AWS Network Helper

AWS Serverless ChatBot Documentation

# Goals & Features

The goal of this project is to provide an AWS network troubleshooting script that runs on a serverless architecture, and can be interacted with via Slack as a chat bot. In simple terms, the goal is to be able to respond to input like:

* Why can’t I connect to ec2-instance-A from ec2-instance-B?
* Troubleshoot connection between ec2-instance-B and rds-instance-C on port 5432
* I cannot connect to S3 from ec2-instance-A
* Help me connect to ec2-instance-B

In order to respond to inputs like these, the script must be able to analyze these network elements:

* Ingress and Egress Security Groups
* Ingress and Egress Network ACLs
* Route Tables
* NAT and Internet Gateways

Also, since information like instance type, port, and ephemeral ports may or may not be provided, the code must be able to look through metadata for these values, or at least make reasonable assumptions for what the user is most likely trying to accomplish.

The code is currently able to understand all of the statements above, as well as small variations in the wording. It can troubleshoot network settings for the following types of connections in both directions:

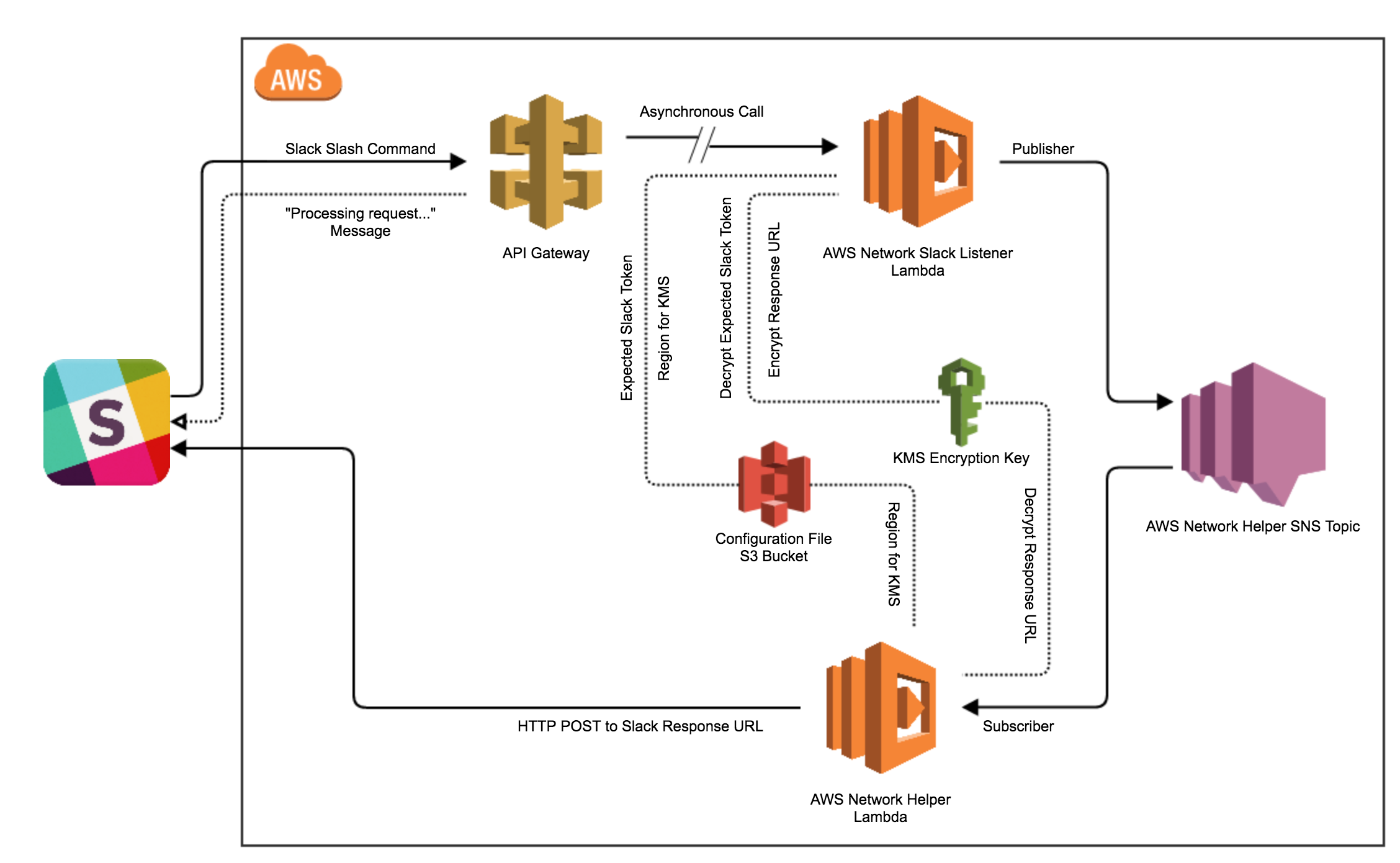
|  |  |  |
| --- | --- | --- |
| **Instance A** | **Instance B** | **Complexities** |
| EC2 | EC2 | Supports Windows and Linux |
| EC2 | RDS | All RDS engine types supported |
| EC2 | The Internet | Supports instances behind both Internet and NAT Gateways |
| RDS | The Internet | Supports instances behind both Internet and NAT Gateways |
| EC2 | AWS Services | S3, DynamoDB, KMS, SNS, SQS, etc. |

