LOR Link Checker



Contents

[Document Change Control 3](#_Toc505448816)

[High-Level Solution Overview 3](#_Toc505448817)

[Access Control 3](#_Toc505448818)

[Entity Relationship Diagram 4](#_Toc505448819)

[Web Application Modules 5](#_Toc505448820)

[Admin 5](#_Toc505448821)

[Sources 5](#_Toc505448822)

[Settings 6](#_Toc505448823)

[Users 7](#_Toc505448824)

[Roles 8](#_Toc505448825)

[Report 9](#_Toc505448826)

[Invalid Links 9](#_Toc505448827)

[Warning Links 9](#_Toc505448828)

[Application Modules 10](#_Toc505448829)

[ObjectExtractor 10](#_Toc505448830)

[LinkExtractor 11](#_Toc505448831)

[Processor 11](#_Toc505448832)

[Queue 11](#_Toc505448833)

[Extractor 11](#_Toc505448834)

[LinkChecker 12](#_Toc505448835)

[Processor 12](#_Toc505448836)

[Queue 12](#_Toc505448837)

[Checker 12](#_Toc505448838)

[Packages 13](#_Toc505448839)

[Appendix A: Project Setup /w AWS 14](#_Toc505448840)

[Appendix B: Settings Available 16](#_Toc505448841)

# Document Change Control

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | Name | Date | Description |
| Phase 1 | Brian Merrick | 12-17-2016 | Finished Phase 1 |
| Phase 2 | Brian Merrick | 3-19-2017 | Big dent in Phase 2 |
| Phase 3 | Brian Merrick | 5-18-2017 | Finished Phase 2 |
| Phase 4 | Brian Merrick | 1-27-2018 | Converted to Serverless |

# High-Level Solution Overview

The LOR Link Checker is composed of several applications as well as a web interface. The ObjectExtractor will get a list of objects in the specified buckets/prefixes. They will be added to the LLC database when new and if they already are in the database, the date last found will be updated. The LinkExtractor extracts URLs from source locations (such as S3 objects) and stores them in the database for further analysis. The LinkChecker goes through the objects identified from the LinkExtractor and determines if they return valid responses or are broken in any way. The web interface allows management of settings, users, roles and view logs and reports. The LLCPackage application processes the links from user uploaded packages that follow the IMSCC format.

# Access Control

|  |  |
| --- | --- |
| Role | Description |
| Admin | Administrator of the site. Can manage settings, users and roles. |
| ReportUser | Allowed to view reports available from the various applications in the project. |

