountStatus": "Leased",
        "adminRoleArn": "arn:aws:iam::123456789012:role/DCEAdmin",
        "createdOn": 1572379783,
        "id": "123456789012",
```

(continues on next page)

(continued from previous page)


```
        "lastModifiedOn": 1572381585,
        "metadata": null,
        "principalPolicyHash": "\"852ee9abbf1220a111c435a8c0e65490\"",
        "principalRoleArn": "arn:aws:iam::123456789012:role/DCEPrincipal"
    }
]
```


You may begin using your leased account once it's status has changed to `Leased`.

Listing leases

You may list leases using the `/leases` endpoint

Request

```
GET ${api_url}/leases
```

Response


```
[
    {
        "accountId": "123456789012",
        "budgetAmount": 20,
        "budgetCurrency": "USD",
        "budgetNotificationEmails": [
            "myuser@example.com"
        ],
        "createdOn": 1572381585,
        "expiresOn": 1572382800,
        "id": "94503268-426b-4892-9b53-3c73ab38aeff",
        "lastModifiedOn": 1572381585,
        "leaseStatus": "Active",
        "leaseStatusModifiedOn": 1572381585,
        "leaseStatusReason": "Active",
        "principalId": "DCEPrincipal"
    }
]
```


Logging into a leased account

The easiest way to log into a leased account is by using the *DCE CLI*. The following steps cover how to log in without using the CLI:

1. Configure *DCE Authentication* if you have not already done so
2. Open a web browser (Google Chrome is recommended)
3. Navigate to `${api_url}/auth` and authenticate as prompted. You will be redirected to a page displaying an authentication code.
4. Base64 decode the authentication code to view plaintext credentials of the form:


```
{
   "accessKeyId":"xxx",
   "secretAccessKey":"xxx",
   "sessionToken":"xxx",
```


(continues on next page)

(continued from previous page)

```
   "expireTime":"Wed Nov 20 2019 13:30:13 GMT-0600 (Central Standard Time)"
}
```

Ending a lease

Leases automatically expire based on their expiration date or budget amount, but leases may also be administratively destroyed at any time. To destroy a lease with the API, send a DELETE request to the `/leases` endpoint.

Request

```
DELETE ${api_url}/leases
```

```
{
    "principalId": "DCEPrincipal",
    "accountId": "123456789012"
}
```

Response

```
{
    "accountId": "519777115644",
    "budgetAmount": 20,
    "budgetCurrency": "USD",
    "budgetNotificationEmails": [
        "john.doe@example.com"
    ],
    "createdOn": 1572381585,
    "expiresOn": 1572382800,
    "id": "94503268-426b-4892-9b53-3c73ab38aeff",
    "lastModifiedOn": 1572442028,
    "leaseStatus": "Inactive",
    "leaseStatusModifiedOn": 1572442028,
    "leaseStatusReason": "Destroyed",
    "principalId": "jdoe123"
}
```

3.1.3 Configure Deployment Options

Budgets and Lease Periods

Every lease comes with a configured **per-lease budget**, which limits AWS account spend during the course of the lease. Additionally there are **per-principal budgets**, which limit spend by a single user across multiple leases during a budget period. This prevents a single user from creating multiple leases as a way of circumventing lease budgets.

DCE budget may be configured as Terraform variables.

Variable	Default	Description
`max_lease_budget_amount`	1000	The maximum budget a user may request for their lease
`max_lease_period`	604800	The maximum duration (seconds) a user may request for their lease
`principal_budget_amount`	1000	The maximum spend a user may accumulate across any number of leases during the `principal_budget_period`
`principal_budget_period`	"WEEKLY"	The period across which the `principal_budget_amount` is measured. Currently only supports "WEEKLY"

Account Resets

To reset AWS accounts between leases, DCE uses the open source aws-nuke tool. This tool attempts to delete every single resource in th AWS account, and will make several attempts to ensure everything is wiped clean.

To prevent `aws-nuke` from deleting certain resources, provide a YAML configuration with a list of resource *filters*. (see aws-nuke docs for the YAML filter configuration syntax). By default, DCE filters out resources which are critical to running DCE – for example, the IAM roles for your account's `adminRoleArn` / `principalRoleArn`.

As a DCE implementor, you may have additional resources you wish protect from `aws-nuke`. If this is the case, you may specify your own custom `aws-nuke` YAML configuration:

- Copy the contents of default-nuke-config-template.yml into a new file
- Modify as needed.
- Upload the YAML configuration file to an S3 bucket in the DCE master account

Then configure reset using Terraform variables:

Variable	Default	Description
`reset_nuke_template_bucket`	See default-nuke-config-template.yml	S3 bucket where a custom aws-nuke configuration is located
`reset_nuke_template_key`	See default-nuke-config-template.yml	S3 key within the `reset_nuke_template_bucket` where a custom aws-nuke configuration is located
`reset_nuke_toggle`	`true`	Set to false to disable aws-nuke
`allowed_regions`	*all AWS regions*	AWS regions which will be nuked. Allowing fewer regions will drastically reduce the run time of aws-nuke

Budget Notifications

When a lease owner approaches or exceeds their budget, they will receive an email notification. These notifications are configurable as Terraform variables:

Variable	Default	Description
`check_budget_enabled`	`true`	Set to `false` to disable budget checks entirely
`budget_notification_threshold_percentiles`	`[75, 100]`	Thresholds (percentiles) at which budget notification emails will be sent to users.
`budget_notification_from_email`	`"dce@example.com"`	`FROM` email address for budget notifications
`budget_notification_bcc_emails`	`[]`	Budget notifications emails will be BCC'd to these addresses
`budget_notification_template_subject`	See variables.tf	Template for budget notification email subject
`budget_notification_template_text`	See variables.tf	Template for budget notification text emails
`budget_notification_template_html`	See variables.tf	Template for budget notification HTML emails

Email Templates

Budget notification email templates are rendered using golang templates, and accept the following arguments:

Argument	Description
IsOverBudget	Set to `true` if the account is over the configured budget
Lease.PrincipalID	The principal ID of the lease holder
Lease.AccountID	The Account number of the AWS account in use
Lease.BudgetAmount	The configured budget amount for the lease
ActualSpend	The calculated spend on the account at time of notification
ThresholdPercentile	The configured threshold percentage for the notification

AWS Regions

By default, DCE users are limited to working in `us-east-1` by IAM Policy. Limiting users to a small number of regions reduces the amount of time it takes to reset accounts.

To override this behavior, you may set the terraform `allowed_regions` variable to a list of AWS region names.

3.1.4 Backup DCE Database Tables

DCE does not backup DynamoDB tables by default. However, if you want to restore a DynamoDB table from a backup, we do provide a helper script in scripts/restore_db.sh. This script is also provided as a Github release artifact, for easy access.

To restore a DynamoDB table from a backup:

```
# Grab the account table name from Terraform state
table_name=$(cd modules && terraform output accounts_table_name)

# Or, grab the leases table name
table_name=$(cd modules && terraform output leases_table_name)

# List available backups
./scripts/restore_db.sh \
  --target-table-name ${table_name} \
  --list-backups

# Choose an backup from the output of the last command, and pass in the ARN
./scripts/restore_db.sh \
  --target-table-name ${table_name} \
  --backup-arn <backup arn>

# If the table already exists, and you want to delete and
# recreate it from a backup, pass in
# the --force-delete-table flag
./scripts/restore_db.sh \
  --target-table-name ${table_name} \
  --backup-arn <backup arn> \
  --force-delete-table
```

After restoring the DynamoDB table from a backup, re-apply Terraform to ensure that your table is in sync with your Terraform configuration.

See AWS guide for backing up DynamoDB tables.

3.1.5 Monitor DCE

CloudWatch Dashboard

DCE comes with a prebuilt CloudWatch dashboard for monitoring things like API calls, account resets, and errors. To enable the DCE CloudWatch Dashboard, set the `cloudwatch_dashboard_toggle` terraform variable to `true` during deployment. e.g.

```
terraform apply -var cloudwatch_dashboard_toggle=true
```

The DCE CloudWatch Dashboard is disabled by default.

Account Pool Monitoring

DCE account pool monitoring may be enabled via the `account_pool_metrics_toggle` terraform variable. Account pool monitoring publishes CloudWatch metrics on the number of accounts in each status (i.e. `Ready`, `Leased`, `NotReady`, and `Orphaned`). The following CloudWatch alarms are included:

- `ready-accounts`: triggers when the number of `Ready` accounts is below a configurable threshold. Controlled by the `ready_accounts_alarm_threshold` terraform variable.
- `orphaned-accounts`: triggers when the number of `Orphaned` accounts is above a configurable threshold. Controlled by the `orphaned_accounts_alarm_threshold` terraform variable.

To enable this feature with logical defaults, simply use:

```
terraform apply \
  -var account_pool_metrics_toggle=true \
  -var cloudwatch_dashboard_toggle=true \
```

DCE periodically queries the Accounts table to retrieve the number of accounts in each status. The frequency of these queries can be controlled using the `account_pool_metrics_collection_rate_expression` terraform variable.

The DCE CloudWatch dashboard includes an `Account Pool` widget that displays the number of accounts in each status over time. The period over which metrics are aggregated in this widget can be controlled using the `account_pool_metrics_widget_period` terraform variable.

In order for data to display accurately in the `Account Pool` dashboard widget, the period of time over which data is aggregated in the widget (`account_pool_metrics_widget_period`) must be shorter than the metrics sampling interval (`account_pool_metrics_collection_rate_expression`). Otherwise, multiple samples will be aggregated together in each data point.

For example, if `account_pool_metrics_collection_rate_expression` is set to `rate(30 minutes)`, then `1200` se
would be an acceptable value for `account_pool_metrics_widget_period`.

You may need to increase the DynamoDB Read Capacity Units on the Accounts table in order to accommodate this feature periodically querying all of the Account records. 13 RCUs per 100 accounts should be sufficient to avoid throttling. If needed,

refer to the AWS Documentation for assistance in

calculating the required read capacity units appropriate for your usage. This may be adjusted using the `accounts_table_rcu` terraform variable.

CloudWatch Alarms

DCE also comes prebuilt with a number of CloudWatch alarms, which will trigger when DCE systems encounter errors or behave abnormally.

These CloudWatch alarms are delivered to an SNS topic. The ARN of this SNS topic is available as a Terraform output:

```
cd modules
terraform output alarm_sns_topic_arn
```

Subscribe to this topic to receive alarm notifications. For example:

```
aws sns subscribe \
  --topic-arn <Alarm Topic ARN> \
  --protocol email \
  --notification-endpoint my-email@example.com
```

CHAPTER 4

Concepts

4.1 Concepts

4.1.1 DCE

Disposable Cloud Environment (DCE) provide temporary, limited access to Amazon Web Services (AWS) accounts. Administrators can configure this limited access to expire based on time or budget. When the access expires, DCE destroys all of the resources in the account and returns the account to the account pool.

4.1.2 Account

An *account* is an AWS account that is available for leasing.

4.1.3 Lease

A *lease* is temporary access to an AWS account. A lease has a budget, an expiration date, and a principal user.

4.1.4 Reset

DCE *resets* a leased child account during *any one* of the following conditions:

- The time set on the `expiresOn` field is now in the past
- The amount set on the `budgetAmount` field is exceeded
- With a `/leases` API call or CLI command

To reset an account, DCE performs the following actions, in order:

1. Marks the lease as Inactive
2. Marks the account as Not Ready

3. Deletes all of the resources in the account.
4. Marks the account as Ready

4.1.5 Account Status

The *account status* indicates if the account is ready to be leased, leased already, or in the process of being prepared to be leased again.

Ready

An account in *Ready* status is available for leasing. All of the resources in the account have been cleaned and the account is like a brand-new, fresh AWS account with the exception of an IAM role.

Not Ready

An account in *Not Ready* status is in the process of being reset so that it can be marked as Ready.

Leased

An account in *Leased* status is currently in use. A lease means that the account is "checked out", much like a library book, a rental car, or a hotel room.

4.1.6 Lease Status

The *lease status* indicates whether or not a lease is currently in use.

Active

An *active* lease is currently in use by the principal associated with the lease.

Inactive

An *inactive* lease is a lease that has either expired or the usage in the leased account has exceeded the budget on the lease.

4.1.7 Lease Status Reason

Expired

A lease that is *expired* has exceeded the time set by the `expiresOn` field of the lease.

The API accepts a `expiresOn` field during lease creation. DCE uses a configurable default read from the `DEFAULT_LEASE_LENGTH_IN_DAYS` environment variable when the `expiresOn` field is not present. If the configurable default is unset, DCE uses a period of seven (7) days.

OverBudget

A lease that is *over budget* has exceeded the budget amount set by the `budgetAmount` field of the lease.

Each lease has a configurable budget. DCE periodically monitors the leased child accounts to determine when usage exceeds the budget amount queues the account for reset.

Destroyed

A lease may be destroyed before it expires or exceeds budget through the API or CLI. In this case, the lease status is marked "Inactive" and the reason is "Destroyed". The account associated with the lease is then reset and the account is returned to the account pool.

Active

A lease with an *Active* status reason is an active lease.

Rollback

A lease with the *Rollback* lease status reason has experienced a failure while DCE was getting the child account ready from the account pool. In the event of a failure, DCE sets the lease status to *Inactive* and the reason to *Rollback* and returns the child account to the child pool.

4.1.8 Account Pool

The *account pool* is the collection of *child accounts* that are available for leasing.

4.1.9 Master Account

The *master account* is the AWS account that contains the DCE infrastructure used to manage the child accounts that are in the account pool.

4.1.10 Child account

A *child account* is an AWS account added to the account pool and controlled by the infrastructure in the master account.

DCE requires an IAM role and permissions permissions to assume the role from the master account to control resources in the child account

4.1.11 Principal

The *principal* is the user to whom a child account is leased.

4.1.12 Admin

The *admin* is the user responsible for administering DCE.

4.1.13 Admin Role

The *admin role* is the role in the master account assumed by DCE to obtain access to all resources in both the master and the child accounts.

4.1.14 Principal Role

The *principal role* is the IAM role in the child account that the principal assumes in order to access the resources in the account.

4.1.15 Budget

A *budget* is the amount of maximum spending that should be incurred during the lease. If the usage in the account exceeds the budget amount, DCE resets the account.

4.1.16 Usage

In DCE, *usage* refers to the cost of running AWS resources in the accounts.

CHAPTER 5

DCE IAM Policies

5.1 DCE IAM Policies

5.1.1 Understanding Principal Policies

When an AWS account is added to the DCE account pool, an IAM role and policy are created within the account. This role is assumed by end-users when accessing their leased account.

The principal user's IAM role is returned as `principalRoleArn` when *creating a new account via the DCE API*. For example:

```
{
  "id": "123456789012",
  "adminRoleArn": "arn:aws:iam::123456789012:role/OrganizationAccountAccessRole",
  "principalRoleArn": "arn:aws:iam::123456789012:role/DCEPrincipal"
}
```

By default, the `DCEPrincipal` role has *near-administrative* access to their leased account, with a few exceptions:

- Users may not create AWS support tickets
  - e.g., we don't want users increasing service limits
- Users may not modify resources required by DCE to manage the child account
  - e.g. users cannot modify the IAM Trust Relationship which allows DCE master to assume into the child account's IAM roles
- Users are limited to a set of configured regions
  - This is to limit the scope of account resets. See Configuring Account Resets
- Users are limited to AWS services which DCE knows how to destroy.
  - This is to prevent orphan resources in accounts after reset.

5.1.2 Principal Role Security

By default, **principal users may elevate their own IAM access**. For example, users may create a new IAM role with an attached `AdministrativeAccess` policy, assign the role to an EC2 instance, and then SSH into the instance as an admin user.

The best way to block this backdoor access to IAM policy elevation is through a Service Control Policy, or SCP. An SCP is an organization-level policy which allows administrators to control access to *all* IAM roles and users within the organizations.

See *DCE Service Control Policies*

5.1.3 DCE Service Control Policies (SCP)

Implementing DCE in an AWS Organization provides the ability to use SCPs, which can be helpful for ensuring the resilience of DCE internal resources. The following SCP is an example policy that contains two statements for protecting your DCE accounts:

- **DenyChangesToAdminPrincipalRoleAndPolicy** is designed to prevent anyone other than the AdminRole from modifying the roles and policies used by DCE.
- **DenyUnsupportedServices** is designed to allow access only to services that are supported by AWS Nuke

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyChangesToAdminPrincipalRoleAndPolicy",
            "Effect": "Deny",
            "NotAction": [
                "iam:GetContextKeysForPrincipalPolicy",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListAttachedRolePolicies",
                "iam:ListInstanceProfilesForRole",
                "iam:ListRolePolicies",
                "iam:ListRoleTags",
                "iam:DeactivateMFADevice",
                "iam:CreateSAMLProvider",
                "iam:UpdateAccountPasswordPolicy",
                "iam:DeleteVirtualMFADevice",
                "iam:EnableMFADevice",
                "iam:CreateAccountAlias",
                "iam:DeleteAccountAlias",
                "iam:UpdateSAMLProvider",
                "iam:DeleteSAMLProvider"
            ],
            "Resource": [
                "arn:aws:iam::*:role/AdminRole",
                "arn:aws:iam::*:role/DCEPrincipal*",
                "arn:aws:iam::*:policy/DCEPrincipal*"
            ],
            "Condition": {
                "StringNotLike": {
                    "aws:PrincipalARN": "arn:aws:iam::*:role/AdminRole"
                }
            }
```

(continues on next page)

(continued from previous page)

```
        },
        {
            "Sid": "DenyUnsupportedServices",
            "Effect": "Deny",
            "NotAction": [
                "acm:*",
                "acm-pca:*",
                "apigateway:*",
                "application-autoscaling:*",
                "appstream:*",
                "athena:*",
                "autoscaling:*",
                "aws-portal:*",
                "backup:*",
                "batch:*",
                "budgets:*",
                "cloud9:*",
                "clouddirectory:*",
                "cloudformation:*",
                "cloudfront:*",
                "cloudhsm:*",
                "cloudsearch:*",
                "cloudtrail:*",
                "cloudwatch:*",
                "codebuild:*",
                "codecommit:*",
                "codedeploy:*",
                "codepipeline:*",
                "codestar:*",
                "cognito-identity:*",
                "cognito-idp:*",
                "cognito-sync:*",
                "comprehend:*",
                "config:*",
                "datapipeline:*",
                "dax:*",
                "devicefarm:*",
                "dms:*",
                "ds:*",
                "dynamodb:*",
                "ec2:*",
                "ecr:*",
                "ecs:*",
                "eks:*",
                "elasticache:*",
                "elasticbeanstalk:*",
                "elasticfilesystem:*",
                "elasticloadbalancing:*",
                "elasticmapreduce:*",
                "elastictranscoder:*",
                "es:*",
                "events:*",
                "execute-api:*",
                "firehose:*",
                "fsx:*",
                "globalaccelerator:*",
                "glue:*",
```

(continues on next page)

(continued from previous page)

```
                "iam:*",
                "imagebuilder:*",
                "iot:*",
                "iotanalytics:*",
                "kafka:*",
                "kinesis:*",
                "kinesisanalytics:*",
                "kinesisvideo:*",
                "kms:*",
                "lakeformation:*",
                "lambda:*",
                "lex:*",
                "lightsail:*",
                "logs:*",
                "machinelearning:*",
                "mediaconvert:*",
                "medialive:*",
                "mediapackage:*",
                "mediastore:*",
                "mediatailor:*",
                "mobilehub:*",
                "mq:*",
                "neptune-db:*",
                "opsworks:*",
                "opsworks-cm:*",
                "rds:*",
                "redshift:*",
                "rekognition:*",
                "resource-groups:*",
                "robomaker:*",
                "route53:*",
                "s3:*",
                "sagemaker:*",
                "sdb:*",
                "secretsmanager:*",
                "servicecatalog:*",
                "servicediscovery:*",
                "servicequotas:*",
                "ses:*",
                "sns:*",
                "sqs:*",
                "ssm:*",
                "states:*",
                "storagegateway:*",
                "sts:*",
                "tag:*",
                "transfer:*",
                "waf:*",
                "wafv2:*",
                "waf-regional:*",
                "worklink:*",
                "workspaces:*"
            ],
            "Resource": "*"
        }
    ]
}
```

5.1.4 Customizing the Principal IAM Policy

Customize the IAM Policies for DCE Principals via Terraform variables.

