

Streamline Operations with Salesforce Email Services:

AUTOMATE, OPTIMIZE, AND EXCEL



Robert Davis
Solution Architect
Hike 2



SOUTHEAST
DREAMIN'

airSlate



CITRIN COOPERMAN®



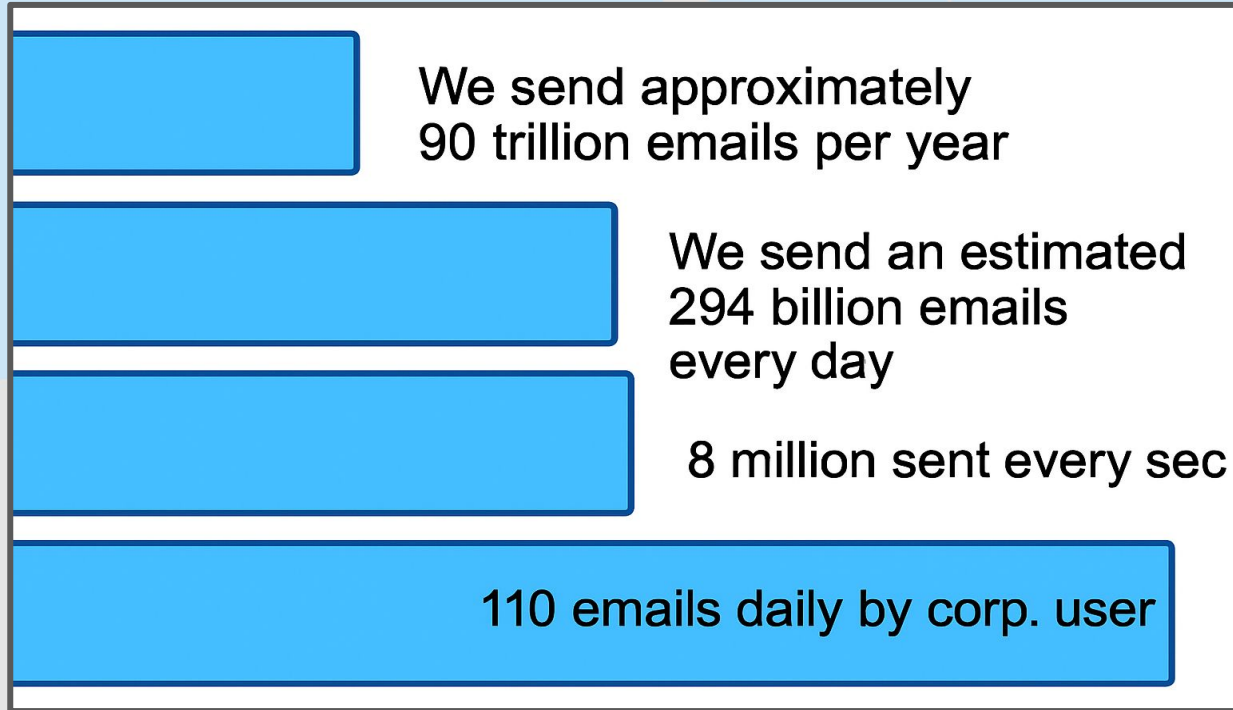
Score
Keeper

for Salesforce



DataGroomr
Data Quality Perfected by AI

Why Email Services?



Source:

Vingapp.com

Inbound Email Services - Out of the Box

Pros

- A native Salesforce feature that converts incoming emails into **Cases**.
- No Apex code needed.
- Great for support scenarios.
- Automatically creates Cases from email.
- Maintains email threads via Ref ID.
- Can attach files.
- Can assign Cases using rules.
- Fast setup



Limitations

- Only creates Cases, not Leads, Contacts, etc.
- Parsing email content into structured fields is limited without extra automation (e.g., Flow).
- Not great for non-service teams (like Sales or Custom Object needs).
- Limits on Case Settings coming in to new Case.

Real-World Story: The Arbitrator's Inbox

 **Real-World Example: One person, one inbox, holding up the whole process**

A legal firm receives critical email threads between lawyers and arbitrators...