Wherever possible, the code should also not limit the scope of this project to only use Slack as the messaging interface.

# Architecture

AWS Services Used:

* API Gateway
* Lambda
* S3
* SNS
* IAM & KMS



An SNS topic is used between the Slack listener Lambda and the network helper Lambda so that in the future, different listeners could be deployed that use interfaces other than Slack. Other interfaces could include, but are not limited to:

* A web app
* A scheduled batch job for auditing network settings
* A command line tool
* A Python module

S3 is used to provide the user with an externalized configuration file for easier changes. The Slack token, SNS ARN, slash command, and other variables can be changed without re-compiling your Lambda

KMS is used to decrypt the Slack token stored in the configuration file upon use, and is also used to encrypt/decrypt the response URL as it gets passed through SNS. This is an added layer of security.

# Installation

1. Create the SNS Topic
   1. Topic Name: **aws-network-helper**
   2. Display Name: **aws-helper**
2. Create IAM Roles
   1. **aws-network-api-role**
      1. Role Type: AWS Service Roles - **Amazon API Gateway**
   2. **aws-network-helper-role**
      1. Role Type: AWS Service Roles – **AWS Lambda**
   3. **aws-network-listener-role**
      1. Role Type: AWS Service Roles – **AWS Lambda**
3. Create Inline Policy for aws-network-api-role
   1. Use the policy generator or create a manual policy to give the following permission to aws-network-api-role in an inline policy
      1. Effect: **Allow**
      2. Action: **lambda:InvokeFunction**
      3. Resource: **\***
   2. Use “**lambda-execution**” as the policy name
4. Create a KMS key in IAM with the following attributes:
   1. Alias: **aws-network-helper-key**
   2. Key Admins: User discretion
   3. Key Users:
      1. **aws-network-helper-role**
      2. **aws-network-listener-role**
5. Create S3 Bucket for Configuration Files
   1. Bucket Name: **aws-network-helper-<*account\_number*>**
      1. Where “<account\_number>” is replaced with the unique AWS account number for the account you are installing this in
   2. Region: User discretion
6. Create a Bucket Policy for your new S3 Bucket
   1. Use the policy generator or create a manual policy that grants the following privileges to the following roles:
      1. Privileges
         1. **s3:GetObject**
         2. **s3:GetObjectVersion**
      2. IAM Roles
         1. **aws-network-helper-role**
         2. **aws-network-listener-role**
7. Create new IAM policy with the following attributes:
   1. Template policy provided at:

