Jarvis

User Manual

Version 1.0.0

Table of Contents

App Overview

This application will help eliminate wasted time when checking for various server health related questions. With the help of Jarvis, a slack bot, developers will be allowed to simply ask Jarvis for a complete or individual health check and get a response from Jarvis. Prior to Jarvis, the developer would have to go check each individual data to find out if everything is okay. Some of the features that are being checked are down time in server, CPU usage, RAM usage, and many others.

Solving this problem allows for developers to save time by not having to repeatedly check these things individually. Developers could just ask Jarvis for a complete health check to get all the data without having to check for themselves. Jarvis also allows for the developers to work without much wasted time because it is integrated with slack, which developers often use to communicate with other developers within their group. Building the bot with slack allows the developers to maintain their normal routine while building in additional functions to make testing/checking easier. Another benefit is, if one developer checked for a complete check of the project it will be followed by a time stamp and other developers wouldn’t have to recheck because it may have been checked recently enough that the data would be accurate to use.

Jarvis is built as a slack bot with Amazon Web Services Lambda. Once a request is submitted it will then go to the server side get the data and return the data. This allows for the company to save costs because it will only be charged when used. Another benefit of Jarvis is that it could easily be built on top of, which allows for the company to add more features on to Jarvis with ease.

Install Instructions

Pre-requisites

- Node.js v4.3.2 (https://nodejs.org)
- npm (https://www.npmjs.com/)
- AWS account with access to IAM and Lambda

Project Setup

Claudia Bot Builder is an npm module that makes it easy to launch the bot as an Amazon Web Services (AWS) Lambda function. It takes care of creating the Lambda function, creating the proper Application Programming Interface (API) gateways, and getting basic permissions setup. This subsection includes step-by-step instructions on how to install the project for development and get the bot working as an AWS Lambda function. Adding the bot to Slack will be in the following section.

Information on Claudia is arduous to find. The site is unorganized and difficult to locate core documentation pages. From inspection, the following are the core documentation pages for Claudia:
Claudia: https://github.com/claudiajs/claudia/tree/master/docs
Claudia-api-builder: https://github.com/claudiajs/claudia-api-builder/tree/master/docs
Claudia-bot-builder: https://github.com/claudiajs/claudia-bot-builder/tree/master/docs

One must understand that, Claudia is a deployment utility and a development dependency that must be installed globally (npm install -g) to execute commands that will deploy a lambda function--it is not a module you use to create bot or API responses. Depending on your objective, you would use Claudia-api-builder and/or Claudia-bot-builder npm module by installing it as a dependency (npm install -S) to the project. Claudia-api-builder is used for creating APIs deployable to AWS. Claudia-bot-builder is used for creating conversational bots that can be deployed to AWS. Note, that Claudia-bot-builder uses Claudia-api-builder to create the API that will be used for setting up the environment for features like, slack landing page where oauth can be handled.

The following pages provide links to specific areas when using claudia (there may be something specific you would like to see):
https://claudiajs.com/tutorials/index.html
https://claudiajs.com/documentation.html
https://github.com/claudiajs/example-projects

For more information about development with claudia, please see:
Chatbot: https://claudiajs.com/tutorials/hello-world-chatbot.html via claudia-bot-builder.
API: https://claudiajs.com/tutorials/hello-world-api-gateway.html via claudia-api-builder.
Constructing easy deployment: https://claudiajs.com/tutorials/package-json-scripts.html.

For more information about installing and and setting up Claudia, please see:
https://claudiajs.com/tutorials/installing.html

1. Create a directory you would like to develop the bot from.

2. Extract and copy the project source files into the same directory, including the project.json file.

3. Navigate to this same directory in a terminal or command prompt.

4. In the terminal window, enter the command `npm install claudia -g` to install the claudia utility globally.

5. Once finished, enter the command `npm install` to install all project dependencies. Wait for the process to finish.

6. Now it is time to create the bot itself, to do this enter the following command:

   ```
   claudia create --region us-east-1 --api-module bot
   -configure-slack-slash-command
   ```

   Replace the region with the AWS region you would like to deploy the bot in. This command will create the needed AWS elements automatically. Answer the questions that follow in the terminal to give Claudia access to AWS.

7. Once finished, some general information is returned about the bot. You will want to save the slackSlashCommand URL for the next section.

