Amazon Connect Lex4EC2 Patch Status Check – Solution Design and Setup Guide

March 8, 2021
Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

© 2021 Amazon Web Services, Inc. or its affiliates. All rights reserved.
About this guide

This guide contains an architecture and step-by-step implementation of a Minimal Viable Product buildout for an operations mechanism. This type of mechanism is a good fit for large organizations which consist of multiple teams with a central IT department. This MVP build enables IT staff members to either call in or chat via a chat client (such as Slack) to check on the status of maintenance scheduled patching jobs. While the MVP case remains simple, the buildout can be extended to other tasks, such as:

- Authorize and trigger an emergency patch job for Amazon EC2 instances with remotely exploitable vulnerabilities.
- Start on-demand network and/or vulnerability scan jobs and summarize the result.

This guide shows how Amazon Connect and Amazon Lex can be leveraged to integrate with the rest of Amazon Web Services (AWS) to simplify common but important IT tasks.
Before you begin

Prior to deploying this mechanism, consider the following:

- Who is responsible for applying the patches and how are they currently being notified of the patching result? For some, email is sufficient, while others might prefer lighter weight mechanism like SMS messages or chat clients. Appropriate changes must be made.

- How many instances are being patched at the same time? Remember that the current mechanism is meant to show the possible integrations, so it provides only the summary of patching status. This can be changed in the code.

- If more details of the patching status are required, Amazon Lex can be easily adjusted to provide additional details.

**Important**: A basic understanding of the Amazon Connect contact flows is required. Understanding of programming languages like Python and/or Nodejs might help.

Architecture and design

In this section, the known caller authentication flow and the unknown caller authentication flows are illustrated separately.
Known caller authentication flow

Key:

1. A user calls in, which initiates authentication by the contact flow.
   - The authentication **AWS Lambda** function is invoked, with the caller’s number as one of the parameters.

2. Using the number provided, the **Employee** table within the DynamoDB table is queried, and the match record is retrieved.

3. The entire employee record with the matching phone number is returned to the Lambda function.

**Note:** A "known" user means the caller’s phone number is in the system (for example, in the **Amazon DynamoDB** table).
4. The Lambda function asks for the permanent personal identification number (PIN) stored in the DynamoDB table.

5. The user enters the permanent PIN.

6. The authentication Lambda function signals the contact flow that the user has been authenticated.

**Unknown caller authentication flow**

**Note:** A "unknown" user means the caller’s phone number is **not** in the system.

**Key:**

1. A user calls in, which initiates the authentication by the contact flow.
The authentication Lambda function is invoked with the caller's number as one of the parameters.

2. Using the number provided, the **Employee** table within the DynamoDB table is queried and the match record retrieved.

3. The Lambda function notifies the contact flow that the **Employee** record lookup returned empty.

4. The Employee ID is requested.

5. The user enters the ID using the phone.
   - Behind the scenes, the Lambda function retrieves the employee record matching the ID.

6. The permanent PIN is requested.

7. The user enters the PIN.

8. The Lambda function signals the contact flow that the user has been authenticated, and this invokes the Lex bot.
Chat client authentication flow

Note: In this example, we are integrating with Slack chat client but Amazon Lex supports other Messaging Platforms such as Facebook messenger and Kik. For more information, see Deploying an Amazon Lex Bot on a Messaging Platform.

Key:

1. The user initiates the chat session with keywords such as “check patching”.
2. Amazon recognizes that the user session is from a chat client such as Slack, and it invokes the Lex bot with the session info.
3. The Lex bot asks for the Employee ID, and it queries the employee record from the DynamoDB table.
4. From the returned record, the bot retrieves the phone number and generates a randomized 6-digit One Time PIN (OTP). The PIN expires within 60 seconds.