# Entity Relationship Diagram

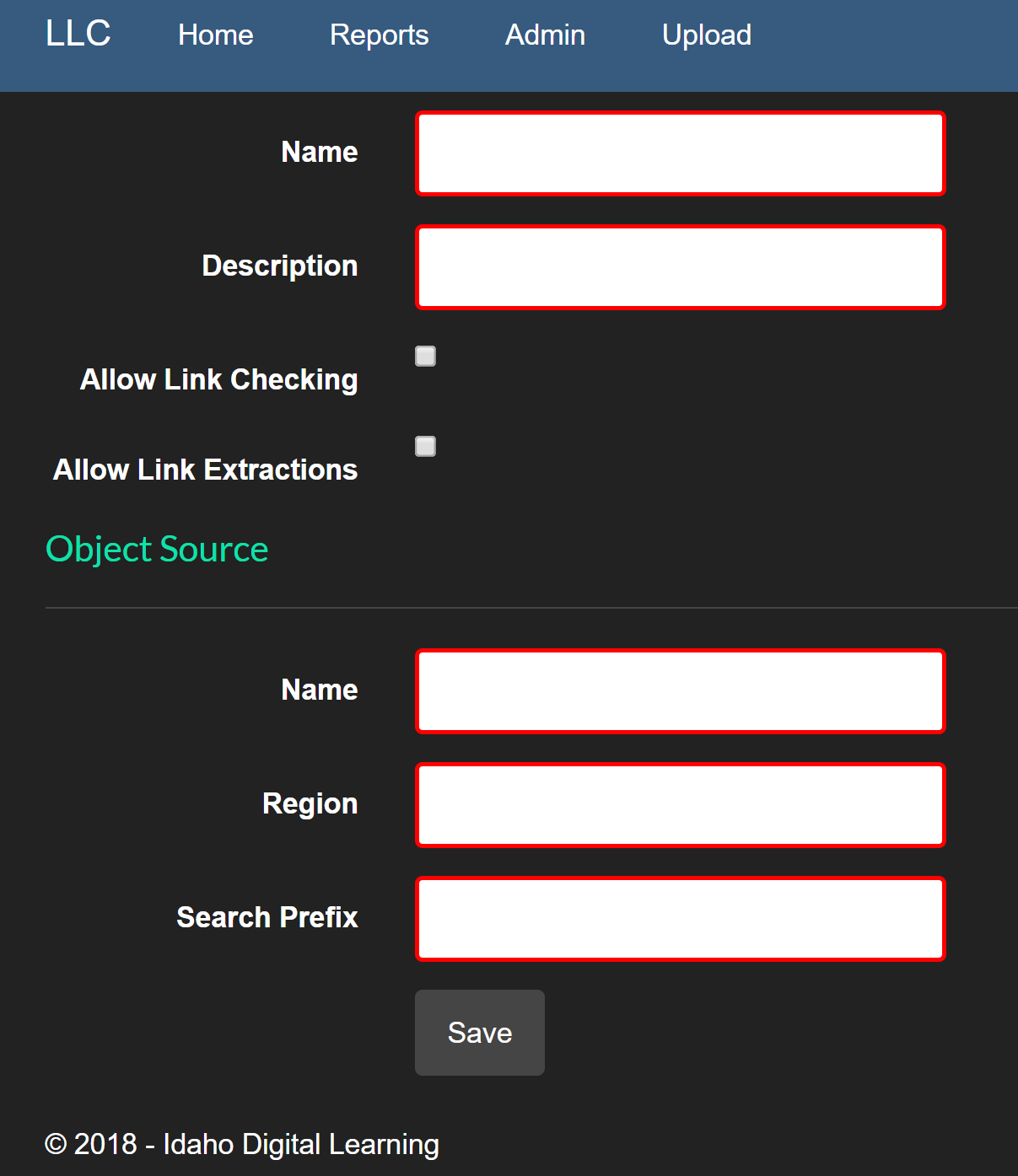
# Web Application Modules

## Admin

The administrative module contains all the features needed to administer the application which includes sources and settings.