***/aws-network-helper/aws\_services/iam\_policies/aws-network-helper-policy.json***

* 1. Policy Name: **aws-network-helper-policy**
  2. S3 Privileges
     1. AWS Service: **Amazon S3**
     2. Privileges
        1. **s3:GetObject**
        2. **s3:GetObjectVersion**
     3. Resource: **ARN for aws-network-helper-<*account\_number*>**
  3. SNS Privileges
     1. AWS Service: **Amazon SNS**
     2. Privileges
        1. **sns:ConfirmSubscription**
        2. **sns:GetEndpointAttributes**
        3. **sns:GetSubscriptionAttributes**
        4. **sns:GetTopicAttributes**
        5. **sns:Publish**
        6. **sns:Subscribe**
     3. Resource: **ARN for SNS topic - aws-network-helper**
  4. KMS Privileges
     1. AWS Service: **AWS Key Management Service**
     2. Privileges
        1. **kms:Decrypt**
        2. **kms:DescribeKey**
        3. **kms:Encrypt**
        4. **kms:ListAliases**
        5. **kms:ListKeys**
     3. Resource: **ARN for KMS Key – aws-network-helper-key**

1. Attach the following IAM policies to the following roles
   1. **aws-network-helper-role**
      1. **AWSLambdaBasicExecutionRole**
      2. **AmazonEC2ReadOnlyAccess**
      3. **AmazonRDSReadOnlyAccess**
      4. **aws-network-helper-policy**
   2. **aws-network-listener-role**
      1. **AWSLambdaBasicExecutionRole**
      2. **aws-network-helper-policy**
2. Create aws-network-slack-listener Lambda
   1. No template
   2. No triggers
   3. Name: **aws-network-slack-listener**
   4. Runtime: **Python 2.7**
   5. Copy and paste the code from the file below into the code box:

**/aws-network-helper/python/slack\_listener\_lambda/aws\_network\_slack\_listener.py**

* 1. Handler: **lambda\_function.lambda\_handler**
  2. Choose an existing role: **aws-network-listener-role**
  3. Memory: **128 MB**
  4. Timeout: **30 sec**
  5. VPC: **No VPC**

1. Use virtualenv and pip to package Python lambda code with dependencies
   1. From the location where you have checked out aws-network-helper:
      1. cd aws-network-helper/python/network\_helper\_lambda
      2. virtualenv env
      3. source env/bin/activate
      4. pip install -r requirements.txt
      5. cp -R env/lib/python2.7/site-packages/\* .
      6. rm -r env
      7. zip -r aws-network-helper-lambda.zip \*
2. Create aws-network-helper Lambda
   1. No template
   2. Trigger: **SNS Topic (aws-network-helper)**
      1. Enable trigger
   3. Name: **aws-network-helper**
   4. Runtime: **Python 2.7**
   5. Code Entry Type: **Upload a .ZIP file**
   6. Upload zipped lambda file from step 10
   7. Handler: **aws\_network\_helper.lambda\_handler**
   8. Choose an existing role: **aws-network-helper-role**
   9. Memory: **128 MB**
   10. Timeout: **2 min**
   11. VPC: **No VPC**
3. Create API Gateway API
   1. New API
   2. API Name: **aws-network-helper**
   3. Actions > Create Resource
      1. Resource Name: **slack**
   4. Actions > Create Method
      1. Method: **POST**
   5. POST Setup
      1. Integration Type: **AWS Service**
      2. AWS Region: User discretion
      3. AWS Service: **Lambda**
      4. AWS Subdomain: leave blank
      5. HTTP Method: **POST**
      6. Action Type: **Use path override**
      7. Path override: **/2015-03-31/functions/<*slack\_listener\_lambda\_arn*>/invocations**
         1. Where “<*slack\_listener\_lambda\_arn*>” is replaced with the ARN for the aws-network-slack-listener Lambda
      8. Execution Role: ***<aws-network-api-role-arn>***
         1. Where *“<aws-network-api-role-arn>”* is replaced with the ARN for the aws-network-api-role IAM role
   6. Integration Request > Body Mapping Templates
      1. Request Body Passthrough: **When there are no templates defined**
      2. Add mapping template
         1. Content-Type: **application/x-www-form-urlencoded**
         2. Template:

***/aws-network-helper/aws\_services/api\_body\_mapping\_templates/int\_request\_application\_x-www-form-urlencoded.json***