5. Using Amazon Simple Notification Service (Amazon SNS), the Lex bot sends out the OTP.

6. The user types in the OTP in the same chat session, and authentication is complete.

Components and information architecture

Amazon Connect

Use Amazon Connect to build the virtual contact center. In this architecture, you need three contact flows:

1. **PatchStatus_MainFlow** — This contact flow is responsible for greeting the caller. It sets up logging, selects the voice, and redirects the call to the authentication flow.

2. **PatchStatus_Authentication** — The authentication flow is responsible for providing the basic authentication mechanism for Amazon Connect and Amazon Lex. This architecture assumes that employee records are kept in a DynamoDB table named “Employee”. (See the DynamoDB section of this document for the schema information.) Authentication uses the following workflow to authenticate or deny a user:

   a. If it’s a voice call, the authentication contact flow asks for an Employee ID. If it’s a chat session, the authentication is handled in the Lex bot, which is discussed later in this document.

   b. The contact flow looks up the table with the EmployeeID.

   c. If the record is not present, the contact flow notifies the user and disconnects.

   d. If the record is present, the calling number is checked against the PhoneNumber record in the Employee table.

      - If it’s a match, the contact flow asks for the permanent PIN stored in the same DynamoDB table.

         o If the permanent PIN is correctly provided, the contact flow hands the conversation off to the Lex bot.

         o If a wrong PIN is provided, the contact flow disconnects.
• If it’s not a match, the contact flow asks for an employee number.
  o The contact flow asks for the permanent PIN associated with the employee number. If the correct PIN is provided, the contact flow invokes the Lex bot. If not, the contact flow disconnects.

3. **PatchStatus_LexBot** — A flow is created to launch the Lex bot while setting the appropriate `sessionAttributes` to let the Lex bot know whether the user has connected via voice or chat. The two attributes that are set and used are as follows:

![The LexBot attributes](image)

When these attributes are set, the information is available in the `sessionAttributes` object. For example:

```json
"sessionAttributes": {
  "phoneNumber": "+14165555555",
  "callType": "VOICE"
}
```

The architecture uses information provided by the `sessionAttribute` object to determine whether a Lex driven one-time PIN authentication is required or not.

**Amazon Lex**

Next you will build the Amazon Lex bot that communicates with end users via voice or chat. The MVP build has a single intent (`patchStatusIntent`) and it must be maintained so that each intent has its own Lambda function, which, in turn, serves a single EC2 operations function such as “apply patching” or “check on the status of the patching”. This is so the Lambda functions can be smaller in size (which can help with the readability and understanding) and kept more modular.
In the patchStatusIntent intent, three slots are required:

- JobName
- EmployeeID
- OneTimePin

When the Lex bot is invoked by an end user who has phoned in, the EmployeeID and OneTimePin values are automatically propagated with dummy values, because the authentication has been completed by the main contact flow (PatchStatus_MainFlow). When the Lex bot is invoked by a chat client such as Slack, it does the following:

1. The Lex bot asks for the EmployeeID value.
2. It extracts the phone number from the Employee record in DynamoDB.
3. It sends a one-time PIN via AWS SMS.
4. It asks for the user to provide the OTP in the same chat session to establish the authenticity of the user.

For more information about Amazon Lex or help with how to get started, see Getting Started with Amazon Lex.

AWS Lambda

There are two AWS Lambda functions that must be created with proper permissions. (The required permissions are listed in the Setup Guide later in this document.) The Lambda functions are as follows:

1. Lex4EC2_PatchStatus_DBLookup — This is the Lambda function used by the authentication contact flow to authenticate the users who phoned in. The phone number is extracted from the DynamoDB table and used to determine whether the Employee ID input is needed or not. Then it asks for the permanent PIN, which is stored in the same DynamoDB table. (See the Amazon DynamoDB section of this document for the table schema.)
2. **Lex4EC2_PatchStatus_lambda** — This is the Lambda function that provides the intelligence to the Lex bot. As mentioned previously, the contact flow which invokes the Lex bot provides contact method information via the `sessionAttributes` object. This Lambda function works with that object to determine whether to deploy one-time authentication, and how to obtain the patching group name for which it is to retrieve the status. The users who use voice are authenticated by the contact flow and do not require additional authentication. The chat clients directly connect to the Lex bot contact flow, so the Lex bot does not need to perform authentication.