See Configuring Terraform Variables.

Variable	Default	Description
`principal_policy`	See principal_policy.tmpl	File location for a IAM principal policy template
`allowed_regions`	*all AWS regions*	AWS regions which the principal is allowed to access

The file specified in `principal_policy` is rendered using golang templates, and accepts the following arguments:

Argument	Description
PrincipalPolicyArn	ARN of the principal IAM policy
PrincipalRoleArn	ARN of the principal IAM role
AdminRoleArn	ARN of the admin access role within the account
PrincipalIAMDenyTags	Populated from the `principal_iam_deny_tags` Terraform variable. By default, these are used to deny access to AWS resources with `AppName=DCE` tags
Regions	AWS Regions, populated from the `allowed_regions` Terraform variable

CHAPTER 6

Deploy with Terraform

6.1 Deploy with Terraform

The AWS infrastructure for the DCE master account is defined as a Terraform module within the github.com/Optum/dce repo. This infrastructure may be deployed using the Terraform CLI:

```
cd modules
terraform init
terraform apply
```

See terraform.io for more information on using Terraform.

After the Terraform deployment is complete, you will need to build and deploy the application code to AWS:

```
make deploy
```

Alternatively, you can download the build artifacts from a Github release, and deploy them directly. Both the `deploy.sh` and `build_artifacts.zip` are supplied with the github release:

```
cd modules
namespace=$(terraform output namespace)
artifacts_bucket=$(terraform output artifacts_bucket_name)
deploy.sh build_artifacts.zip ${namespace} ${artifacts_bucket}
```

6.1.1 Configuring Terraform Variables

The DCE Terraform module accepts a number of configuration variables to tweak the behavior of the DCE deployment. These variables can be provided to the `terraform apply` CLI command, or configured in a `tfvars` file.

For example:

```
terraform apply \
    -var namespace=nonprod \
    -var check_budget_enabled=false \
    -var-file my-dce.tfvars
```

See Terraform documentation for details on configuring input variables.

See /modules/variables.tf for a full list of configurable Terraform variables.

6.1.2 Accessing Terraform Outputs

The DCE Terraform module outputs a number of parameters, which may be useful for interacting with the configured resources. For example, the `api_url` output provides the base url for your DCE API Gateway endpoint.

Use the terraform output CLI command to access outputs.

```
cd modules
terraform output api_url
```

For a full list of available outputs, see /modules/outputs.tf

6.1.3 Extending the Terraform Configuration

You may want to extend the DCE Terraform configuration with our own infrastructure. For example, you may want to subscribe your own Lambda to DCE *SNS Lifecycle Events*.

To do this, pull in the DCE Terraform module as a submodule from within your own Terraform configuration:

```
# Load DCE as a Terraform submodule
module "dce" {
  source = "github.com/Optum/dce//modules"
  # Optionally, configure additional input variables
  namespace= "nonprod"
  check_budget_enabled = false
}

# Reference DCE module outputs as needed
# For example, here we'll subscribe to the "lease-added" SNS topic
resource "aws_sns_topic_subscription" "assign_topic_lambda" {
  topic_arn = module.dce.lease_added_topic_arn
  protocol  = "lambda"
  endpoint  = aws_lambda_function.my_fn.arn
}
resource "aws_lambda_permission" "assign_sns" {
  statement_id  = "AllowExecutionFromSNS"
  action        = "lambda:InvokeFunction"
  function_name = aws_lambda_function.my_fn.name
  principal     = "sns.amazonaws.com"
  source_arn    = module.dce.lease_added_topic_arn
}
```

CHAPTER 7

API Reference

7.1 API Reference

7.1.1

OPTIONS /accounts

CORS support

- **Description:** Enable CORS by returning correct headers
- **Consumes:** [‘application/json’]
- **Produces:** [‘application/json’]

Responses

200 - Default response for CORS method

GET /accounts

Lists accounts

- **Produces:** [‘application/json’]

Responses

200 - A list of accounts

403 - Unauthorized

POST /accounts

Add an AWS Account to the account pool

- **Consumes:** [‘application/json’]
- **Produces:** [‘application/json’]

Parameters

Name	Position	Description	Type
account	body	Account creation parameters	

Responses

201 -

403 - Failed to authenticate request

OPTIONS /accounts/{id}

CORS support

- **Description:** Enable CORS by returning correct headers
- **Consumes:** [‘application/json’]
- **Produces:** [‘application/json’]

Responses

200 - Default response for CORS method

GET /accounts/{id}

Get a specific account by an account ID

- **Produces:** [‘application/json’]

Parameters

Name	Position	Description	Type
id	path	AWS Account ID	string

Responses

200 -

403 - Failed to retrieve account

PUT /accounts/{id}

Update an account

- **Consumes:** [‘application/json’]
- **Produces:** [‘application/json’]

Parameters

Name	Position	Description	Type
id	path	AWS Account ID	string
account	body	Account parameters to modify	

Responses

200 -

403 - Forbidden

DELETE /accounts/{id}

Delete an account by ID.

Parameters

Name	Position	Description	Type
id	path	The ID of the account to be deleted.	string

Responses

204 - The account has been successfully deleted.

403 - Unauthorized.

404 - No account found for the given ID.

409 - The account is unable to be deleted.

OPTIONS /leases

CORS support

- **Description:** Enable CORS by returning correct headers
- **Consumes:** ['application/json']
- **Produces:** ['application/json']

Responses

200 - Default response for CORS method

POST /leases

Creates a new lease.

- **Consumes:** ['application/json']
- **Produces:** ['application/json']

Parameters

Name	Position	Description	Type
lease	body	The owner of the lease	

Responses

201 -

400 - If the "expiresOn" date specified is non-zero but less than the current epoch date, "Requested lease has a desired expiry date less than today: <date>" or "Failed to Parse Request Body" if the request body is blank or incorrectly formatted.

403 - Failed to authenticate request

409 - Conflict if there is an existing lease already active with the provided principal and account.

500 - Server errors if the database cannot be reached.

DELETE /leases

Removes a lease.

- **Consumes:** ['application/json']
- **Produces:** ['application/json']

Parameters

Name	Position	Description	Type
lease	body	The owner of the lease	

Responses

200 -

400 - "Failed to Parse Request Body" if the request body is blank or incorrectly formatted. or if there are no account leases found for the specified accountId or if the account specified is not already Active.

403 - Failed to authenticate request

500 - Server errors if the database cannot be reached.

GET /leases

Get leases

- **Produces:** ['application/json']

Parameters

Name	Position	Description	Type
principalId	query	Principal ID of the leases.	string
accountId	query	Account ID of the leases.	string
status	query	Status of the leases.	string
nextPrincipalId	query	Principal ID with which to begin the scan operation. This is used to traverse through paginated results.	string
nextAccountId	query	Account ID with which to begin the scan operation. This is used to traverse through paginated results.	string
limit	query	The maximum number of leases to evaluate (not necessarily the number of matching leases). If there is another page, the URL for page will be in the response Link header.	integer

Responses

200 - OK

400 - "Failed to Parse Request Body" if the request body is blank or incorrectly formatted.

403 - Failed to authenticate request

OPTIONS /leases/{id}

CORS support

- **Description:** Enable CORS by returning correct headers
- **Consumes:** ['application/json']
- **Produces:** ['application/json']

Responses

200 - Default response for CORS method

GET /leases/{id}

Get a lease by Id

- **Produces:** ['application/json']

Parameters

Name	Position	Description	Type
id	path	Id for lease	string

Responses

200 -

403 - Failed to retrieve lease

OPTIONS /leases/{id}/auth

CORS support

- **Description:** Enable CORS by returning correct headers
- **Consumes:** [‘application/json’]
- **Produces:** [‘application/json’]

Responses

200 - Default response for CORS method

POST /leases/{id}/auth

Create lease authentication by Id

- **Produces:** [‘application/json’]

Parameters

Name	Position	Description	Type
id	path	Id for lease	string

Responses

201 -

401 - Unauthorized

403 - Failed to retrieve lease authentication

500 - Server failure

OPTIONS /usage

CORS support

- **Description:** Enable CORS by returning correct headers
- **Consumes:** [‘application/json’]
- **Produces:** [‘application/json’]

Responses

200 - Default response for CORS method

GET /usage

Get usage records by date range

- **Produces:** [‘application/json’]

Parameters

Name	Position	Description	Type
startDate	path	start date of the usage	number
endDate	path	end date of the usage	number

Responses

200 -

403 - Failed to authenticate request

CHAPTER 8

API Auth

8.1 API Auth

There are two ways to authenticate against the DCE APIs:

1. *AWS Cognito*
2. *IAM credentials*

8.1.1 Roles

Admins

Admins have full access to all APIs and will not get back filtered results when querying APIs.