* 1. Integration Response > 200 response status > Body Mapping Templates
     1. **application/json**
        1. Template:

***/aws-network-helper/aws\_services/api\_body\_mapping\_templates/int\_response\_application\_json.json***

* 1. Deploy API to new stage
  2. Copy Invoke URL for step 13

1. Create New Slack Slash Command
   1. Use the Invoke URL from step 12 and these instructions to create your Slack slash command:

<https://api.slack.com/slash-commands>

* 1. Use **/aws-network** as the command
  2. Be sure to add “/slack” to the end of your API Gateway Invoke URL
  3. Make note of the token provided by Slack for step 14

1. Encrypt Slack token using “aws-network-helper-key” KMS key
2. Add these custom values to the file below:
   1. SNS ARN
   2. Encrypted Slack Token
   3. KMS Key Alias
   4. Slack Slash Command

***aws-network-helper/aws\_services/s3\_files/aws-network-helper-config.json***

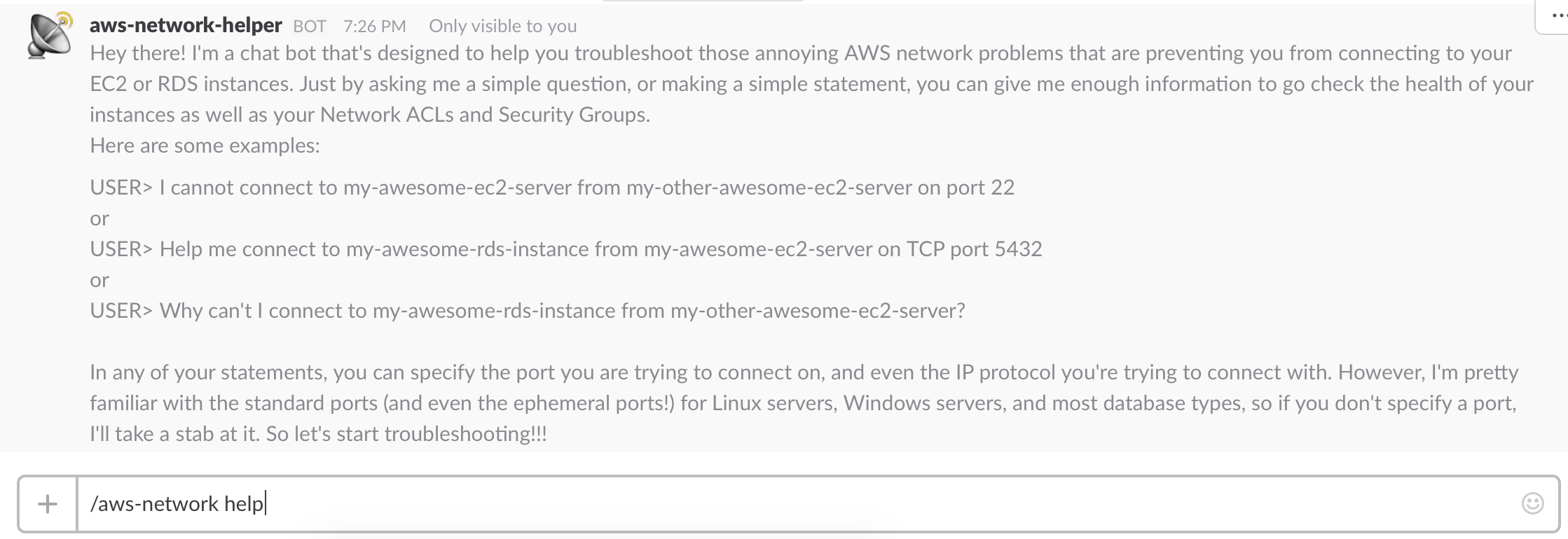
1. Upload the config file to:

**s3://aws-network-helper-<*account\_number*>/conf**

1. Test your slash command by using the following command:

/aws-network help

You should see something similar to this:



It is possible that the first call to API Gateway will timeout, so if you receive a timeout error from Slack, try again. If you get a successful response to the help request, you are ready to go! You can start asking questions about network connectivity in your AWS account. Happy troubleshooting!