For an example of Lex bot information flow, see [Details of Information Flow](#). Other resources include:

- [API Reference](#)
- [Amazon Lex Developer Guide](#)

### Amazon DynamoDB

This architecture requires two Amazon DynamoDB tables:

1. **Employee**

   This is where employees’ personal information is kept. The columns are:

   - **Primary Key** — `EmployeeID`
   - `EmployeeID` — String (a three-digit number)
   - `EmployeeName` — String
   - `EmployeePIN` — String (a four-digit number kept as string)
   - `PhoneNumber` — String (E.164 format)

2. **Onetimepin**

   This is where one-time PINs are stored with a unique identifier (UUID) and an associated time stamp. The longevity of the one-time PIN is 60 seconds.

   - **Primary Key** — `uuid`
   - `Uuid` — String (32-character UUID)
   - `pin` — String (6-digit random number generated by the Lambda function)
   - `timestamp` — Number (epoch timestamp of the OTP – expires after 60 seconds)
Amazon SNS

In the architecture, an Amazon SNS API is used in the Lex4EC2_PatchStatus_lambda Lambda function to send out the OTP to the phone number associated with the provided Employee ID.

AWS Systems Manager

For the Lex bot to properly report on the patching status, two things in AWS Systems Manager must be configured:

1. Maintenance Windows and associated names — By design, the Lex bot reports on the patching tasks associated with a maintenance window. The names of the maintenance windows created for the patching operation must start with patch-job-reportedbyconnect-, and unique names should follow. This design gives the operations a choice to pick which jobs can or should be reported by the Lex bot.

2. The target groups for the patch job maintenance windows must be a “patch group” available in the Systems Manager.

For details about creating a patch group in the Systems Manager, see the Setup Guide in this document.

Instead of any other tagging or manually selecting EC2 instances, select targets based on patch groups, because patch groups are tag based. To read more about the Systems Manager patch group, see About patch groups.

Setup guide

The purpose of this guide is to provide you with step-by-step instructions for setting up the infrastructure required to check the EC2 Patching Status by dialing into a contact center and having an automated Lex bot check its status. This guide will touch upon:

1. Basic Amazon Connect contact center information
2. The Lex bot and the associated Lambda function
3. The basic contact flow
4. Systems Manager – Patch Manager
5. Systems Manager – Maintenance Window
1. Amazon Connect instance creation

See the following article to learn how to create the initial contact center in your account: [Create an Amazon Connect instance](https://aws.amazon.com/solutions/solutions-highlight-amazon-connect-instance/).

After you have created your instance and you have at least one administrative user that you can use to log in to the Connect instance, you are ready to proceed to the next step.

2. Amazon connect instance configuration

See [Set up a phone number for your call center](https://aws.amazon.com/solutions/solutions-highlight-set-up-phone-number/) to get started with a typical call center instance configuration. Continue with the documentation in this section to learn how to include defaults, additional queues, and routing profiles. For this architecture, you only need to claim a number because there is no human agent interaction.

3. Lex configuration

Lex bots leverage Lambda functions to interact with AWS resources outside the Lex bot. In this use case, you need a Lex bot to interact with the systems manager, and retrieve the status of the patching jobs in maintenance windows.

The first step to create and configure the Lex bot is to create a proper [AWS Identity and Access Management](https://aws.amazon.com/iam/) (AWS IAM) role, followed by the Lambda function. After that, you can configure Lex.