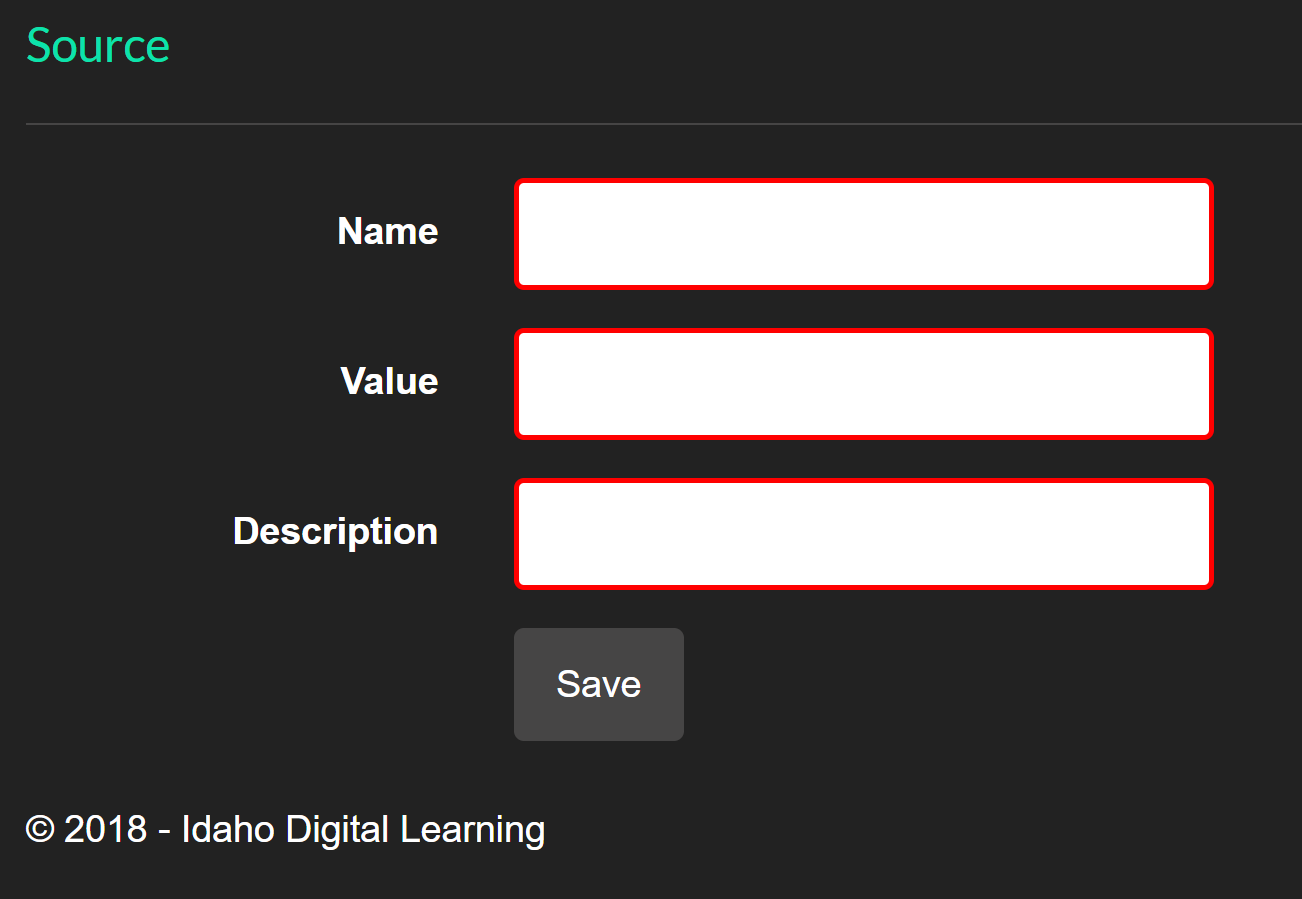
### Sources

This page allows administrators to add/edit/delete sources. A *source* currently is only defined as an S3 object. The create option allows for the following inputs. The boxes with red are required fields.