Scenario: Legal firm receives arbitration-related emails

Problem: Sole person manually forwards messages to the legal team

Limitations of Email-to-Case: No support for custom workflows, other objects

Solution (Apex Email Service):

-  Automatically attaches emails to the correct Case
-  Creates follow-up Tasks
-  Adds metadata for accountability and audit trail

Email Services - On Demand / Apex

Pros

- Full control over how emails are processed.
- Great for custom logic (e.g., parsing CSV attachments, conditional routing).
- You can implement retry logic, error handling, custom logging.
- Can run external calls to other systems.

Limitations

- Requires Apex and tests.
- Must manage governor limits and potential errors carefully.
- More complex deployment and testing process.
- CI/CD Friendly
- Scalability
- Complex Logic



Real-World Story: Flow Error Logging: Catching What the User Never Reports

 **Real-World Example: Sometimes the most important errors are the ones nobody sees**

Scenario: Emails sent when automations fail

Problem: Visibility may be lost in email. How to prioritize on issues.

Limitations of Email-to-Case: No support for custom workflows, other objects

Solution (Apex Email Service):

-  Created a custom **Email Service** to receive Flow error messages
-  Parsed body, subject line, and headers
-  Logged errors to **Flow_Error__c** object for triage and reporting
-  Now searchable, reportable, and trackable over time

Key Components of an Apex Email Service

```
public with sharing class AutomationEmailService implements Messaging.InboundEmailHandler {  
    public Messaging.InboundEmailResult handleInboundEmail(Messaging.InboundEmail email, Messaging.InboundEnvelope envelope) {  
  
        Messaging.InboundEmailResult result = new Messaging.InboundEmailResult();  
  
        try {  
            // Log the incoming email details using Nebula Logger  
            Logger.error('Inbound Email Received');  
            Logger.setField(LogEntryEvent__e.Email_Subject__c, email.subject);  
            Logger.setField(LogEntryEvent__e.Email_From__c, email.fromAddress);  
            Logger.setField(LogEntryEvent__e.Email_Body__c, email.plainTextBody);  
            Logger.setField(LogEntryEvent__e.Email_To__c, String.join(email.toAddresses, ','));  
            Logger.saveLog();  
  
            result.success = true;  
        } catch (Exception e) {  
            Logger.error('Error handling inbound email: ' + e.getMessage() + ' stack trace: ' + e.getStackTraceString());  
            Logger.saveLog();  
  
            result.success = false;  
            result.message = 'Error: ' + e.getMessage();  
        }  
  
        return result;  
    }  
}
```

- **handleInboundEmail** — the **required method name**.
- **Messaging.InboundEmail** — contains the actual email content (subject, body, attachments, etc.).
- **Messaging.InboundEnvelope** — contains metadata like **fromAddress**, **toAddress**, etc.
- **Messaging.InboundEmailResult** — must be returned, and typically you just set **result.success = true**.

Messaging.InboundEmail Fields Available

- **binaryAttachments**: An array of binary attachments included in the email. Each attachment is represented by a `Messaging.InboundEmail.BinaryAttachment` object, containing details like `body`, `fileName`, and `mimeType`.
- **ccAddresses**: An array of strings representing the CC (carbon copy) email addresses specified in the incoming email.
- **fromAddress**: A string containing the email address of the sender.
- **fromName**: A string containing the display name of the sender, if available.
- **headers**: A string capturing the raw email headers of the incoming email.
- **htmlBody**: A string containing the HTML version of the email body, if the email was sent in HTML format.
- **inReplyTo**: A string representing the 'In-Reply-To' header of the email, which can be used to identify the message to which this email is replying.
- **messageId**: A string containing the unique identifier of the email message.
- **plainTextBody**: A string containing the plain text version of the email body.
- **subject**: A string representing the subject line of the incoming email.
- **textAttachments**: An array of text attachments included in the email. Each attachment is represented by a `Messaging.InboundEmail.TextAttachment` object, containing details like `body` and `fileName`.
- **toAddresses**: An array of strings representing the primary recipient email addresses specified in the incoming email.

Messaging.InboundEnvelope Fields Available

- **fromAddress**: A string containing the email address of the sender, as specified in the envelope.
- **toAddress**: A string containing the email address of the recipient, as specified in the envelope.