3.1 Execute the AWS CloudFormation template

The [AWS CloudFormation](https://aws.amazon.com/cloudformation/) template deploys and configures the following:

- The Lex bot-backing Lambda function
- The [AWS IAM](https://aws.amazon.com/iam/) role and resource-based policy assigned to the Lex bot Lambda function
- A Lambda function that looks up the DynamoDB table (used by the Contact Flow to authenticate users)
• An IAM role and resource-based policy assigned to the Lex bot Lambda function
• The creation of DynamoDB tables

To create the AWS CloudFormation template:

1. Click the Launch Stack button (signing required) to open the AWS CloudFormation console.

2. Alternatively, on the Specify template window, choose Amazon S3 URL and use the following URL to download a custom template:

   https://s3.amazonaws.com/blogpost-staging/lex4EC2_PatchStatus_solution-lambda.template

   Specify template
   A template is a JSON or YAML file that describes your stack’s resources and properties.

   Template source
   Selecting a template generates an Amazon S3 URL where it will be stored.
   ○ Amazon S3 URL  ○ Upload a template file

   Amazon S3 URL
   https://s3.amazonaws.com/blogpost-staging/lex4EC2_PatchStatus_solution-lambda.template

   S3 URL:  https://s3.amazonaws.com/blogpost-staging/lex4EC2_PatchStatus_solution-lambda.template

   Specify the template in CloudFormation

3. Click Next. The stack name should be automatically populated, but if not, use lex4EC2-PatchStatus-base-infrastructure.

4. Click Next.

5. On the Configure stack options page, click Next.

6. On the Review page under Capabilities, select I acknowledge that AWS CloudFormation might create IAM resources with custom names, then choose Create stack.

7. Wait until the status is CREATE_COMPLETE.

3.2 Create a sample entry in the DynamoDB table “Employee”

The CloudFormation template has created a table with the configuration that you specified, but you need one full entry to be present to complete the Lex4EC2 Patch Status configuration.
1. Go to the DynamoDB console and choose the **Employee** table.

2. Choose the **Items** tab. You will see one entry with two keys: EmployeeID and PhoneNumber.

![DynamoDB console Items tab](image)

3. Choose **Create Item**.

4. Add append records and add values for:
   - **PhoneNumber**: String (E.164 format). For example, for US numbers, this would be “+1” followed by the area code and phone number.
     - **EmployeeName**: String
     - **EmployeePIN**: String (a four-digit number kept as string)

   See the following figure for the result:

![Create item screen](image)

5. Choose **Save**.

   You can now move on to the Lex bot configuration.

### 3.3 Configure the Lex bot

1. Download the Lex bot export zip file:
2. Unzip and edit the resulting `patchStatusBot_Export.json` file:
   a) Look for `dialogCodeHook` and modify the Uniform Resource Identifier (URI).
      ```json
      "dialogCodeHook": {
          "uri": "arn:aws:lambda:us-east-1:123456789012:function:Lex4EC2_PatchStatus_lambda",
          "messageVersion": "1.0"
      }
      
      *dialogCodeHook line*
   
   b) Currently, the URI points at an account with the ID 123456789012. Change this to your account number. For example, if your account number is 098765432109, then the URI should be `arn:aws:lambda:us-east-1:098765432109:function:Lex4EC2_PatchStatus_lambda`.

3. Click Save.

4. Zip the `patchStatusBot_Export.json` file as `patchStatusBot_Export.json.zip`.

5. From your browser, open the Amazon Lex console.

6. Choose Actions > Import.

7. Choose Browse and select the zip file you just created.

8. Choose Import.

9. After successful import, open `patchStatusBot` and choose Build. You will see a “successful build” message.

10. Choose Publish.

11. Set the alias as `prod`, and choose Publish again.

4. Create and configure contact flows

In this step, you will add two additional contact flows and modify the main flow to include the new ones. First, ensure that the Lex bot you built is available for use in the contact flows:

1. Open the Amazon Connect console and choose the instance alias.

2. From the left menu, choose Contact flows.

3. Under **Amazon Lex**, select the bot you previously built.
Amazon Web Services

Amazon Lex

Integrate Amazon Lex bots into your contact flows to take advantage of the same speech recognition and natural language understanding technology that powers Alexa.