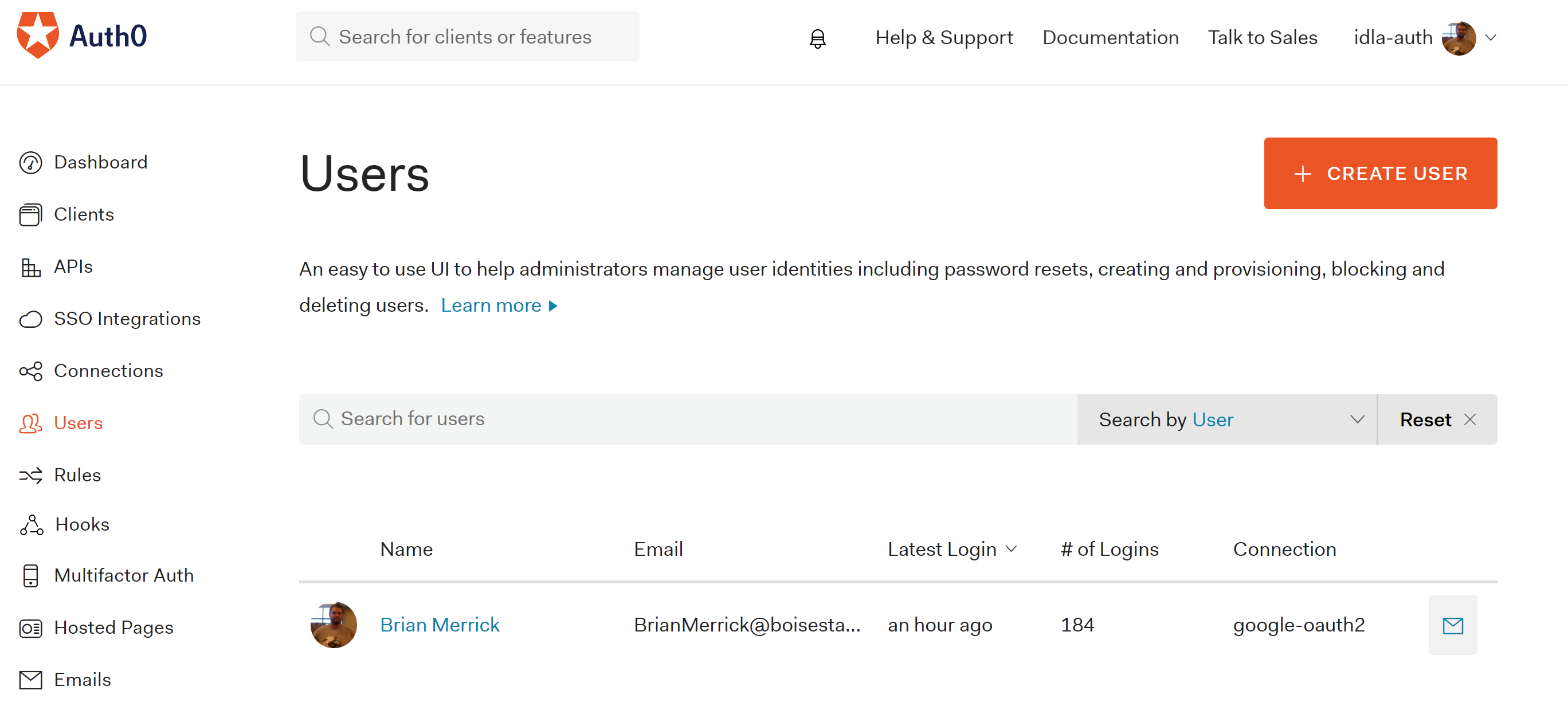


### Settings

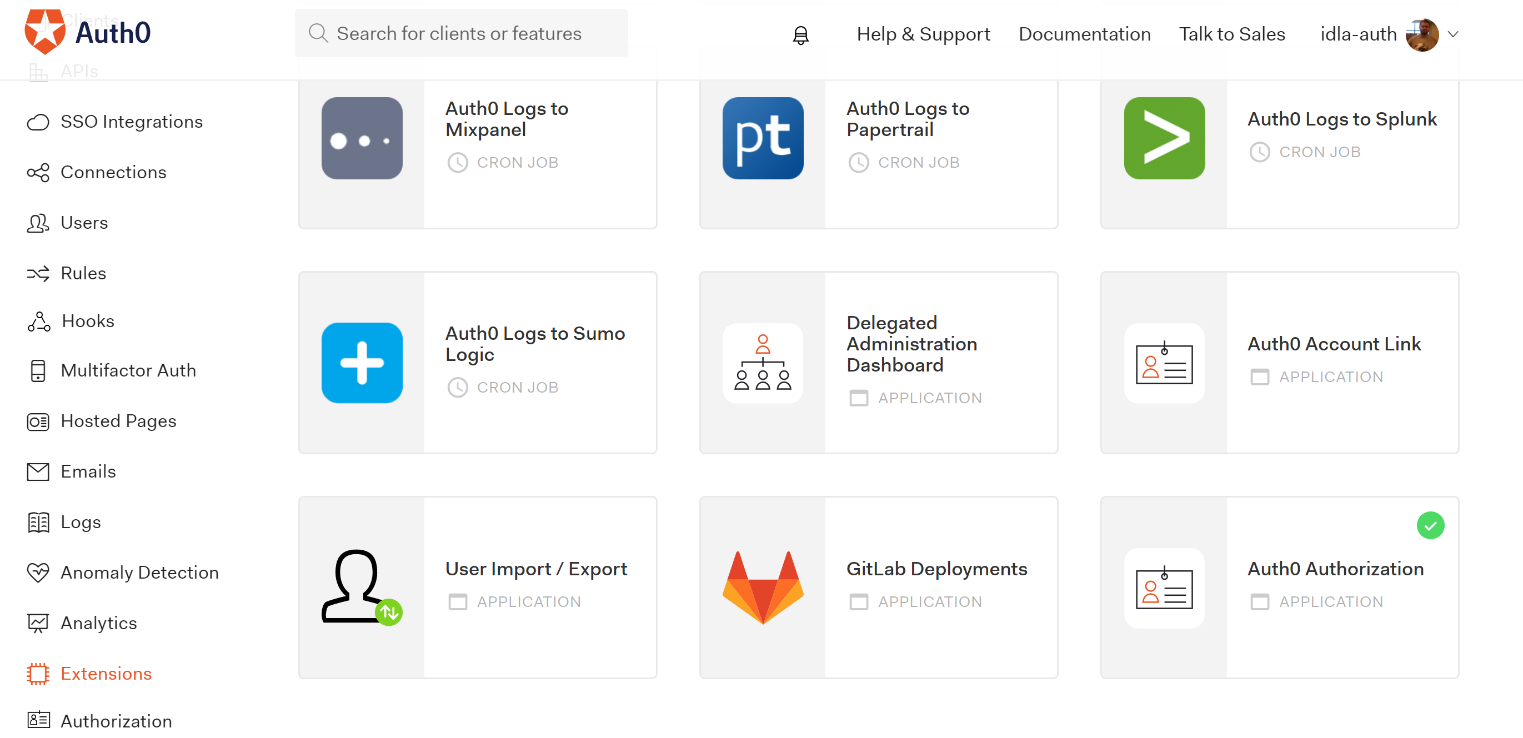
The list of settings found in [Appendix B](#_Appendix_B:_Settings) are the only settings that are supported in the application. The settings screen allows the administrators to add/edit/remove settings. The screen below shows the create options for new/edit settings.