8. That’s it! The bot is now deployed to AWS. Please proceed to the next section to connect the bot with your teams Slack. To push changes to code and update the bot, use the following command from the same directory: `claudia update`

Connecting Jarvis to Slack

Jarvis should now be uploaded to AWS as a Lambda function. The following steps will add Jarvis to your teams Slack. This involves creating your own Jarvis Slack app. The app is not intended to be published yet, it will be used locally by the Slack team you create the app with.

1. Visit https://api.slack.com/apps and sign in with your team information.

2. Click the Create New App button.

3. Enter Jarvis as the app name, or any other name you prefer. Select your team as the Development Slack Team and click Create App.

4. Under Features on the left pane, select Slack Commands. Click Create New Command. For the Command, enter */jarvis* and for the Request URL, enter the URL from step 7 in project setup. Fill in appropriate values for Short Description and Usage Hint before clicking Save.

5. Under Features on the left pane, click Bot Users. Then click Add Bot User. The bot name should be autofilled. Finally, click the green Add Bot User to add the bot.

6. On left side, click *OAuth & Permissions*. Click add new redirect url and enter your Slack redirect URL. This URL is the request URL, but change */slash-command* to */landing*. Select Permission scopes, pick commands and bot. Then save changes and click install app to team and authorize.

Giving Jarvis the proper AWS permissions

As of now, Jarvis commands will not entirely work in Slack. AWS permissions must be enabled for Jarvis. These permissions can be customized to disallow jarvis access to specific services if need be.

1. Open your AWS console.

2. Navigate to IAM → Roles.

3. You should see a role that matches the name you provided for the bot in Project Setup. The role name will end in *-executer*. Open this role by clicking on it.

4. Now you must attach the needed permissions to Jarvis so that the bot can access these services. Click the Attach Policy button and add the following policies:
   ```
   AWSHealthFullAccess
   AmazonRDSFullAccess
   AmazonEC2FullAccess
   AWSLambdaFullAccess
   IAMFullAccess
   CloudWatchFullAccess
   CloudWatchActionsEC2Access
   AWSLambdaRole
   ```

5. Jarvis should now have all the needed permissions to access the AWS information it needs for each command.

Commands

To send a query on AWS, enter a ‘/jarvis’ and the command you would like to execute. For instance, if you want to know the status of an EC2 server, you would enter:

‘/jarvis ec2status’

Another feature that comes with these commands are options/flags. These allow the user to be able to access the desired information in a more specific manner. So, going back to the EC2 server status, if you want to sort the information by tag name, the syntax for that would be:

‘/jarvis ec2status -t[or --tag] tag_name’

Here is the list of commands that are currently available for the Jarvis app:

Command	**Description**
help	Lists available commands.

EC2

Command	**Description**
ec2status	Instance health status.
ami	Amazon Machine Image (AMI) status information.
ec2cpu	Current server CPU usage.
ec2disk	Instance volume usage information.
ec2network	Instance network usage information.
ec2info	Generic EC2 instance information.

ec2net	Instance network information.
ec2ebs	Elastic Block Storage (EBS) volume information.
ec2bytag	Lists instances by specified tag data.

More Info:

ec2status

Description: Instance health status. Shows current status information for the EC2 instances. Instance State, System Status and Instance Status are provided.

Flags/Options:

'--help'	Displays help information for command.
-t|--tag [tag_name]	Filter by tag name.
-k|--key	Filter by tag key instead of value.

Syntax:
```
'/jarvis ec2status'
'/jarvis ec2status -t TagName'
'/jarvis ec2status -kt TagKey'
```

ami

Description: Amazon Machine Image (AMI) status information. Image information for images currently attached to an EC2 instance. Status is pending, available, invalid, deregistered, transient, failed or error.

Flags/Options:

'--help'	Displays help information for command.
-t|--tag [tag_name]	Filter by tag name.
-k|--key	Filter by tag key instead of value.

Syntax:

```
‘/jarvis ami’
‘/jarvis ami -t TagName’
‘/jarvis ami -kt TagKey’
```

ec2cpu

Description: Current server CPU usage. CPU usage is displayed as a percentage from 0 to 100%. The usage data is an average of the CPU usage over a period of time. The default time period is the last 5 minutes. All time periods are the difference between current time and the time provided.

Flags/Options:

‘--help’	Displays help information for command.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.
-m | --minutes	Sets data time period to specific minute value.
-h | --hours	Sets data time period to specific hour value.
-d | --days	Sets data time period to specific day value.