There are three different ways a user is considered an admin:

1. They have an IAM user/role/etc with a policy that gives them access to the API
2. A Cognito user is placed into a Cognito group called `Admins`
3. A Cognito user has an attribute in `custom:roles` that will match a search criteria specified by the Terraform variable `cognito_roles_attribute_admin_name`

Users

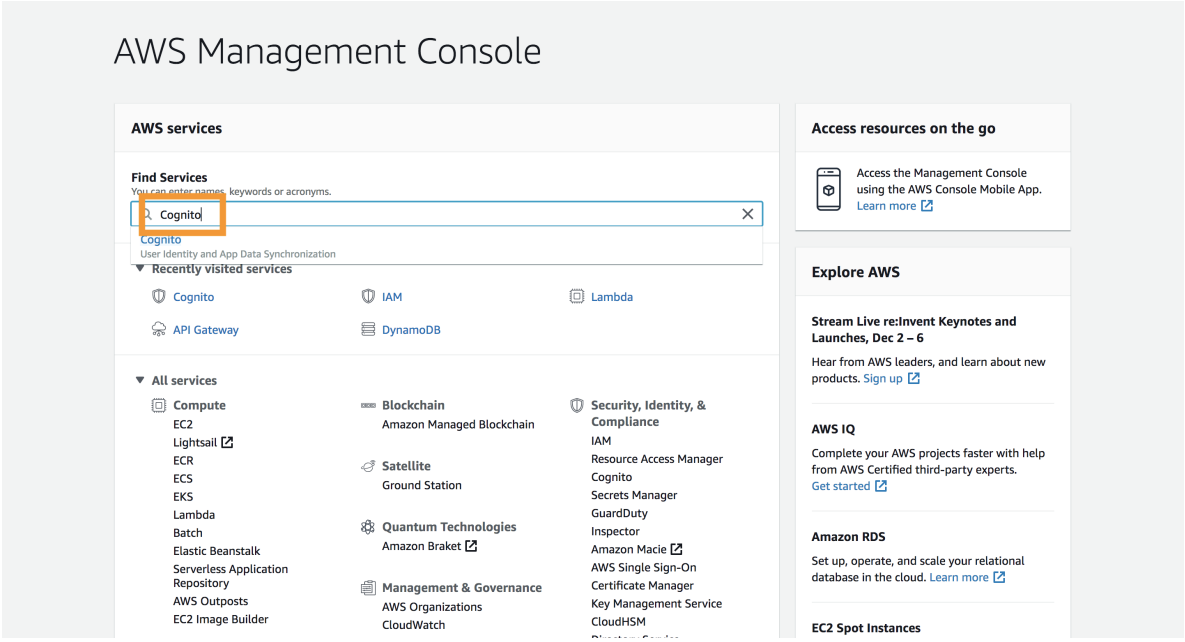
Users are given access to the leases and usage APIs. This is done so they can request their own lease and look at the usage of their leases. Any user authenticated through Cognito will automatically fall into the `Users` role unless designated as an Admin.

8.1.2 Using AWS Cognito

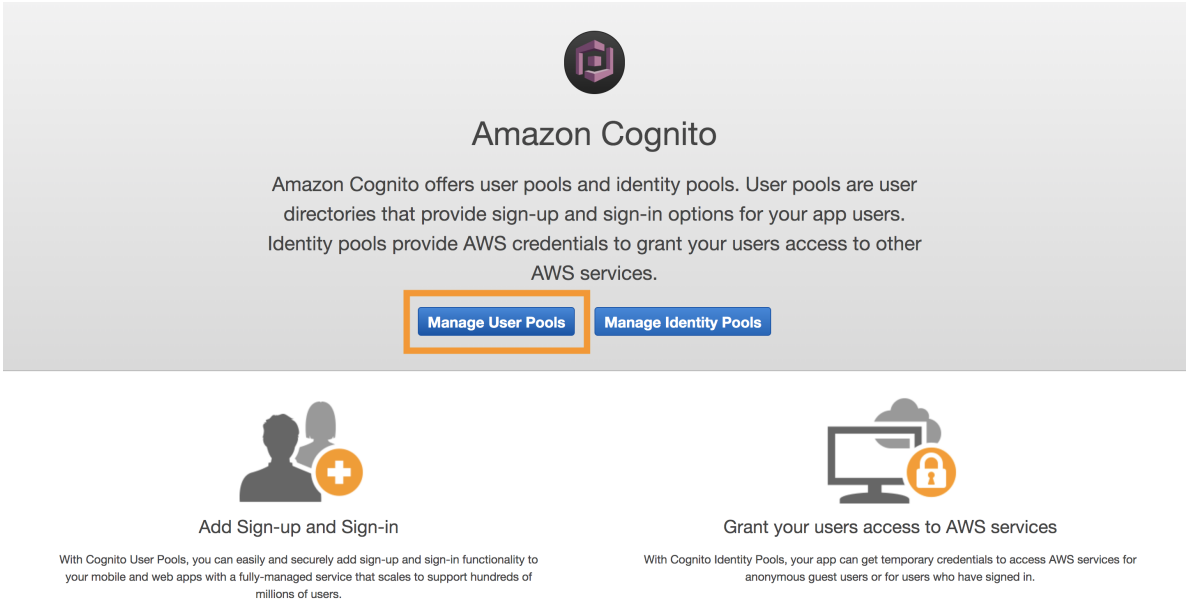
AWS Cognito is used to authenticate and authorize DCE users. This section will walk through setting this up in the AWS web console, but note that all of these operations can be automated using the AWS CLI or SDKs. While this example uses Cognito User Pools to create and manage users, you may also integrate Cognito with your own IdP.

Configuring Cognito

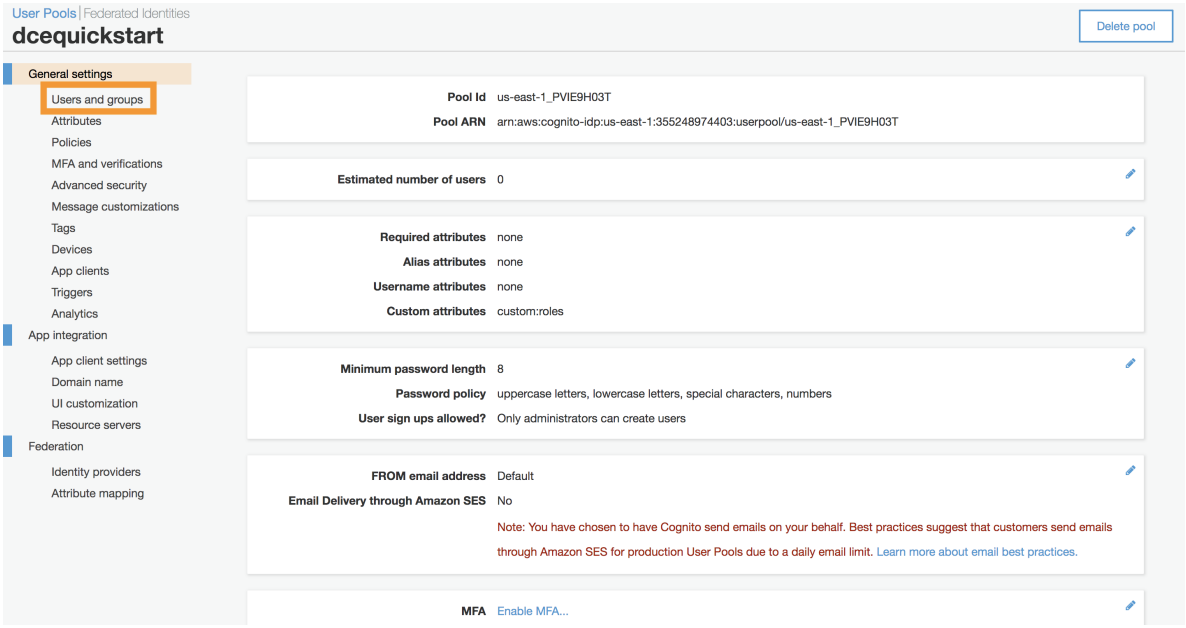
1. Open the AWS Console in your DCE Master Account and Navigate to AWS Cognito by typing `Cognito` in the search bar



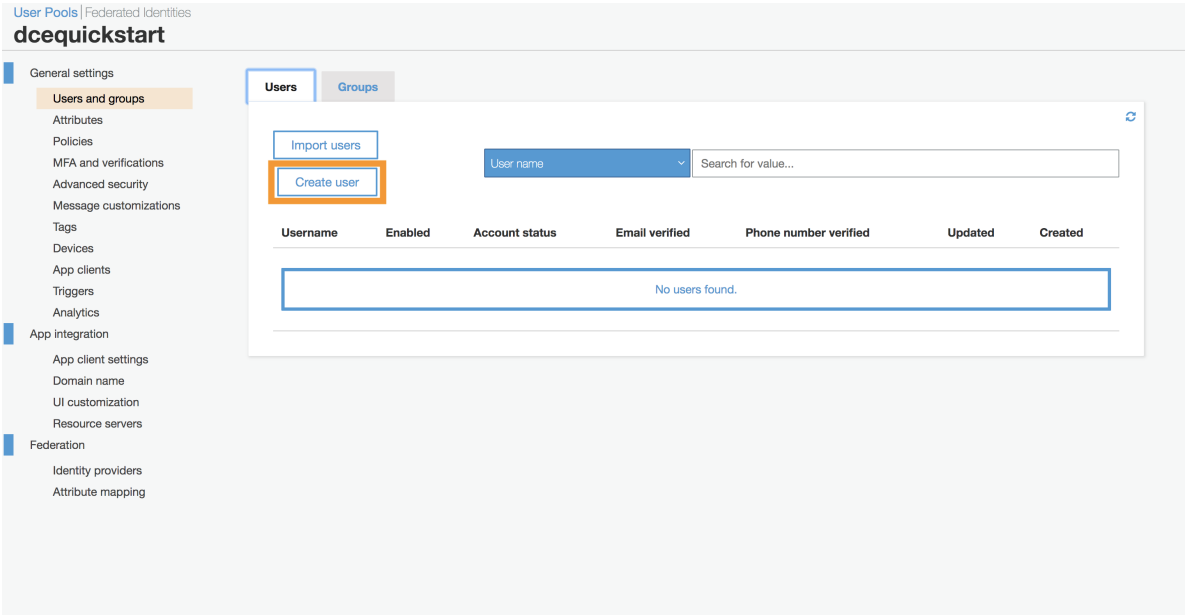
2. Select `Manage User Pools` and click on the dce user pool.