Note: By adding Lex bots, you are granting Amazon Connect permission to interact with them Create a new Lex bot

Select the bot you previously built


5. Under AWS Lambda, select the Lex4EC2_PatchStatus_DBLookup_Lambda authentication Lambda function that you previously built.

AWS Lambda

Amazon Connect can interact with your own systems and take different paths in IVR dynamically. To achieve this, invoke AWS Lambda functions in contact flows to interact with your own systems or other services, then build personalized and dynamic experiences based on data returned.

Note: By adding Lambda functions, you are granting Amazon Connect permission to invoke them Create a new Lambda function

Select the authentication Lambda function


4.1 Create PatchStatus LexBot contact flow

1. Download the Contact Flow template from: https://blogpost-staging.s3.amazonaws.com/PatchStatus_LexBot

2. Navigate to your Amazon Connect Instance.


4. Choose Create Contact Flow.

5. From the drop-down list by the blue Save button, choose Import Flow.

6. Choose the downloaded Contact Flow Template (PatchStatus_LexBot).

7. Choose Import.
8. On successful import, choose **Get customer input** on the left side.

---

**PatchStatus_LexBot contact flow**

9. Choose your LexBot.

<table>
<thead>
<tr>
<th>DTMF</th>
<th>Amazon Lex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Plays an audio prompt and branches based on DTMF or Amazon Lex intents. The audio prompt is interruptible when using DTMF.</td>
</tr>
</tbody>
</table>

- Lex bot: patchStatusBot (US East: N. Virginia)
- Name: patchStatusBot (US East: N. Virginia)
- Alias: prod

**Choose your LexBot**

10. Verify the **Contact Flow Name** and **Description**.

11. Click **Save**.

12. Click **Publish**

---

4.2 Create an employee authentication contact flow

1. Download the Contact Flow template from: [https://blogpost-staging.s3.amazonaws.com/PatchStatus_Authentication](https://blogpost-staging.s3.amazonaws.com/PatchStatus_Authentication)

2. Navigate to your Amazon Connect instance.
3. Choose **Routing > Contact Flows**.

4. Choose **Create Contact Flow**.

5. From the drop-down list by the blue **Save** button, choose **Import Flow**.

6. Choose the downloaded Contact Flow template (PatchStatus_Authentication).

7. Choose **Import**.

8. There are three instances of Lambda function invocation. All three Lambda functions **must** point to the **Lex4EC2_PatchStatus_DBLookup** Lambda function you previously created.

9. Choose **Transfer to flow**. See the following figure.
10. Ensure that the selected flow is the *PatchStatus_LexBot* flow you previously imported.

11. Check the Contact Flow *Name* and *Description*, then choose *Save*.

12. Choose *Publish*.

### 4.3 Create PatchStatus main contact flow

1. Download the Contact Flow template from:  
   https://blogpost-staging.s3.amazonaws.com/PatchStatus_MainFlow

2. Navigate to your Amazon Connect instance.

3. Choose *Routing > Contact Flows*.

4. Choose *Create Contact Flow*.

5. From the drop-down list by the blue *Save* button, choose *Import Flow*.

6. Choose the downloaded Contact Flow template (*PatchStatus_MainFlow*).

7. Choose *Import*.
8. Choose **Transfer to flow**.

```
PatchStatus_MainFlow flow
```

9. Ensure that the chosen flow is the **PatchStatus_Authentication** flow you previously created.

10. Check **Contact Flow Name** and **Description**, then click **Save**.

11. Click **Publish**.

### 4.4 Claim a number and associate it to the main flow

1. Ensure that you are signed into the Amazon Connect Instance, then navigate to **Routing > Phone Numbers**.

2. On the **Manage Phone numbers** screen, choose **Claim Phone number**.

3. Choose “DID” and the relevant country.

4. Choose **Save**.

5. Navigate to **Routing > Phone Numbers** to show the **Manage Phone numbers** dialog.

6. Choose the phone number to open a new window.

7. Under **Contact flow / IVR**, from the drop-down menu, select **PatchStatus_MainFlow**.

8. Choose **Save**.
5. Test case setup

Before you can proceed with testing, you need to set up an environment where the Lex Bot has something to report on. In this use case, set up the following:

- Number of EC2 instances – Amazon Linux 2
- Systems Manager Patch Group
- Systems Manager Maintenance Window
- Systems Manager Maintenance Window Task (patching)
- Systems Manager Maintenance Window Target

**Prerequisite**

Create an IAM role to be assigned to the EC2 instance. The policy that must be attached is:

```
arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

The trust policy (or the use case) must be set to EC2. For more information on the instance profile creation, see [Create an IAM instance profile for your Amazon EC2 instances](#).

**5.1 Launch EC2 instances**

Using Management Console (or another method of your choice), launch one or more Amazon Linux 2 EC2 instances whose Operating System state will be patched. During the launch, specify the following tag for each instance:

<table>
<thead>
<tr>
<th>Tag Key</th>
<th>Tag Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patch Group</td>
<td>AmazonLinux2</td>
</tr>
</tbody>
</table>

These will be used later by the Systems Manager to determine which Instances to patch. Ensure that the IAM role you created in the [Prerequisite](#) section is assigned to the EC2 instances.
5.2 Create a Systems Manager patch group

1. Open AWS Console and navigate to the Systems Manager console.
2. On the left side menu, choose **Patch Manager**.
3. If this is your first time in Patch Manager, your screen will look similar to the following:

![AWS Systems Manager first login screen](image)

4. Choose **View Predefined Patch Baselines** on the upper-right of the screen.
5. In the **Patch Manager** menu, choose the radio button beside **AWS-AmazonLinux2DefaultPatchBaseline**.
6. Choose **Actions > Modify patch groups**.

7. In the **Patch Groups** text field, enter AmazonLinux2, then choose **Add**.
8. Choose Close.

5.3 Create a Systems Manager maintenance window

**Note:** The **Name** of a maintenance window is important. It must start with the string `patch-job-reportedbyconnect-` and then be followed by other unique names such as `patch-job-reportedbyconnect-amazon-dev`. This is because the Lex bot Lambda function specifically searches for a maintenance window that starts with that string.

1. From the Systems Manager Console, choose **Maintenance Windows**.
2. Choose **Create maintenance window**.
3. Name the maintenance widow `patch-job-reportedbyconnect-amazonlinux`.

**Schedule, Duration,** and **Stop initiating tasks** can be of any values according to the organization’s internal standard (an example of a development environment’s patch cycle might be once a week, over the weekend, with a maintenance window of four
hours.) For the purpose of testing, build up some history of patching so that Lex4EC2 can report on it.

5.4 Create a Systems Manager maintenance window target

**Note:** The target name is very important. When a user calls in, this is the name for which the patching status is checked. A Target name maps to a Patch Group, so keep the target name simple and easy to pronounce; for example, AmazonLinuxDev or AmazonLinuxTest.

1. From the Systems Manager Console, choose Maintenance Windows.
2. Choose the desired maintenance window.
3. Choose the Targets tab.
4. Choose Register target.
5. Set the Target name to something of your choice that is distinctive and easy to pronounce.
7. Under Instance tags, specify the following:
   - **Tag key:** Patch Group
   - **Tag value:** AmazonLinux2
8. Choose Register target.

5.5 Create a Systems Manager maintenance window task

1. From the Systems Manager Console, choose Maintenance Windows.
2. Choose the maintenance window you created earlier.
3. Choose the Tasks tab.
4. Choose Register tasks > Register Run command task.
5. In the Command document section, locate AWS-RunPatchBaseline and choose the radio button next to the document.
6. Scroll down to the Targets section and choose the target you previously created.
7. Under Rate control, choose the percentages and specify 100%.
Rate control

Concurreny
Specify the number or percentage of targets on which to execute the task at the same time
- [ ] targets
- [ ] 100 percentage

Error threshold
Stop the task after the task fails on the specified number or percentage of targets
- [ ] errors
- [ ] 100 percentage