Syntax:

```
‘/jarvis ec2cpu’
‘/jarvis ec2cpu -t TagName’
‘/jarvis ec2cpu -kt TagKey’
‘/jarvis ec2cpu -minutes 30’
‘/jarvis ec2cpu -days 10’
‘/jarvis ec2cpu -kt TagKey --hours 5’
```

ec2disk

Description: Instance volume usage information. Retrieves read/write IOPS (Input/Output Operations per Second). The usage data is an average of the volume usage over a period of time. The default time period is the last 5 minutes. All time periods are the difference between the current time and the time provided.

Flags/Options:

'--help'	Displays help information for command.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.
-m | --minutes	Sets data time period to specific minute value.
-h | --hours	Sets data time period to specific hour value.
-d | --days	Sets data time period to specific day value.

Syntax:
```
'/jarvis ec2disk'
'/jarvis ec2disk -t TagName'
'/jarvis ec2disk -kt TagKey'
'/jarvis ec2disk -minutes 30'
'/jarvis ec2disk -days 10'
'/jarvis ec2disk -kt TagKey --hours 5'
```

ec2network

Description: Instance network usage information. Retrieves size of data transferred in and out of the instance network. The usage data is an average of the network usage over a period of time. The default time period is the last 5 minutes. All time periods are the difference between the current time and the time provided.

Flags/Options:

‘--help’	Displays help information for command.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.
-m | --minutes	Sets data time period to specific minute value.
-h | --hours	Sets data time period to specific hour value.
-d | --days	Sets data time period to specific day value.

Syntax:

```
‘/jarvis ec2network’
‘/jarvis ec2network -t TagName’
‘/jarvis ec2network -kt TagKey’
‘/jarvis ec2network -minutes 30’
‘/jarvis ec2network -days 10’
‘/jarvis ec2network -kt TagKey --hours 5’
```

ec2info

Description: Generic EC2 instance information. Retrieves “everyday” information about an EC2 instance. Instance ID, Attached AMI, Instance Type, Region and more are provided.

Flags/Options:

‘--help’	Displays help information for command.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.

Syntax:

```
‘/jarvis ec2info’
```

```
‘/jarvis ec2info -t TagName’
‘/jarvis ec2info -kt TagKey’
```

ec2net

Description: Instance network information. Retrieves information about the instance network configuration. Instance ID, Network Status, Public IP, Private IP and more are provided.

Flags/Options:

‘--help’	Displays help information for command.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.

Syntax:

```
‘/jarvis ec2net’
‘/jarvis ec2net -t TagName’
‘/jarvis ec2net -kt TagKey’
```

ec2ebs

Description: EBS (Elastic Block Storage) volume information. Retrieves information for volumes attached to an EC2 instance. Volume SIze, Region, Status, Encryption State and more are provided.

Flags/Options:

‘--help’	Displays help information for command.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.
-e | --encrypted	Filter volumes by encryption state (encrypted only).

-n | --not-encrypted	Filter volumes by encryption state (non-encrypted only).

Syntax:

```
‘/jarvis ec2info’
‘/jarvis ec2info -t TagName’
‘/jarvis ec2info -kt TagKey’
‘/jarvis ec2info --not-encrypted’
‘/jarvis ec2info -e’
‘/jarvis ec2info -nkt TagKey’
```

ec2bytag

Description: Lists instances by specified tag data. Instance Name/ID provided for all instances that match the user’s filter criteria. If no arguments are provided, then all instances will be listed.

Flags/Options:

‘--help’	Displays help information for command.
--notags	Filter results by instances that have no tags.
--notag [tag_name]	Filter results by instances that do NOT have a specified tag.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.

Syntax:

```
‘/jarvis ec2bytag’
‘/jarvis ec2bytag -t TagName’
‘/jarvis ec2bytag -kt TagKey’
‘/jarvis ec2bytag --notags’
‘/jarvis ec2bytag -notag TagName’
‘/jarvis ec2bytag -kt TagKey --notag TagName’
```

S3

Commands	**Description**
s3bytag	Lists buckets by specified tag data.
s3objects	Lists objects in S3 buckets(s).
s3acl	Retrieves the Lifecycle Configuration for an S3 bucket.
s3policy	Retrieves S3 bucket policy information.
s3info	Generic S3 bucket information.
s3logging	Bucket logging information.