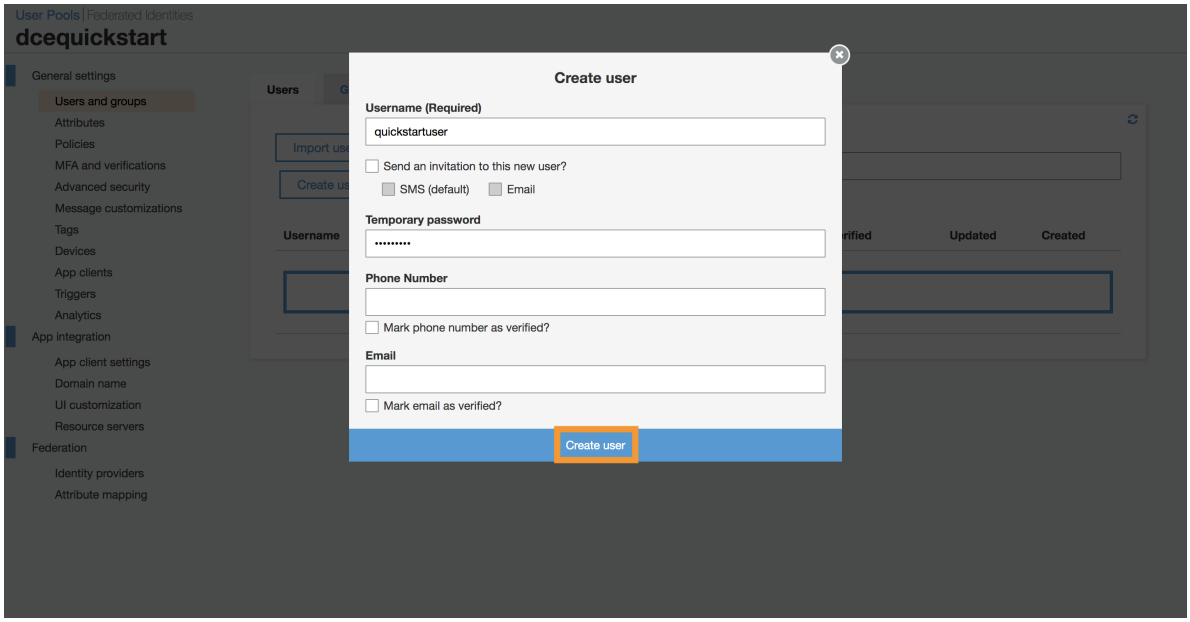
3. Select `Users and groups`



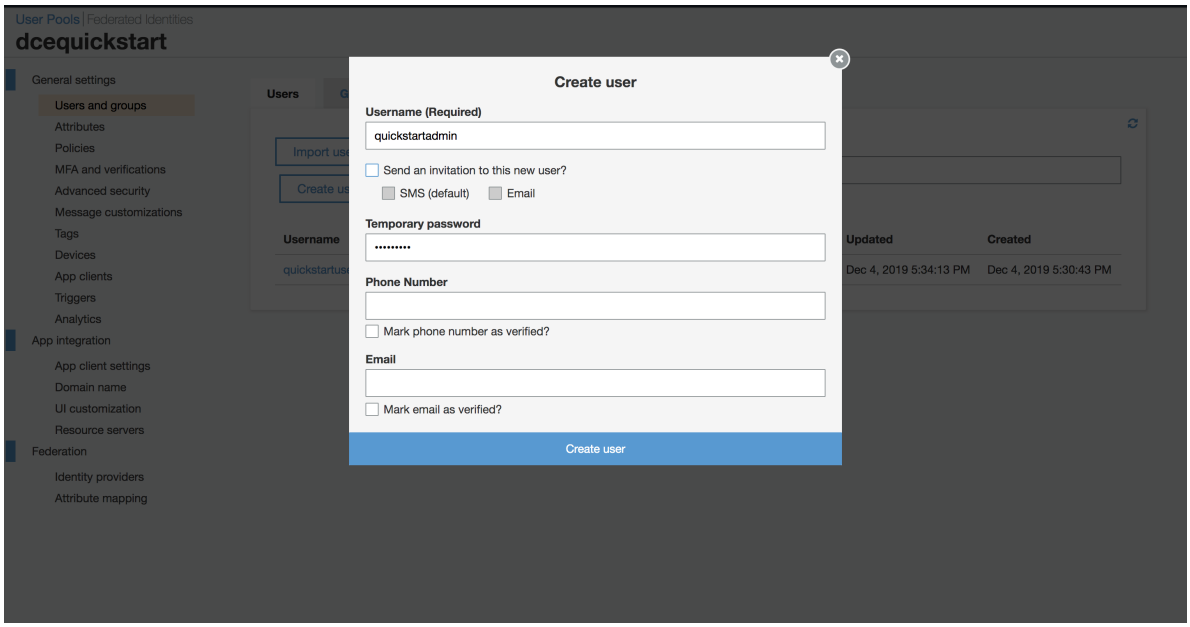
4. Create a user



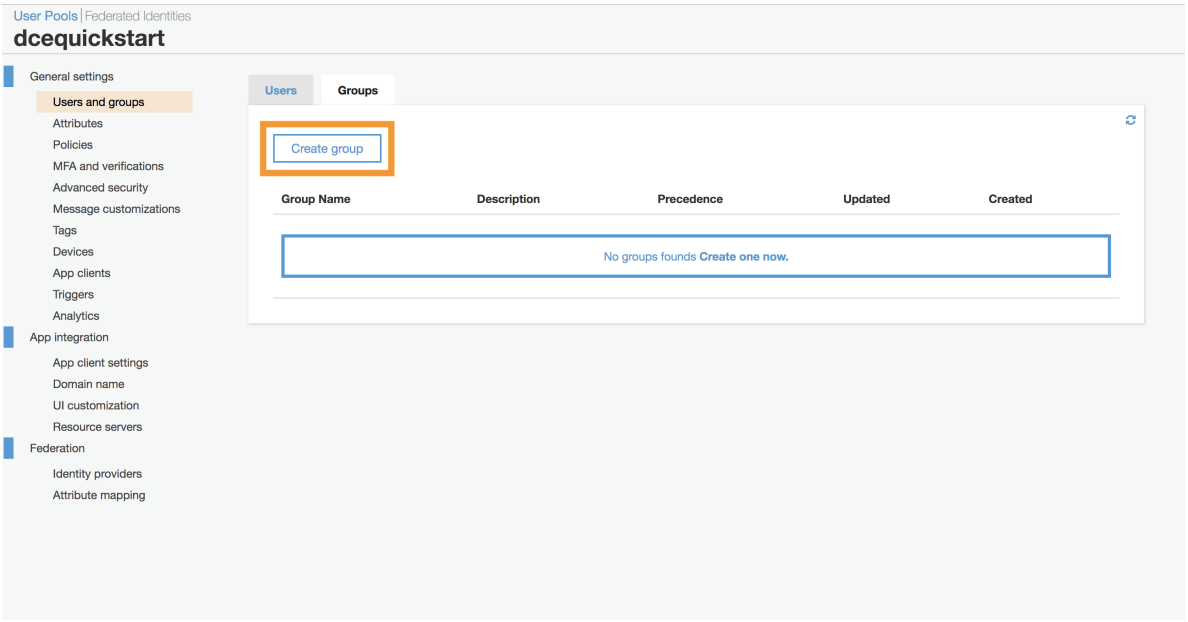
5. Name the user and provide a temporary password. You may uncheck all of the boxes and leave the other fields blank. This user will not have admin priviliges.



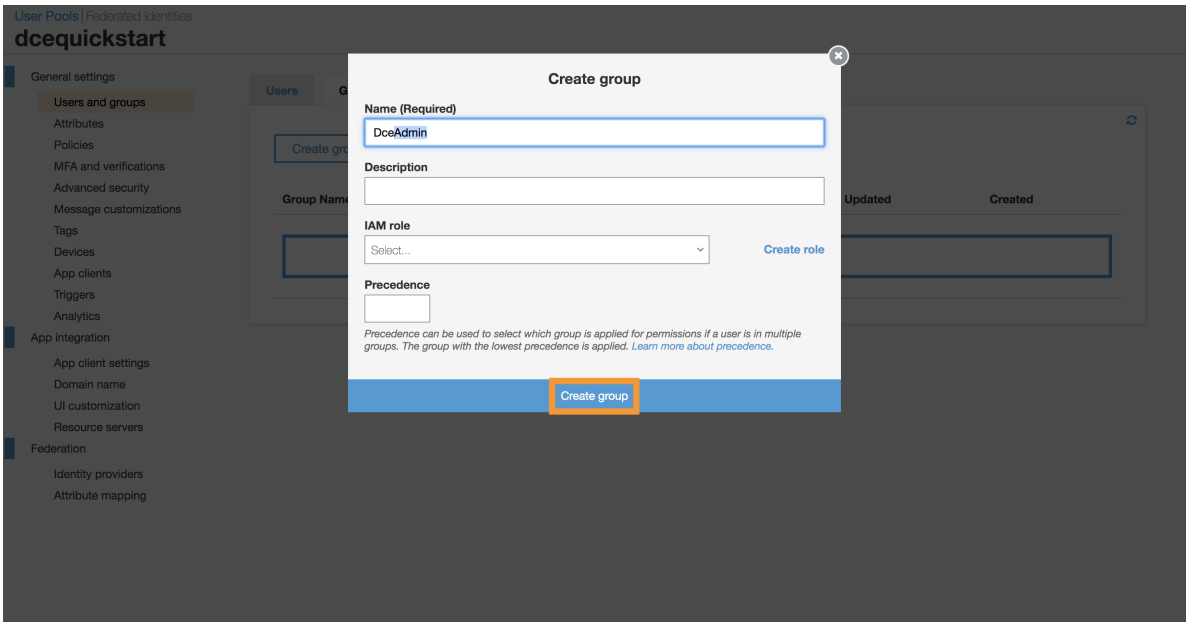
6. Create a second user to serve as a system admin. Follow the same steps as you did for creating the first user, but name this one something appropriate for their role as an administrator.