Email Service Setup

The screenshot shows the Salesforce Setup interface for Email Services. The left sidebar contains navigation links for 'Email Service', 'Email' (with sub-link 'Send through External Email Services'), and 'Custom Code' (with sub-link 'Email Services'). The main content area is titled 'Email Services' and includes a description: 'Email services are automated processes that use Apex classes to process the contents, headers, and attachments of inbound email. For example, you can create an email service that automatically creates contact records based on contact information in messages. Each email service has one or more email service addresses that can receive messages for processing.' Below this is a code block with Apex code for implementing the `Messaging.InboundEmailHandler` interface. At the bottom, there is a table listing existing email services.

Setup Home Object Manager

Search Setup

Email Service

▼ Email

- Send through External Email Services

▼ Custom Code

- Email Services

Didn't find what you're looking for?
Try using Global Search.

Email Services

Email services are automated processes that use Apex classes to process the contents, headers, and attachments of inbound email. For example, you can create an email service that automatically creates contact records based on contact information in messages. Each email service has one or more email service addresses that can receive messages for processing.

Before creating email services, create Apex classes that implement the `Messaging.InboundEmailHandler` interface.

```
public with sharing class myHandler implements Messaging.InboundEmailHandler {  
    public Messaging.InboundEmailResult handleInboundEmail(Messaging.InboundEmail email, Messaging.InboundEnvelope envelope) {  
        Messaging.InboundEmailResult result = new Messaging.InboundEmailresult();  
        return result;  
    }  
}
```

View:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z Other **All**

Action	Active	Email Service Name ↑	Apex Class	Last Modified By	Last Modified Date
Edit	✓	AIEmailHandler	EmailHandlerAI	ema	3/16/2025
Edit	✓	FlowErrorEmailService	AutomationEmailService	ema	3/16/2025

[New Email Service](#)

Email Service Setup

 **SETUP**
Email Services

[Edit](#) [Activate](#) [Deactivate](#) [Delete](#) [Cancel](#)

Email Service Name	AIEmailHandler
Apex Class	EmailHandlerAI
Accept Attachments	Binary attachments only
Advanced Email Security Settings	☐ i
Accept Email From	All email addresses (subject to security settings)
Convert Text Attachments to Binary Attachments	☐ i
Active	☒

▼ Failure Response Settings

Over email rate limit action	Discard message
Deactivated Email Address Action	Discard message
Deactivated Email Service Action	Discard message
Unauthenticated sender action	Discard message
Unauthorized sender action	Discard message
Enable Error Routing	☐ i

Email Addresses [New Email Address](#)

Action	Email Address Name	Email Address	Context User
View Edit	EmailHandler	chatgptemailhandler@9-1msmdtsznyi7v84w86b9xrch92m60i3d6poi hod20xxnqi39c1-gl-cgbsuam.can98.apex.salesforce.com	Robert Davis

Review the AutomationEmailService

```
public with sharing class AutomationEmailService implements Messaging.InboundEmailHandler {  
    public Messaging.InboundEmailResult handleInboundEmail(Messaging.InboundEmail email, Messaging.InboundEnvelope envelope) {  
  
        Messaging.InboundEmailResult result = new Messaging.InboundEmailResult();  
  
        try {  
            // Log the incoming email details using Nebula Logger  
            Logger.error('Inbound Email Received');  
            Logger.setField(LogEntryEvent__e.Email_Subject__c, email.subject);  
            Logger.setField(LogEntryEvent__e.Email_From__c, email.fromAddress);  
            Logger.setField(LogEntryEvent__e.Email_Body__c, email.plainTextBody);  
            Logger.setField(LogEntryEvent__e.Email_To__c, String.join(email.toAddresses, ','));  
            Logger.saveLog();  
  
            result.success = true;  
        } catch (Exception e) {  
            Logger.error('Error handling inbound email: ' + e.getMessage()+' stack trace: '+e.getStackTraceString());  
            Logger.saveLog();  
  
            result.success = false;  
            result.message = 'Error: ' + e.getMessage();  
        }  
  
        return result;  
    }  
}
```