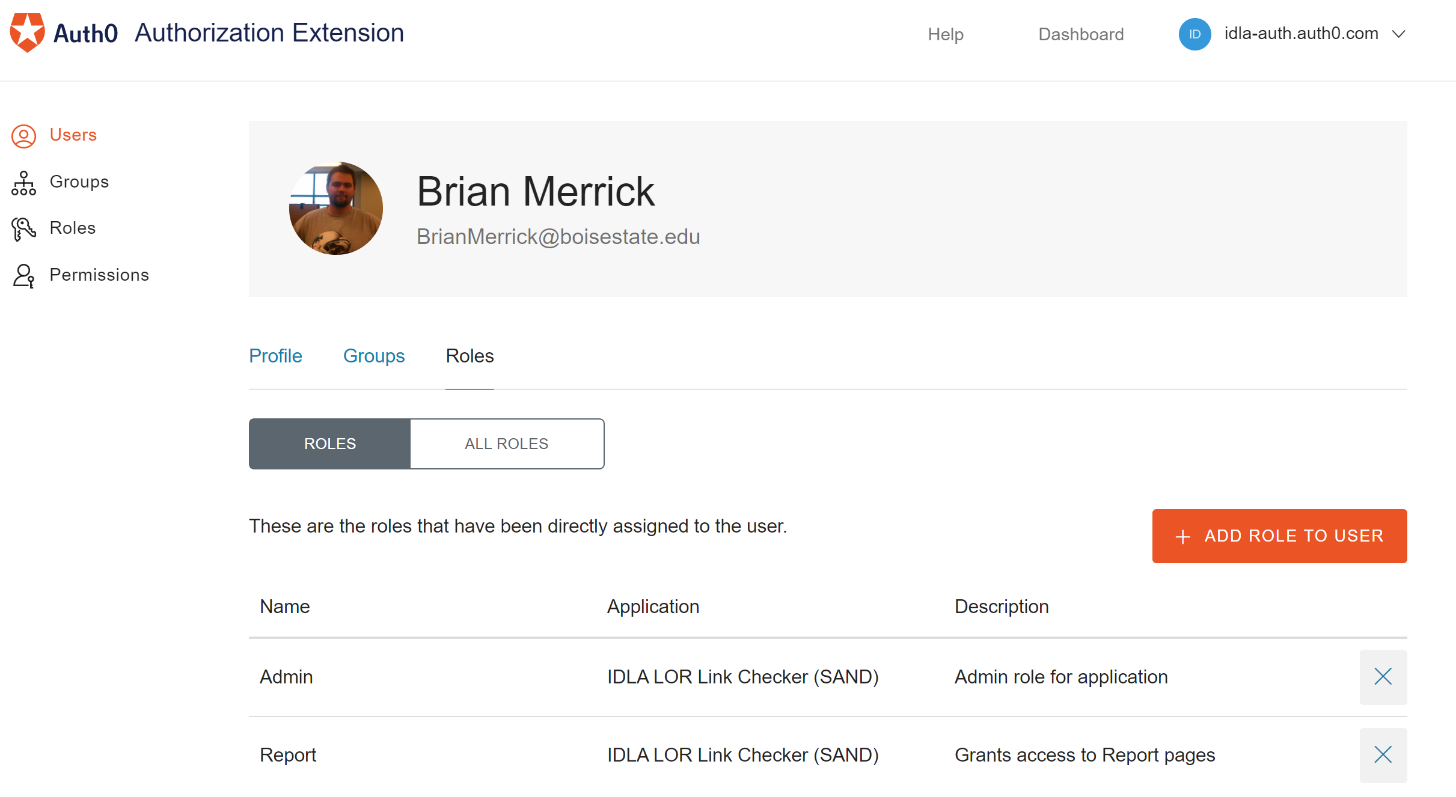


### Users

Users are handled through [Auth0](https://manage.auth0.com). New users will authenticate through the application. If they are a member of the correct Google domain, they will be allowed through. Otherwise, they are denied. There is no need to create users. They must first login and then a role can be assigned to them.