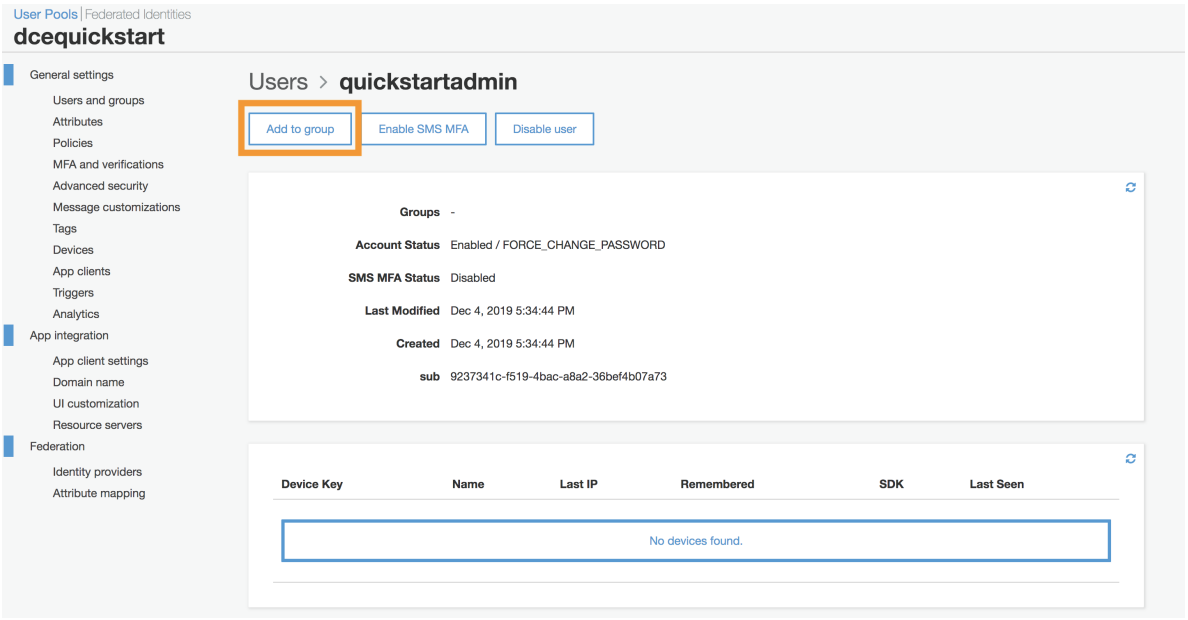
7. Create a group

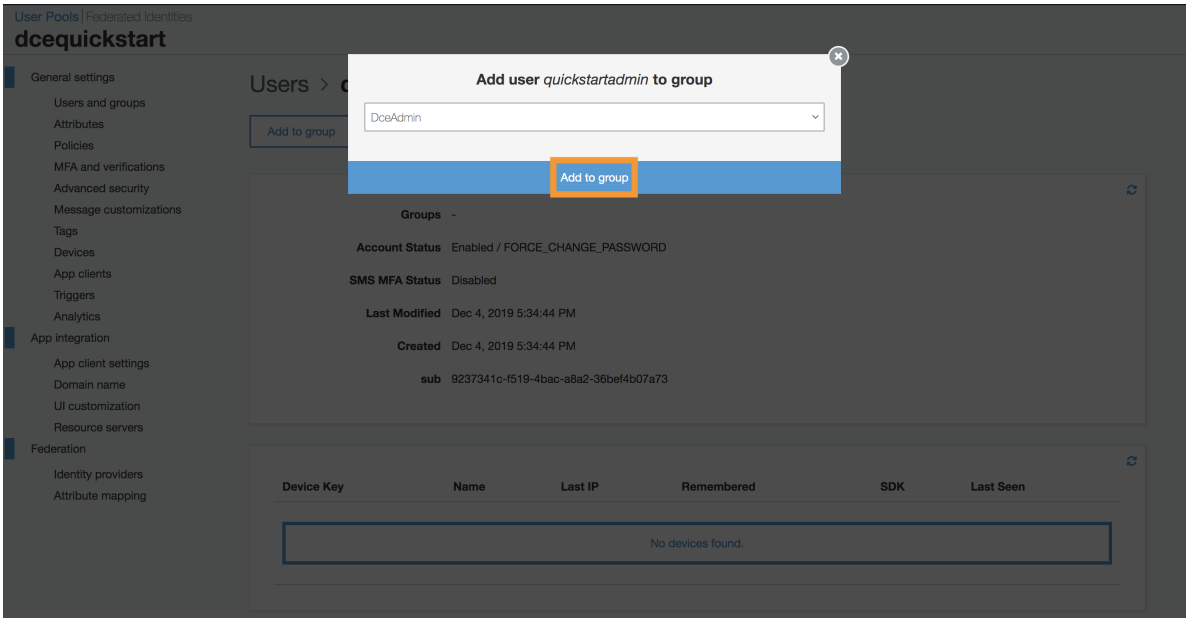


8. Users in this group will be granted admin access to DCE. The group name must contain the term `Admin`. Choose a name and click on the `Create group` button.

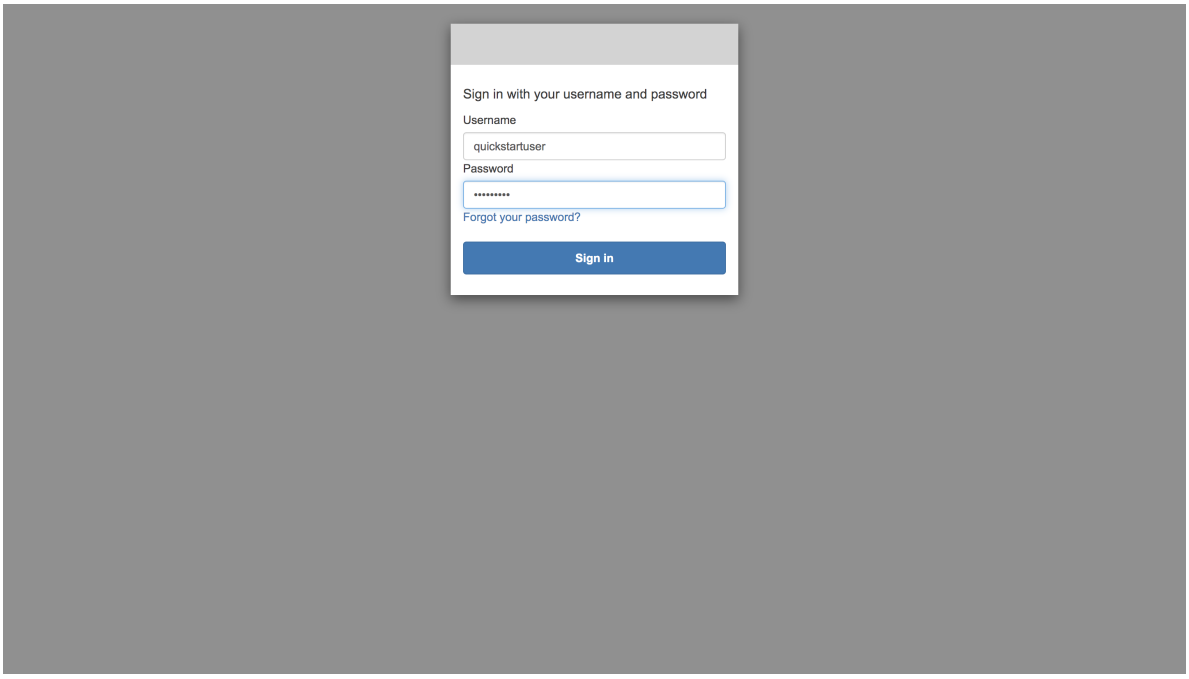


9. Add your admin user to the admin group to grant them admin privileges.

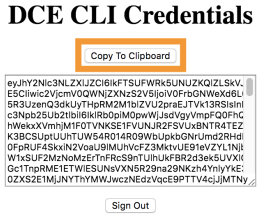




10. Type `dce auth` in your command terminal. This will open a browser with a log in screen. Enter the username and password for the non-admin user that you created. Reset the password as prompted.



11. Upon successfully logging in, you will be redirected to a credentials page containing a temporary authentication code. Click the button to copy the auth code to your clipboard.



12. Return to your command terminal and paste the auth code into the prompt.

```
dce auth
✓ Enter API Token: ::
```

13. You are now authenticated as a DCE User. Test that you have proper authorization by typing `dce leases list`. This will return an empty list indicating that there are currently no leases which you can view. If you are not properly authenticated as a user, you will see a permissions error.

```
dce leases list
[]
```

14. Users are not authorized to list child accounts in the accounts pool. Type `dce accounts list` to verify that you get a permissions error when trying to view information you do not have access to.

```
dce accounts list
err:  [GET /accounts][403] getAccountsForbidden
```

8.1.3 Using IAM Credentials

The DCE API accepts authentication via IAM credentials using SigV4 signed requests.