- Review in VS Code
- Might use truncation in case too long.
- Create an Event and forward Automated Process User Email Address

Capture Errors Over Time

The screenshot displays the Nebula Logger application interface. At the top, there's a search bar and navigation tabs for 'Log Entries', '00001050 | Case', '00001052 | Case', 'Log-000056 | ...', 'Recently View...', and 'Log-000057 | ...'. The main content area is divided into two panels. The left panel shows a 'Stack Trace' for 'Class.AutomationEmailService.handleInboundEmail: line 8, column 1'. Below this, it details an email subject 'FW: An error occurred with your "Generate Contact Error" flow' and an email body that includes a Salesforce logo and a message about an error in the 'Generate Contact Error' flow. The right panel shows the 'Logging Level' set to 'ERROR', the 'Username' as 'emailservices2025031501630@agentforce.com', and the 'Event UUID' as '27765b57-6f36-4e64-b71f-c694afc37b97'. At the bottom right, there are links for 'Logger Settings' and 'History'.

- Flow errors often go unnoticed unless users report them.
- Able to Log and reduce inbox and see errors over time.
- Searchable and reportable.
- Easier triage and faster response.

Real-World Story: Snap, Send, Save: Simplifying Expense Tracking

 **Real-World Example: No app. No clicks. Just an email with a receipt.**

No Receipt left behind

Scenario: Remove Friction of uploading receipts, process Expenses quicker.

Problem: Uploading expense reports may require using desktop app and not terribly user friendly.

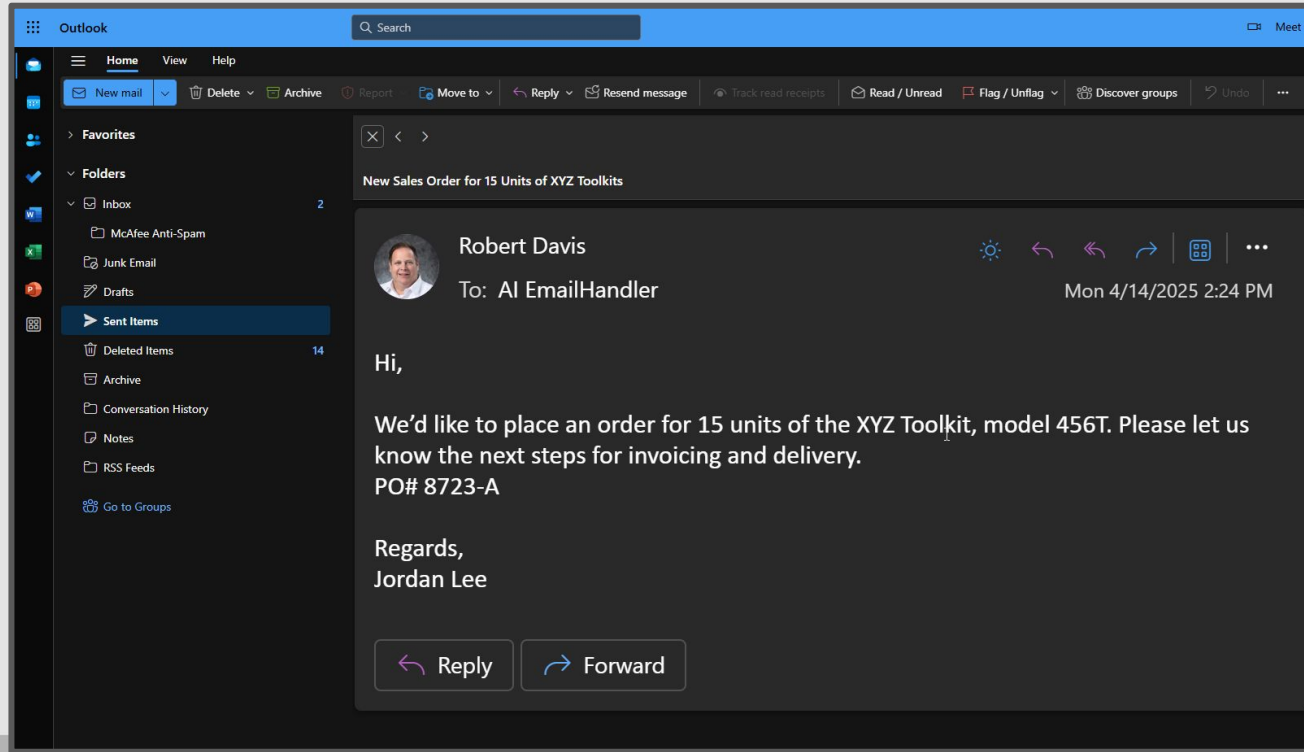
Limitations of Email-to-Case: No support for custom workflows, other objects