<u>More Info:</u>

s3bytag

Description: Lists buckets by specified tag data. Bucket Name/ID provided for all buckets that match the user's filter criteria. If no arguments are provided, then all buckets will be listed.

Flags/Options:

'--help'	Displays help information for command.
--notags	Filter results by instances that have no tags.
--notag [tag_name]	Filter results by instances that do NOT have a specified tag.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.

Syntax:
```
'/jarvis s3bytag'
'/jarvis s3bytag -t TagName'
'/jarvis s3bytag -kt TagKey'
'/jarvis s3bytag --notags'
```

```
‘/jarvis s3bytag -notag TagName’
‘/jarvis s3bytag -kt TagKey --notag TagName’
```

s3objects

Description: Lists objects in S3 bucket(s). Provides a list of objects based on users criteria from S3 buckets. Object filename and size are provided.

(WARNING: Filtering by --objtag or --objkey will cause delays in output.)

Flags/Options:

‘--help’	Displays help information for command.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.
-n | --name [bucket_name]	Filter results by bucket name.
-a | --alpha	Sorting objects alphabetically A-Z.
-s | --size	Sort objects by file size.
-d | --date	Sort objects by date last modified.
--date-range	Get objects last modified in a date range.
--search [keyword]	Search for objects related to given string.
--objtag	Filter objects by object value.
--objkey	Filter object by object key.
-o | --owner [owner_name]	Filter objects by provided owner name. ***Not available in all regions***
-m | --max	Max objects to retrieve per bucket.

Syntax:
```
‘/jarvis s3objects’
‘/jarvis s3objects -t TagName’
‘/jarvis s3objects -kt TagKey’
```

```
'/jarvis s3objects --alpha'
'/jarvis s3objects --search Keyword -s'
'/jarvis s3objects -ao OwnerName'
'/jarvis s3objects --date-range 3/4/2016-2/18/2017'
```

s3acl

Description: Retrieves the Access Control List configuration for an S3 bucket. The Access Control List sets grants to manage access to the S3 bucket. Owner name, email, ID and more are retrieved for each grant.

Flags/Options:

'--help'	Displays help information for command.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.
-n | --name [bucket_name]	Filter results by bucket name.

Syntax:
```
'/jarvis s3acl'
'/jarvis s3acl -t TagName'
'/jarvis s3acl -kt TagKey'
'/jarvis s3acl -n BucketName'
```

s3policy

Description: Retrieves S3 bucket policy information. S3 bucket policy is the access policy for a bucket. Policy version, ID, SID, Principals and more are provided.

Flags/Options:

'--help'	Displays help information for command.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.

-n | --name [bucket_name]	Filter results by bucket name.
-r | --raw	Flag which will return the raw JSON policy of the bucket.

Syntax:
```
‘/jarvis s3policy’
‘/jarvis s3policy -t TagName’
‘/jarvis s3policy -kt TagKey’
‘/jarvis s3policy -n BucketName’
‘/jarvis s3policy --raw --name BucketName’
```

s3policy

Description: Generic bucket information. Retrieves “everyday” information about an S3 bucket. Bucket region, owner, size, and more are provided.

Flags/Options:

‘--help’	Displays help information for command.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.
-n | --name [bucket_name]	Filter results by bucket name.
-q | -quick	Flag to speed up this command by not providing bucket size.

Syntax:
```
‘/jarvis s3info’
‘/jarvis s3info -t TagName’
‘/jarvis s3info -kt TagKey’
‘/jarvis s3info -n BucketName’
‘/jarvis s3info -qn BucketName’
```

s3logging

Description: Bucket logging information. Logging provides records of access requests to the bucket. Logs are usually stored in a target bucket. Target Bucket and Target Prefix are provided.

Flags/Options:

‘--help’	Displays help information for command.
-t | --tag [tag_name]	Filter by tag name.
-k | --key	Filter by tag key instead of value.
-n | --name [bucket_name]	Filter results by bucket name.

Syntax:

```
‘/jarvis s3logging’
‘/jarvis s3logging -t TagName’
‘/jarvis s3logging -kt TagKey’
‘/jarvis s3logging -n BucketName’
```

Extending The Application

Overview

The application was developed using the WebStorm IDE by JetBrains, though any Javascript enabled IDE will work for development. Node.Js 4.3.2 was used for development and is required. See Project Setup under Install Instructions to setup the project before development.