**Note:** For **Concurrency** and **Error threshold**, 100% is set for the purpose of this demo. This is not recommended for production use cases, because system administrators tend to favor failing early and fixing any issues before re-attempting to patch. For production use cases, we recommend that you follow the Systems Manager’s default threshold:

- Fleets smaller than 25 will abort patching after one instance fails.
- Fleets between 25 and 100 will abort patching after five instances fail.
- Fleets larger than 100 will abort patching after ten instances fail.

8. Under **Parameters**, ensure that the **Operations** is **Install**.
9. Choose **Register Run command task**.

5.6 Run voice test

The environment is ready for testing. Call the number obtained from the Connect Instance, authenticate, then say “check patching”.

Table 1 shows the possible output from the Lex bot:

*Table 1 — Possible Lex bot output*
### Situation | Lex Bot output
--- | ---
- The caller asked for the “latest” patching job(s) status  
- There were two or less patching jobs within the last 24 hours | A patching job associated with a maintenance window named ${maintenance window run name} started at ${start time} has the status of ${stop time}

- The caller asked for the “latest” patching job status  
- There were three or more patching jobs within the last 24 hours | [Message Header]  
You have a total of ${total number of patching jobs} patch jobs in the last 24 hours and  
[Either one of]  
- they all succeeded  
- they all failed  
- they are all in progress  
- ${number of successful jobs} of which succeeded, ${number of failed jobs} failed  
- ${number of successful jobs} of which succeeded, ${number of failed jobs} failed and ${number of in-progress jobs} in progress

- The caller asked for a particular maintenance window patching target group | The latest run status for the maintenance window target group ${maintenance window target group name} is ${status of the patching job}. It started at ${start time} and ended at ${stop time}

### 5.7 Run chat test

Follow the [Integrating an Amazon Lex Bot with Slack](guide) guide to integrate a Slack app with Amazon Lex. The important steps in the guide are:

- **Step 3**: Create a Slack application  
- **Step 4**: Integrate the Slack application with the Amazon Lex bot  
- **Step 5**: Complete Slack integration

In summary, a Slack app is created with authentication tokens which Amazon Lex records and authorizes. After the Slack app is integrated, you can add the app in the **Apps** section of the Slack window.
1. The end user initiates the session by entering “check patching”.
2. The Lex bot recognizes that the request comes from a chat session and invokes the OTP authentication.
   - The Lex bot asks for an Employee ID: 101.
   - The Lex bot sends OTP to the phone number associated with the Employee ID 101: 430437 (randomly generated).
   - The user sends the OTP within 60 seconds.
   - The Lex bot authenticates the user.
3. The Lex bot offers to check on a patching job.
4. The user checks on the `afdafa` job. The Lex bot searches but finds no matching Patching Group. It sends the message back.
5. The user checks on the “latest” job. The Lex bot retrieves the status of all patching jobs that executed within the last 24 hours and returns the message.
6. The user ends the session with the word “goodbye”.
7. The session ends and the authentication token becomes invalidated.

**Conclusion**

This guide has shown how Amazon Connect and Amazon Lex can be used to build a simple chat operation (chatops), and the initial buildout of a patch status report that can be obtained via phone call or chat.

Amazon Lex easily integrates with other AWS Services. With Amazon Lex, you can build bots to increase contact center productivity, automate simple tasks, drive operational efficiencies across the enterprise, and more.

**Contributors**

Contributors to this document include:
• Andrew Park: Solutions Architect, AWS WWCS Geo
• Parind Poi: Practice Manager, AWS WWCS PS-GSP
• Peter Villiers: Connect Consultant, AWS WWCS PS-GSP
• Erik Weber: Sr WW Management Tools Specialist SA, AWS WWCS Special Service Solution Architect

Document revisions

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 8, 2021</td>
<td>First publication</td>
</tr>
</tbody>
</table>