Solution (Apex Email Service):

- Sales reps emailed photos of receipts from the road
- Email sent to expenses@company.com
- Apex Email Service logic:
 -  Match to existing open **Expense__c** by sender
 -  If none, create new **Expense__c** record
 -  Attach image file to record
- Later enhancement: OCR to auto-fill vendor, amount, and date



Send / Forward Email



- Create an Opportunity not an option for out of the box
- See in industries where there is a unique identifier

Created Opportunity

The screenshot displays the Nebula Logger interface. At the top, there's a search bar and navigation tabs for 'Cases', 'Robert Davis | Cont...', and 'Opportuni...'. The main header shows the opportunity title 'Opportunity from Email: New Sales Order for 15 Units of XYZ Toolkits' with buttons for '+ Follow', 'New Case', 'New Note', and 'Clone'. Below the title, key details are listed: Account Name 'Edge Communications', Close Date '5/14/2025', Amount, and Opportunity Owner 'Robert Davis'. A progress bar indicates the current stage is 'Prospecting', with other stages like 'Qualification', 'Needs Analysis', 'Value Proposition', 'Id. Decision Mak...', 'Perception Analy...', 'Proposal/Price Q...', 'Negotiation/Revi...', and 'Closed'. A 'Mark Stage as Complete' button is at the end of the bar. The 'Activity' section shows options to 'New Task', 'Log a Call', 'New Event', and 'Email', along with filters and a 'No activities to show' message. The 'Related' section includes 'Products (0)' and 'Notes & Attachments (0)' with an 'Upload Files' button. The footer contains a URL and links for 'Logger Settings' and 'History'.

Nebula Logger | Cases | Robert Davis | Cont... | Opportuni...

Opportunity
Opportunity from Email: New Sales Order for 15 Units of XYZ Toolkits

+ Follow | New Case | New Note | Clone

Account Name: [Edge Communications](#) | Close Date: 5/14/2025 | Amount: | Opportunity Owner: [Robert Davis](#)

Prospecting | Qualification | Needs Analysis | Value Proposition | Id. Decision Mak... | Perception Analy... | Proposal/Price Q... | Negotiation/Revi... | Closed | [✓ Mark Stage as Complete](#)

Activity | Details | Chatter

New Task | Log a Call | New Event | Email

Filters: All time • All activities • All types | Refresh | Expand All | View All

Upcoming & Overdue

No activities to show.
Get started by sending an email, scheduling a task, and more.