### Roles

Administrators with Auth0 access can add/edit/remove system roles. To do so, you navigate to the Extensions tab to get to the Auth0 Authorization application within Auth0.

By Clicking on the user, you can manage their roles and manage roles for the applications.

## Report

The reports interface gives admins and report users the ability to see detailed data from the application. Currently two reports exist, the **Invalid Links** and **Warning Links** reports.

### Invalid Links

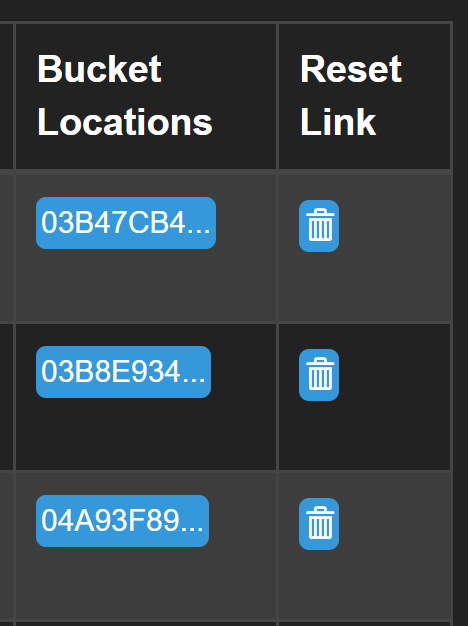
This report returns all links that have an invalid state from the last time checked. It will show the source, URL, attempt count, date last checked, and date last found.

Any link that was in an error state will be updated with an **Valid** flag set to false (0). If an admin hits the **Reset Link**option on the report, the **Valid** flag will reset to 1 (and make it valid).

### Warning Links

The warning links report is updated every time the [LinkChecker](#_LinkChecker) application runs. It will check links from the settings that are less than **LinkCheckDays**old and within **LinkCheckStandardDeviations**. If the mean of those links is outside of that number of standard deviations, the last link is added to the report. Otherwise, it is removed from the report since it corrected itself.

By default, all **Stats** will be considered in the calculated of the standard deviation. If an admin hits the **Reset Link** option on the **Report**, the **ReportNotBeforeDate** on the **Links** will be updated with the current date. This means that no **Stats** before this date will be considered in the standard deviation calculation and the current **Reports** entry will be removed.



# Application Modules

The applications perform most the work of the LLC application. They are all written using AWS Lambda and dotnetcore 1.0 (Visual Studio SAM possibly supports dotnetcore 2.x now). The web interface is simply the mechanism to view the results. The available applications are explained below.

## ObjectExtractor

This application will get a list of objects in specified buckets/prefixes and will be added to the LLC database. New items will be added, and existing items will have their date last found updated to the current time. This application has an S3 trigger that is attached to the S3 bucket(s). Whenever a new bucket is added, the Lambda function (with ObjectExtractor in the name) trigger needs updated on the new bucket and the new source added in the application.

The ***<stack name>-LLCServerless-ObjectExtractor-<guid>*** Lambda function will be attached to all object bucket(s). When an item is added to the bucket, the trigger will fire the function and it will attempt to parse the file and find any hyperlinks inside of them to add to the LLC Database. The following meta-data is supported in the application:

* mobile\_pages – if the key contains the text ***mobile\_pages***, it will be skipped
* x-amz-meta-linkcheck: exclude – if the meta ***linkcheck*** exists on the object, it will be skipped

The diagram below shows the flow of data.