At minumum, the IAM credentials used to access DCE must have an attached policy that grants permission to invoke the DCE API Gateway methods, e.g.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:Invoke"
      ],
      "Resource": [
        "<DCE API Gateway ARN>"
      ]
    }
  ]
}
JSON
}
```

This policy is accessible via the `api_access_policy_name` and `api_access_policy_arn` terraform outputs.

Any requests made with IAM credentials that have sufficient permissions to invoke the DCE API, but which are not associated with a Congito User Pool User, will be treated as an *admin role*.

The process for signing requests with SigV4 is somewhat involved, but luckily there are a number of tools to make this easier. For example:

- AWS Golang SDK signer/v4 package
- aws-requests-auth for Python
- Postman AWS Signature authentication

AWS also provides examples for a number of languages in their docs.

See DCE CLI Credentials to configure IAM credentials for the DCE CLI.

CHAPTER 9

SNS Lifecycle Events

9.1 SNS Lifecycle Events

The DCE master account publishes messages to a number of SNS topics, to indicate lifecycle events. This allows DCE system administrators to customize their implementation of DCE by subscribing and reacting to these events.

For example, you could setup an *auto-renewal* system by listening to the `lease-removed` SNS topic, and triggering a Lambda that recreates the lease as soon as it expires.

See the Extending Terraform Configuration documentation, for an example of using Terraform to subscribe to DCE SNS topics

9.1.1 account-created

An account was added to the account pool

This SNS topic ARN is provided as a Terraform output:

```
terraform output account_created_topic_arn
```

Payload

This message includes a payload as JSON, with the following fields:

Field	Type	Description
id	string	AWS Account ID
accountStatus	"Ready", "NotReady", "Orphaned", or "Leased"	Account status
adminRoleArn	string	ARN for the IAM role used by the DCE master account to manage the account
lastModifiedOn	int	Last modified timestamp
createdOn	int	Last modified timestamp
metadata	JSON object	Metadata field contains any organization specific data pertaining to the account that needs to be persisted

Example:

```
{
  "id": "1234567890",
  "accountStatus": "NotReady",
  "adminRoleArn": "arn:aws:iam::1234567890123:role/adminRole",
  "principalRoleArn": "arn:aws:iam::1234567890123:role/DCEPrincipal",
  "principalPolicyHash": "\"d41d8cd98f00b204e9800998ecf8427e-38\"",
  "createdOn": 1560306008,
  "lastModifiedOn": 1560306008,
  "metadata": {}
}
```

9.1.2 account-deleted

An account was deleted from the account pool

This SNS topic ARN is provided as a Terraform output:

```
terraform output account_deleted_topic_arn
```

Payload

This message includes a payload as JSON, with the following fields:

Field	Type	Description
id	string	AWS Account ID
accountStatus	"Ready", "NotReady", "Orphaned", or "Leased"	Account status
adminRoleArn	string	ARN for the IAM role used by the DCE master account to manage the account
lastModifiedOn	int	Last modified timestamp
createdOn	int	Last modified timestamp
metadata	JSON object	Metadata field contains any organization specific data pertaining to the account that needs to be persisted

Example:

```
{
  "id": "1234567890",
  "accountStatus": "NotReady",
  "adminRoleArn": "arn:aws:iam::1234567890123:role/adminRole",
  "principalRoleArn": "arn:aws:iam::1234567890123:role/DCEPrincipal",
  "principalPolicyHash": "\"d41d8cd98f00b204e9800998ecf8427e-38\"",
  "createdOn": 1560306008,
  "lastModifiedOn": 1560306008,
  "metadata": {}
}
```

9.1.3 lease-added

Triggered when a lease is created.

This SNS topic ARN is provided as a Terraform output:

```
terraform output lease_added_topic_arn
```

Payload

This message includes a payload as JSON, with the following fields:

Field	Type	Description
accountId	string	AWS Account ID
principalId	string	ID of the principal user, associated with the lease
leaseStatus	string	Status of the lease.
createdOn	integer	Timestamp (epoch) of creation
lastModifiedOn	integer	Timestamp (epoch) of last modification
leaseModifiedOn	integer	Timestamp (epoch) of lease status modification
expiresOn	integer	Timestamp (epoch) when the lease will expire

Example:

```
{
  "accountId": "1234567890",
  "principalId": "jdoe17",
  "leaseStatus": "Active",
  "createdOn": 1560306008,
  "lastModifiedOn": 1560306008,
  "leaseStatusModifiedOn": 1560306008,
  "expiresOn": 1560306008
}
```

9.1.4 lease-removed

Triggered when a lease is deleted.

This SNS topic ARN is provided as a Terraform output:

```
terraform output lease_removed_topic_arn
```

Payload

This message includes a payload as JSON, with the following fields:

Field	Type	Description
accountId	string	AWS Account ID
principalId	string	ID of the principal user associated with the lease
leaseStatus	string	Status of the lease.
createdOn	integer	Timestamp (epoch) of creation
lastModifiedOn	integer	Timestamp (epoch) of last modification
leaseStatusModifiedOn	integer	Timestamp (epoch) of last lease status modification
expiresOn	integer	Timestamp (epoch) when the lease will expire

Example:

```
{
  "accountId": "1234567890",
  "principalId": "jdoe17",
  "leaseStatus": "Decommissioned",
  "createdOn": 1560306008,
  "lastModifiedOn": 1560306008,
  "leaseStatusModifiedOn": 1560306008,
  "expiresOn": 1560306008
}
```

CHAPTER 10

Permissions and Policies

10.1 Policies and Permissions

10.1.1 Principal Role Policy

Users access their leased accounts through an assumed role. This role also restricts their privileges within their leased account. The policy is defined here. This policy is designed to protect the IAM principal policy and trusts so that DCE can continue to manage the account. Additionaly the policy is designed around services that AWS Nuke supports.

10.1.2 Organizations and Service Control Policies (SCPs)

Implementing DCE in an AWS Organization provides the ability to use SCPs, which can be helpful for ensuring the resilience of your DCE resources. The following SCP is an example policy that contains two statements for protecting your DCE accounts:

- **DenyChangesToAdminPrincipalRoleAndPolicy** is designed to prevent anyone other than the AdminRole from modifying the roles and policies used by DCE.
- **DenyUnsupportedServices** is designed to allow access to services that are supported by AWS Nuke

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyChangesToAdminPrincipalRoleAndPolicy",
            "Effect": "Deny",
            "NotAction": [
                "iam:GetContextKeysForPrincipalPolicy",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListAttachedRolePolicies",
                "iam:ListInstanceProfilesForRole",
                "iam:ListRolePolicies",
```

(continues on next page)

(continued from previous page)

```
                "iam:ListRoleTags",
                "iam:DeactivateMFADevice",
                "iam:CreateSAMLProvider",
                "iam:UpdateAccountPasswordPolicy",
                "iam:DeleteVirtualMFADevice",
                "iam:EnableMFADevice",
                "iam:CreateAccountAlias",
                "iam:DeleteAccountAlias",
                "iam:UpdateSAMLProvider",
                "iam:DeleteSAMLProvider"
            ],
            "Resource": [
                "arn:aws:iam::*:role/AdminRole",
                "arn:aws:iam::*:role/DCEPrincipal*",
                "arn:aws:iam::*:policy/DCEPrincipal*"
            ],
            "Condition": {
                "StringNotLike": {
                    "aws:PrincipalARN": "arn:aws:iam::*:role/AdminRole"
                }
            }
        },
        {
            "Sid": "DenyUnsupportedServices",
            "Effect": "Deny",
            "NotAction": [
                "acm:*",
                "acm-pca:*",
                "apigateway:*",
                "application-autoscaling:*",
                "appstream:*",
                "athena:*",
                "autoscaling:*",
                "backup:*",
                "batch:*",
                "cloud9:*",
                "clouddirectory:*",
                "cloudformation:*",
                "cloudfront:*",
                "cloudhsm:*",
                "cloudsearch:*",
                "cloudtrail:*",
                "cloudwatch:*",
                "codebuild:*",
                "codecommit:*",
                "codedeploy:*",
                "codepipeline:*",
                "codestar:*",
                "cognito-identity:*",
                "cognito-idp:*",
                "comprehend:*",
                "config:*",
                "datapipeline:*",
                "dax:*",
                "devicefarm:*",
                "dms:*",
                "ds:*",
```

(continues on next page)

(continued from previous page)

```
                "dynamodb:*",
                "ec2:*",
                "ecr:*",
                "ecs:*",
                "eks:*",
                "elasticache:*",
                "elasticbeanstalk:*",
                "elasticfilesystem:*",
                "elasticloadbalancing:*",
                "elasticmapreduce:*",
                "elastictranscoder:*",
                "es:*",
                "events:*",
                "execute-api:*",
                "firehose:*",
                "fsx:*",
                "globalaccelerator:*",
                "glue:*",
                "iam:*",
                "imagebuilder:*",
                "iot:*",
                "iotanalytics:*",
                "kafka:*",
                "kinesis:*",
                "kinesisanalytics:*",
                "kinesisvideo:*",
                "kms:*",
                "lakeformation:*",
                "lambda:*",
                "lex:*",
                "lightsail:*",
                "logs:*",
                "machinelearning:*",
                "mediaconvert:*",
                "medialive:*",
                "mediapackage:*",
                "mediastore:*",
                "mediatailor:*",
                "mobilehub:*",
                "mq:*",
                "neptune-db:*",
                "opsworks:*",
                "opsworks-cm:*",
                "rds:*",
                "redshift:*",
                "rekognition:*",
                "resource-groups:*",
                "robomaker:*",
                "route53:*",
                "s3:*",
                "sagemaker:*",
                "secretsmanager:*",
                "servicecatalog:*",
                "servicediscovery:*",
                "ses:*",
                "sns:*",
                "sqs:*",
```

(continues on next page)

(continued from previous page)

```
                "ssm:*",
                "states:*",
                "storagegateway:*",
                "sts:*",
                "tag:*",
                "transfer:*",
                "waf:*",
                "wafv2:*",
                "waf-regional:*",
                "worklink:*",
                "workspaces:*"
            ],
            "Resource": "*"
        }
    ]
}
```

CHAPTER 11

Account Cleanup with AWS Nuke

11.1 Account Cleanup with AWS Nuke

DCE uses a fork of AWS Nuke to facilitate account cleanup. Shown here is a list of services that are supported and those that are not supported for account cleanup.