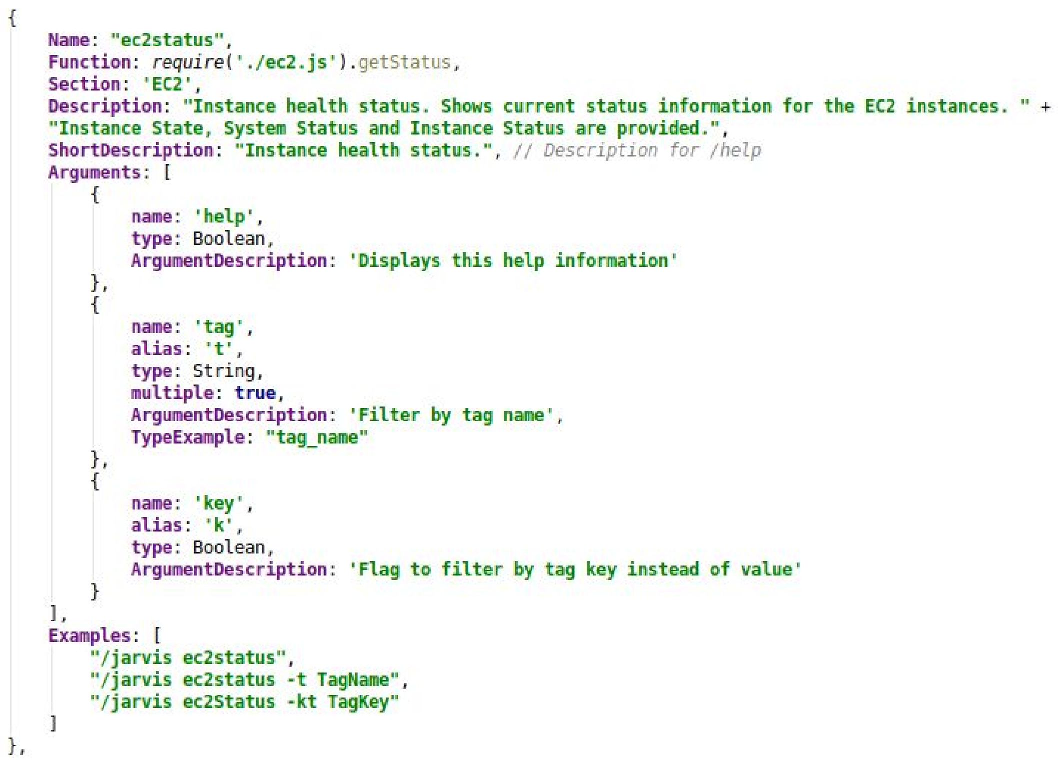
Extending the application is fairly straightforward. Once the project is set up, the code can be edited and pushed to overwrite the existing bot in AWS Lambda by invoking the following terminal command in the project directory:

```
claudia update
```

This will automatically update the new code, changing any gateways as needed. Once updated, jarvis can be tested in Slack with */jarvis [command] [arguments].*

Adding Commands

The process of adding commands was created to be as easy as possible. All commands are located in the *commands_list.js* file. They are stored as JSON objects. See the example command below.



```
{
    Name: "ec2status",
    Function: require('./ec2.js').getStatus,
    Section: 'EC2',
    Description: "Instance health status. Shows current status information for the EC2 instances. " +
    "Instance State, System Status and Instance Status are provided.",
    ShortDescription: "Instance health status.", // Description for /help
    Arguments: [
        {
            name: 'help',
            type: Boolean,
            ArgumentDescription: 'Displays this help information'
        },
        {
            name: 'tag',
            alias: 't',
            type: String,
            multiple: true,
            ArgumentDescription: 'Filter by tag name',
            TypeExample: "tag_name"
        },
        {
            name: 'key',
            alias: 'k',
            type: Boolean,
            ArgumentDescription: 'Flag to filter by tag key instead of value'
        }
    ],
    Examples: [
        "/jarvis ec2status",
        "/jarvis ec2status -t TagName",
        "/jarvis ec2Status -kt TagKey"
    ]
},
```

When a command block similar to the example is added to the file, the command is then automatically added to the help output and is automatically given --help argument functionality.

The *Name* field is the actual command name itself. This is what the user will input into Slack.