## LinkExtractor

This application extracts URLs from source locations (such as S3 objects) and stores the download information in the database for further analysis. This application is made up of three parts to work in AWS. The three parts are:

* Processor
* Queue
* Extractor

### Processor

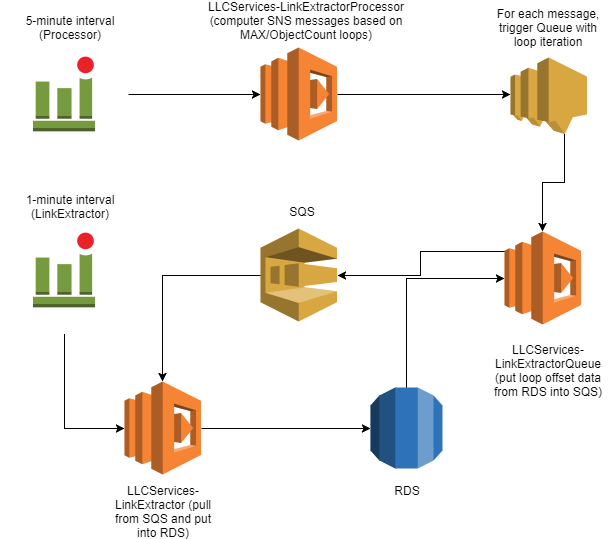
The processor runs on a 5-minute schedule (configured in CloudWatch). If the current SQS queue is empty, it will get all the HTML Objects that have a **ContentLastModified** date > than **DateLinksLastExtracted** date. If the Queue is not empty, it will silently exit. Since there could be a very large number of Objects, this function does not process all of the links directly. Instead, **MAX** loops are calculated from the total amount. For each loop, an **SNS** message is sent out received by the [Queue](#_Queue).

### Queue

The Queue is responsible for putting the **MAX** Objects into an SQS queue. Based on the loop value based through to the **SNS** message, the Queue will grab the Objects that match the offset from the Database and put them into the SQS queue. The **MAX** constant needs to be a reasonable value that can run within Lambda time constraints.

### Extractor

The Extractor will grab a message from the SQS queue. Links are extracted from the HTML response of the object and put into the Database if they do not already exist. After processing of the Object, it is removed from the SQS queue. If the queue is empty, it will silently exit until the [Processor](#_Processor) has a chance to run again. The diagram that shows how the Processor, Queue and Extractor work together is shown below.



## LinkChecker

The LinkChecker application will check a fraction of the links extracted from the various sources. Each check of a URL is stored as a separate entry in the **Stats** table. If the mean of the stats within the specified days (set in the admin settings) are outside of the maximum standard deviation allowed, they are added to the warning report/table. Any invalid links are added to the invalid report/table. This application is made up of three parts to work in AWS. The three parts are:

* Processor
* Queue
* Checker

### Processor

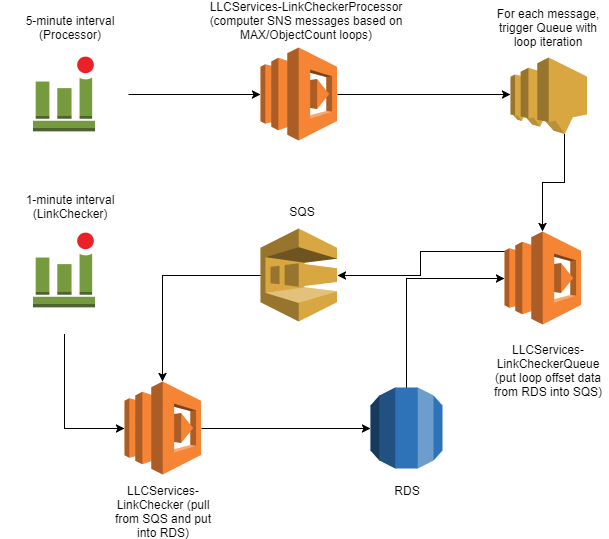
The processor runs on a 1-minute schedule (configured in CloudWatch). If the current SQS queue is empty, it will get all the HTML Links from the Database. If the Queue is not empty, it will silently exit. Since there could be a very large number of Links, this function does not process all of the links directly. Instead, **MAX** loops are calculated from the total amount. For each loop, an **SNS** message is sent out received by the [Queue](#_Queue_1).

### Queue

The Queue is responsible for putting the **MAX** Links into an SQS queue. Based on the loop value based through to the **SNS** message, the Queue will grab the Links that match the offset from the Database and put them into the SQS queue. The **MAX** constant needs to be a reasonable value that can run within Lambda time constraints.