April • 2025 | This Month

Related

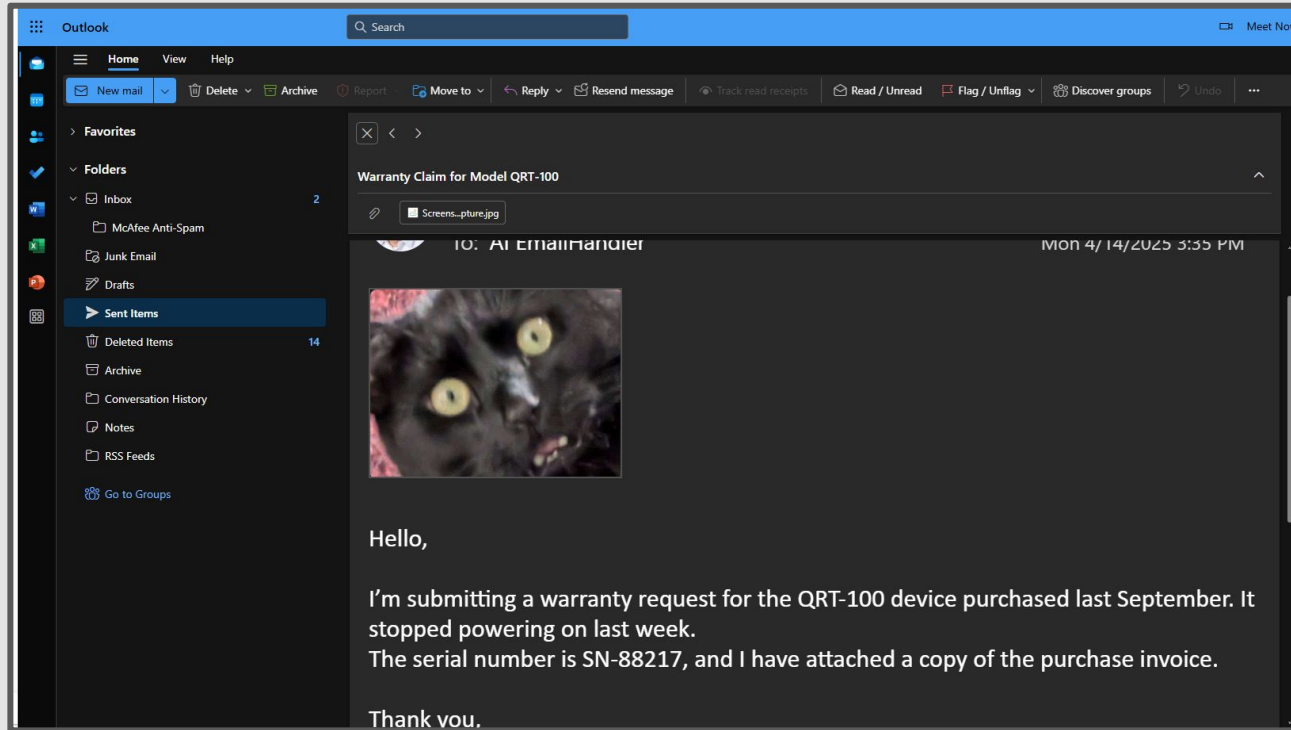
Products (0)

Notes & Attachments (0)

Upload Files | Or drop files

[Logger Settings](#) | [History](#)

Send / Forward Email



Case Created

The screenshot displays the Nebula Logger web application. At the top, there's a search bar and navigation icons. The main header shows 'Nebula Logger' and a dropdown menu for 'Cases' with a selected item '00001048 | Case'. The case title is 'Warranty: Warranty Claim for Model QRT-100'. Below the title, the case details are listed: Priority (Medium), Status (New), and Case Number (00001048). The interface is divided into two main sections: 'Feed' and 'Details'. The 'Feed' section has a 'Related' tab and contains four panels: 'Open Activities (0)' with 'New Task' and 'New Event' buttons; 'Activity History (1)' with a 'View All' button and a table of activities; 'Case Comments (0)' with a 'New' button; and 'Files (1)' with an 'Add Files' button and a file upload entry. The 'Details' section on the right lists various case attributes in two columns, each with an edit icon.

Case Information:

- Case: Warranty: Warranty Claim for Model QRT-100
- Priority: Medium
- Status: New
- Case Number: 00001048

Feed - Related

- Open Activities (0)** [New Task] [New Event]
- Activity History (1)** [View All]

Subject	Name	Task	Due Date
Email: Warranty Claim for Model QRT-100		☒	4/14/2025

[View All]
- Case Comments (0)** [New]
- Files (1)** [Add Files]
 - Screenshot_20250413_160637_Samsung_capture.jpg
Apr 14, 2025 • 24KB • jpg[View All]

Details

Case Owner: Robert Davis	Status: New
Case Number: 00001048	Priority: Medium
Contact Name: Robert Davis	Contact Phone: (512) 757-6000
Account Name: Edge Communications	Contact Email: robt.davis@outlook.com
Type:	Case Origin: Email
Case Reason:	
Web Email:	Web Company:
Web Name:	Web Phone:
Date/Time Opened: 4/14/2025, 3:35 PM	Date/Time Closed:
Product:	Engineering Req Number:
Potential Liability:	SLA Violation:

EmailHandlerAI

```
public class EmailHandlerAI implements Messaging.InboundEmailHandler {  
  
    public Messaging.InboundEmailResult handleInboundEmail(Messaging.InboundEmail email, Messaging.  
InboundEnvelope envelope) {  
        Messaging.InboundEmailResult result = new Messaging.InboundEmailResult();  
  
        // Callout to ChatGPT to classify email  
        String emailType = EmailClassifier.classifyEmail(email.subject, email.plainTextBody);  
  
        if(emailType == 'Product Inquiry'){  
            SalesforceRecordCreator.createCase(email, 'Product Inquiry:');  
        }  
        else if(emailType == 'Sales Order'){  
            SalesforceRecordCreator.createOpportunity(email);  
        }  
        else if(emailType == 'Warranty'){  
            SalesforceRecordCreator.createCase(email, 'Warranty:');  
        }  
        else if(emailType == 'Request for More Information'){  
            SalesforceRecordCreator.createCase(email, 'Request for More Information:');  
        } else{  
            SalesforceRecordCreator.createCase(email, 'Unable to Determine Email Type:');  
        }  
  
        result.success = true;  
        return result;  
    }  
}
```

- Implements Messaging.InboundEmailHandler
- Passes off to a callout to do the classification
- Comes back and creates a Case or Opportunity

EmailClassifier

```
public class EmailClassifier {  
    public static AI_Integration_Config__mdt  
    config;  
  
    public static String classifyEmail(String  
    subject, String body) {  
  
        // Initialize configuration - get the values for c  
        onfig  
        initializeConfig();  
  
        // Prepare and send request  
        HttpResponse response =  
        sendClassificationRequest(subject, body);  
  
        // Log the response  
        logResponse(response);  
  
        // Process the response  
        return processResponse(response);  
    }  
}
```

- Get api info from Custom Metadata
- Send the Classification to Gemini LLM
- Log the response
- Pass back the classification

SalesforceRecordCreator

```
public class SalesforceRecordCreator {  
  
    public static void createCase(Messaging.InboundEmail email, String type){  
        Contact cont = [  
            SELECT Id, AccountId  
            FROM Contact  
            WHERE Email = :email.fromAddress  
            LIMIT 1  
        ];  
  
        Case c = new Case();  
        c.Subject = (type == 'Unable to Determine Email Type' ?  
'Unable to Determine Email Type: ' : '') + Type+' '+email.subject;  
        c.Description = email.plainTextBody;  
        c.ContactId = cont.Id;  
        c.AccountId = cont.AccountId;  
        c.Origin = 'Email';  
        insert c;  
  
        attachEmailAndFiles(c.Id, email);  
    }  
}
```

- Create Case
- Create Task
- Create Email Message
- Create Account and Opportunity
- Create Opportunity
- Attach Attachments

THANK YOU / Q & A

Robert Davis
Solution Architect, Hike 2

🙏 “Thanks for spending time with me today — and for letting me talk about something I really enjoy building.

Slides

<https://bit.ly/4kBerrm>



LinkedIn



<https://bit.ly/3XcyG5d>

When to Use Apex Email Services

When out-of-the-box just isn't enough.

- You need to create something *other than* a Case
- You want to parse email content or attachments
- You need to route messages based on subject or sender
- You want to log Flow or system activity by email
- You want to integrate with AI, OCR, or external systems
- You want to reduce inbox chaos with automation

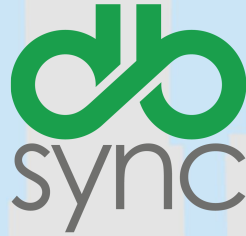
Resources

- [Salesforce CI/CD DevOps - A REAL How To](#)
- [Increase Email-to-Case Limit](#)
- [Setup Email-to-Case in Salesforce](#)
- [Using the InboundEmail Object](#)
- [Email Services in Salesforce with simple example](#)
- [InboundEmail Object fields](#)

Thank You



airSlate



CITRIN COOPERMAN®



Score
Keeper

for Salesforce



DataGroomr
Data Quality Perfected by AI