11.1.1 Supported Services

- AWS Backup
- AWS Batch
- AWS Certificate Manager
- AWS Certificate Manager Private Certificate Authority
- AWS Cloud Map
- AWS Cloud9
- AWS CloudFormation
- AWS CloudHSM
- AWS CloudTrail
- AWS CodeBuild
- AWS CodeCommit
- AWS CodeDeploy
- AWS CodePipeline
- AWS CodeStar
- AWS Config
- AWS Database Migration Service

- AWS Device Farm
- AWS Directory Service
- AWS Elastic Beanstalk
- AWS Elemental MediaConvert
- AWS Elemental MediaLive
- AWS Elemental MediaPackage
- AWS Elemental MediaStore
- AWS Elemental MediaTailor
- AWS Glue
- AWS IoT
- AWS Key Management Service
- AWS Lambda
- AWS Mobile Hub
- AWS OpsWorks
- AWS OpsWorks Configuration Management
- AWS Resource Groups
- AWS RoboMaker
- AWS Secrets Manager
- AWS Service Catalog
- AWS Step Functions
- AWS Systems Manager
- AWS WAF
- Amazon AppStream 2.0
- Amazon Athena
- Amazon Cloud Directory
- Amazon CloudFront
- Amazon CloudSearch
- Amazon CloudWatch
- Amazon CloudWatch Logs
- Amazon Cognito Identity
- Amazon Cognito User Pools
- Amazon DynamoDB
- Amazon DynamoDB Accelerator (DAX)
- Amazon EC2
- Amazon EC2 Auto Scaling
- Amazon ElastiCache

- Amazon Elastic Container Registry
- Amazon Elastic Container Service
- Amazon Elastic Container Service for Kubernetes
- Amazon Elastic File System
- Amazon Elastic MapReduce
- Amazon Elastic Transcoder
- Amazon Elasticsearch Service
- Amazon EventBridge
- Amazon FSx
- Amazon Kinesis
- Amazon Kinesis Analytics
- Amazon Kinesis Analytics V2
- Amazon Kinesis Firehose
- Amazon Kinesis Video Streams
- Amazon Lightsail
- Amazon MQ
- Amazon Machine Learning
- Amazon Managed Streaming for Kafka
- Amazon Pinpoint Email Service
- Amazon RDS
- Amazon Redshift
- Amazon Rekognition
- Amazon Route 53
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS
- Amazon SageMaker
- Amazon Storage Gateway
- Amazon WorkLink
- Amazon WorkSpaces
- Data Pipeline
- Elastic Load Balancing
- Elastic Load Balancing V2
- Identity And Access Management

11.1.2 Unsupported Services

- AWS Accounts
- AWS Amplify
- AWS App Mesh
- AWS App Mesh Preview
- AWS AppConfig
- AWS AppSync
- AWS Artifact
- AWS Auto Scaling
- AWS Backup storage
- AWS Billing
- AWS Budget Service
- AWS Chatbot
- AWS Code Signing for Amazon FreeRTOS
- AWS CodeStar Notifications
- AWS Connector Service
- AWS Cost Explorer Service
- AWS Cost and Usage Report
- AWS Data Exchange
- AWS DeepLens
- AWS DeepRacer
- AWS Direct Connect
- AWS Elemental MediaConnect
- AWS Elemental MediaPackage VOD
- AWS Firewall Manager
- AWS Global Accelerator
- AWS Ground Station
- AWS Health APIs and Notifications
- AWS IQ
- AWS IQ Permissions
- AWS Import Export Disk Service
- AWS IoT 1-Click
- AWS IoT Analytics
- AWS IoT Device Tester
- AWS IoT Events
- AWS IoT Greengrass

- AWS IoT SiteWise
- AWS IoT Things Graph
- AWS Lake Formation
- AWS License Manager
- AWS Managed Apache Cassandra Service
- AWS Marketplace
- AWS Marketplace Catalog
- AWS Marketplace Entitlement Service
- AWS Marketplace Image Building Service
- AWS Marketplace Management Portal
- AWS Marketplace Metering Service
- AWS Marketplace Procurement Systems Integration
- AWS Migration Hub
- AWS Organizations
- AWS Outposts
- AWS Performance Insights
- AWS Price List
- AWS Private Marketplace
- AWS Resource Access Manager
- AWS SSO
- AWS SSO Directory
- AWS Savings Plans
- AWS Security Hub
- AWS Security Token Service
- AWS Server Migration Service
- AWS Serverless Application Repository
- AWS Shield
- AWS Snowball
- AWS Support
- AWS Transfer for SFTP
- AWS Trusted Advisor
- AWS WAF Regional
- AWS WAF V2
- AWS Well-Architected Tool
- AWS X-Ray
- Alexa for Business

- Amazon API Gateway
- Amazon Chime
- Amazon CloudWatch Synthetics
- Amazon CodeGuru Profiler
- Amazon CodeGuru Reviewer
- Amazon Cognito Sync
- Amazon Comprehend
- Amazon Connect
- Amazon Data Lifecycle Manager
- Amazon Detective
- Amazon EC2 Image Builder
- Amazon EC2 Instance Connect
- Amazon Elastic Block Store
- Amazon Elastic Inference
- Amazon EventBridge Schemas
- Amazon Forecast
- Amazon Fraud Detector
- Amazon FreeRTOS
- Amazon GameLift
- Amazon Glacier
- Amazon GroundTruth Labeling
- Amazon GuardDuty
- Amazon Inspector
- Amazon Kendra
- Amazon Lex
- Amazon Macie
- Amazon Managed Blockchain
- Amazon Mechanical Turk
- Amazon Message Delivery Service
- Amazon Mobile Analytics
- Amazon Neptune
- Amazon Personalize
- Amazon Pinpoint
- Amazon Pinpoint SMS and Voice Service
- Amazon Polly
- Amazon QLDB

- Amazon QuickSight
- Amazon RDS Data API
- Amazon RDS IAM Authentication
- Amazon Resource Group Tagging API
- Amazon Route 53 Resolver
- Amazon Route53 Domains
- Amazon Session Manager Message Gateway Service
- Amazon Simple Workflow Service
- Amazon SimpleDB
- Amazon Sumerian
- Amazon Textract
- Amazon Transcribe
- Amazon Translate
- Amazon WorkDocs
- Amazon WorkMail
- Amazon WorkMail Message Flow
- Amazon WorkSpaces Application Manager
- Application Auto Scaling
- Application Discovery
- Application Discovery Arsenal
- CloudWatch Application Insights
- Comprehend Medical
- Compute Optimizer
- DataSync
- Database Query Metadata Service
- IAM Access Analyzer
- Launch Wizard
- Manage Amazon API Gateway
- Network Manager
- Service Quotas

CHAPTER 12

Local Development

12.1 Local Development

This page will guide you through some basic points for getting started developing against the DCE codebase.

Note: unless otherwise noted, all commands shown here should be executed from the DCE base directory

12.1.1 Configuring your development environment

You may find development easiest on a Mac OS or Linux-based machine. Development should be possible on Windows 10 with the Windows Subsystem for Linux installed, but at the time of this writing has not been verified.

You will need the following:

1. Go (version 1.13.x)
2. Terraform (version v0.12.x)
3. GNU make (version 3.x)
4. GNU bash, which is used for shell scripts
5. An AWS account for deploying resources
6. The AWS CLI *Note: if you install version 2, see "Configuring AWS CLI 2"*
7. An AWS IAM user with command line access.
8. Git (version 2.x)

Important: deploying DCE to your AWS account can incur cost.

Configuring AWS CLI 2

The AWS CLI version 2 includes a breaking change that creates problems with the automation scripts. See https://docs.aws.amazon.com/cli/latest/userguide/cliv2-migration.html for more information.

For DCE, the recommeonded solution to this problem is to add the following line in your `~/aws/config` file:

```
[default]
cli_pager=
```

Getting the code locally

To get the code locally fork this repo (https://github.com/Optum/dce) and then clone the repository:

```
git clone https://github.com/${mygithubid}/dce.git dce # replace with your fork's␣
↪HTTPS URL
cd dce
make setup
```

The last command, `make setup`, will run the `scripts/install_ci.sh` script which will install the necessary tools for building and testing the project.

12.1.2 Code Structure

The DCE codebase is comprised of Go application code, along with Terraform infrastructure configuration.

The Go code is primarily located within:

- /cmd: entrypoint for applications targeting AWS Lambdas and CodeBuild
- /pkg: common services used by entrypoint code.

Each subdirectory within the /cmd/lambda directory targets an individual Lambda function of the same name.

12.1.3 Building application code

To compile the Go application code, run:

```
make build
```

This generates a `/bin/build_artifacts.zip` file, which includes Go binaries for each entrypoint application.

12.1.4 Unit Tests

Unit tests are located within the `/cmd` and `/pkg` directories, adjacent to their corresponding Go code. So, for example, the code in `/pkg/api/user_test.go` includes tests against `/pkg/api/user.go`.

Execute unit tests by running:

```
make test
```

12.1.5 Code Linting

When you run `make test`, the `lint` target is executed automatically. You can, however, run the linting by itself by using the command:

```
make lint
```

During `make lint`, the `scripts/lint.sh` script executes golangci-lint. The configuration file is `.golangci.yml`. Enabled linters and rule exceptions can be found in this file.

The `make lint` target also executes tflint to lint the terraform code found in `modules`.

12.1.6 Functional Tests

Functional tests are located in `tests` and are used to test the integration between a number of services or verify that end-to-end behavior is working properly. For example, we rely heavily on functional tests for DynamoDB interactions, to verify that we are using the DynamoDB SDKs correctly.

Before running functional tests, DCE must be deployed to a test AWS account. **Functional tests truncate the database tables, so do not run them against production environments.**

To deploy DCE for testing, first authenticate against an AWS test account, then run:

```
# Deploy AWS infrastructure using Terraform
cd modules
terraform init
terraform apply

# Deploy application code to AWS
cd ..
make deploy
```

See Deploying DCE With Terraform for more details.

To run functional tests:

```
make test_functional
```

Functional tests load the details of the DCE deployment from Terraform module outputs, so there is no need for additional configuration to run functional tests.

12.1.7 Before committing code

The `make test` target is used by continuous integration build. A failure of the target will cause the build to fail, so before committing code or creating a pull request you should run the following commands:

```
make build
make test
```

12.1.8 Building the documentation

The documentation is located in `docs` and is based on the Sphinx project and hosted on http://readthedocs.io.

If you are making changes to documenation and would like to verify the build of the documentation, you will need to make sure Python 3 is installed. It is *highly recommended* that you use `virtualenv` and configure your workspace with the commands shown here:

```
virtualenv -p python3 ENV
source ENV/bin/activate
pip install -r docs/requirements.txt
```

With the Python requirements installed and the virtualenv sourced, use the following command from the base project directory:

```
make documentation
```

To serve the documentation locally, run the following command:

```
make serve_docs
```

By default, the documenation will be served at `http://127.0.0.1:8000`.