### Checker

The Checker will grab a message from the SQS queue. The Link is downloaded, and a new Stat is created. If a screenshot does not exist, a new one will be created. If the Link is invalid, it will be added as an invalid report item. If it is determined to have a significant change, it will be added as a warning report. After processing of the Link, it is removed from the SQS queue. If the queue is empty, it will silently exit until the [Processor](#_Processor_1) has a chance to run again. The diagram that shows how the Processor, Queue and Extractor work together is shown below.



## Packages

The Packages application runs every time a package is uploaded to S3. Those packages must have a .zip or .imscc extension. The packages will be downloaded from S3, extracted to the file system, and then scanned for links. Once fully processed, the records will be added to the database and flagged as processed. This is accomplished by the following flow using an S3 trigger on the package bucket.



# Appendix A: Project Setup /w AWS

The following steps build the LOR Link Checker on AWS. These steps show how to build the project from scratch.

1. **Deploy the Screenshot service**
   1. You can use [npm](https://www.npmjs.com/get-npm) to install the serverless application.

*npm install -g serverless*

* 1. Navigate to the *serverless-screenshot* folder and execute:

*npm install*

*serverless deploy -s prod*

1. **Deploy the back-end from Visual Studio**
   1. Install Visual Studio 2017
   2. Install [AWS Toolkit for Visual Studio 2017](https://marketplace.visualstudio.com/items?itemName=AmazonWebServices.AWSToolkitforVisualStudio2017)
   3. Right-click on **SAM** project from **LLC-Serverless** solution and **Publish to AWS Lambda…** and use the following properties with <stack name>:
      * BucketName : <blank>
      * DbUsername : <llcadmin>
      * DbPassword : <omitted>
      * DbCidr : <0.0.0.0/0>
      * DbSnapshot : <arn for snapshot>
      * ObjectBucketName : slor
      * ScreenshotApiKey : <omitted, retrieve from API gateway>
      * ScreenshotServiceUrl : <from API gateway>
      * ShouldCreateBucket : true
      * ShouldCreateObjectBucket : false
2. Deploy the front-end
   1. Find the file under *react-llc\src\Auth\auth0-variables.js.prod* and update the following properties:
      1. AUTH\_CONFIG.domain (from auth0)
      2. AUTH\_CONFIG.clientId (from auth0)
      3. AwsConstants.InvokeUrl (from API gateway for back-end URL)
   2. Re-name *react-llc\src\Auth\auth0-variables.js to react-llc\src\Auth\auth0-variables.js.sand*
   3. Re-name *react-llc\src\Auth\auth0-variables.js.prod to react-llc\src\Auth\auth0-variables.js*
   4. Navigate to the *react-llc* folder and execute:

*npm install*

*npm run build*

* 1. Copy the build folder contents (not including the build directory) to the S3 bucket created (or referenced) by the **BucketName** property from the back-end deploy
  2. Optionally, attach a DNS hostname to the CloudFront distribution backed by the **BucketName** S3 bucket

1. **Attach Lambda trigger on slor bucket**

Add two S3 trigger events:

* + 1. Name: Delete (LLC)
    2. Delete Marker Created
    3. Send to: Lambda
    4. Lambda function like <stack name>-LLC-Serverless-ObjectExtractor
    5. Name: Create (LLC)
    6. Object Create (All)
    7. Send to: Lambda
    8. Lambda function like <stack name>-LLC-Serverless-ObjectExtractor

# Appendix B: Settings Available

|  |  |  |
| --- | --- | --- |
| Setting | Description | Update on New Site |
| Email.InvalidLinksLink | Link to the invalid reports. | Yes |
| Email.LogsLink | Link to the logs for the application. | No |
| Email.NotificationEmail | The email to send to with impacted links. | Yes |
| Email.Pass | Password for the Google SMTP | No |
| Email.User | Username for the Google SMTP | No |
| Email.WarningLinksLink | Link to the warning reports. | Yes |
| LinkCheckDays | The number of days to get links back to check to see if they have changed significantly. | Optional |
| LinkCheckStandardDeviations | Number of standard deviations that the checked links can be between without triggering a notification. | Optional |
| PackagesBucket | The URL of the S3 bucket that holds the packages. | Yes |

lorlinkcheckeridla@gmail.com

#Sc8B{Y8)&8FWQ8p