The *Function* field is used to specify which function is responsible for handling output for this command. Every function with this role MUST return a String or a Promise that resolves a SlackTemplate object. Also, parenthesis cannot be added to the end of this function assignment, such as *getStatus()*. Existing command functionality can be found by referencing this value for that command in *command_list.js.*

The *Arguments* field is used to add arguments. The format for this area follows the required format from the *command-line-args* module, with the exception of *ArgumentDescription* which is used by the help commands.

All other fields are used by the help commands to display information about the command and its arguments.

Known Issues

Issue #	**Issue Name**	**Description**
1	Slack Timestamp	The timestamp function to add a timestamp to message attachments in Slack using Claudia Bot Builder shows the incorrect time.
2	Jarvis Unresponsive	Rarely, the bot will not respond to the user when a request is made. The “Thinking...” message will not appear either. This may be due to backed up failed commands while developing.
3	Text as Emoji’s	Outside of an attachment, any text will be as if the user typed it in. Special values such as “:)” will be converted to an emoji.
4	EBS Cloudwatch	EBS Cloudwatch (AWS/EBS) API is not returning any data from AWS. This currently affects the disk command and any future EBS related commands.
5	AddAttachment	The AddAttachment function does not work outside of a callback. Reason unknown.
6	S3 Object Tags	Searching for an S3 Object by tag can be extremely slow. No way to help this, as a separate API call must be made for each object. Gets worse with more objects.
7	S3 Object Owners	Object owner names are not returned using the API. Searching by owners currently does not work.
8	Bucket Region	The region string that a bucket belongs in will not be returned while in specific regions. The API states the following regions will be recognized: "EU" "eu-west-1" "us-west-1" "us-west-2" "ap-south-1" "ap-southeast-1" "ap-southeast-2" "ap-northeast-1" "sa-east-1" "cn-north-1" "eu-central-1"
9	String Arguments	String arguments may not gracefully fail if the user does not provide a string.
10	Health API	The AWS Health API does not return any information beyond an error message.

Licensing Manifest

Outside Software Packages Used

- aws-sdk 2.7.10
    - Link: https://www.npmjs.com/package/aws-sdk
    - License: Apache 2.0

- claudia-bot-builder 2.2.0
    - Link: https://github.com/claudiajs/claudia-bot-builder
    - License: MIT

- columnify 1.5.4
    - Link: https://www.npmjs.com/package/columnify
    - License: MIT

- command-line-args 4.0.1
    - Link: https://www.npmjs.com/package/command-line-args
    - License: MIT

- promise-delay 2.1.0
    - Link: https://www.npmjs.com/package/promise-delay
    - License: ISC

- string-similarity 1.1.0
    - Link: https://www.npmjs.com/package/string-similarity
    - License: ISC

Internal components

arguments.js	
Helper file to handle functionality associated with user arguments.	
Package Used	**Scope**
None	

bot.js	
Entry point of the program. Used to receive/transmit messages to/from Slack.	
Package Used	**Scope**
aws-sdk	Used for invoking the lambda function that the bot exists in.
claudia-bot-builder	Used to create Slack message templates.
promise-delay	Used in sending messages to users Slack window.

cloudwatch.js	
Handles all commands using the Cloudwatch API.	
Package Used	**Scope**
aws-sdk	Used to access the AWS Cloudwatch API.
claudia-bot-builder	Used to create Slack message templates.

commands.js	
Parses user commands and routes to the appropriate function for output.	
Package Used	**Scope**
command-line-args	Used for parsing command arguments.
columnify	Improves output formatting for help commands/arguments.
string-similarity	Used to search for similar commands/arguments as part of error feedback.

commands_list.js	
Storage for all command information.	
Package Used	**Scope**
None	

ec2.js	
Handles all commands using the EC2 API.	
Package Used	**Scope**
aws-sdk	Used to access AWS EC2 API.
claudia-bot-builder	Used to create Slack message templates.

message.js	
Formatting for Slack messages and some message building helper functionality.	
Package Used	**Scope**
claudia-bot-builder	Used to create Slack message templates.

s3.js	
Handles all commands using the S3 API.	
Package Used	**Scope**
aws-sdk	Used to access AWS EC2 API.
claudia-bot-builder	Used to create Slack message templates.
string-similarity	Used to search for S3 Objects matching user input.

setEnviornments.js	
Helper to set environment variables in AWS lambda. Currently not utilized.	
Package Used	**Scope**
None	