Real-World CRM Integration Scenario

The Scenario

Company: TechSupport Inc. (50 agents, 3 locations)
Location: Austin, Toronto, São Paulo
CRM: Salesforce Service Cloud
Requirement: Integrate Genesys Cloud so agents see customer context during calls

---

Business Requirements

What We Need

When customer calls:

- ├─ Agent sees: Name, account, last 3 cases, contact history
- ├─ Call logged in Salesforce (Task)
- └─ Contact list stays in sync

Manual Requirements:

- ├─ Support agents can edit notes in Salesforce
- ├─ Notes should show in next call
- └─ Compliance: GDPR deletion within 30 days

Success Metrics

- ✓ 100% of calls show customer context (no "Contact not found")
- ✓ Average call handle time reduced by 10% (less lookup time)

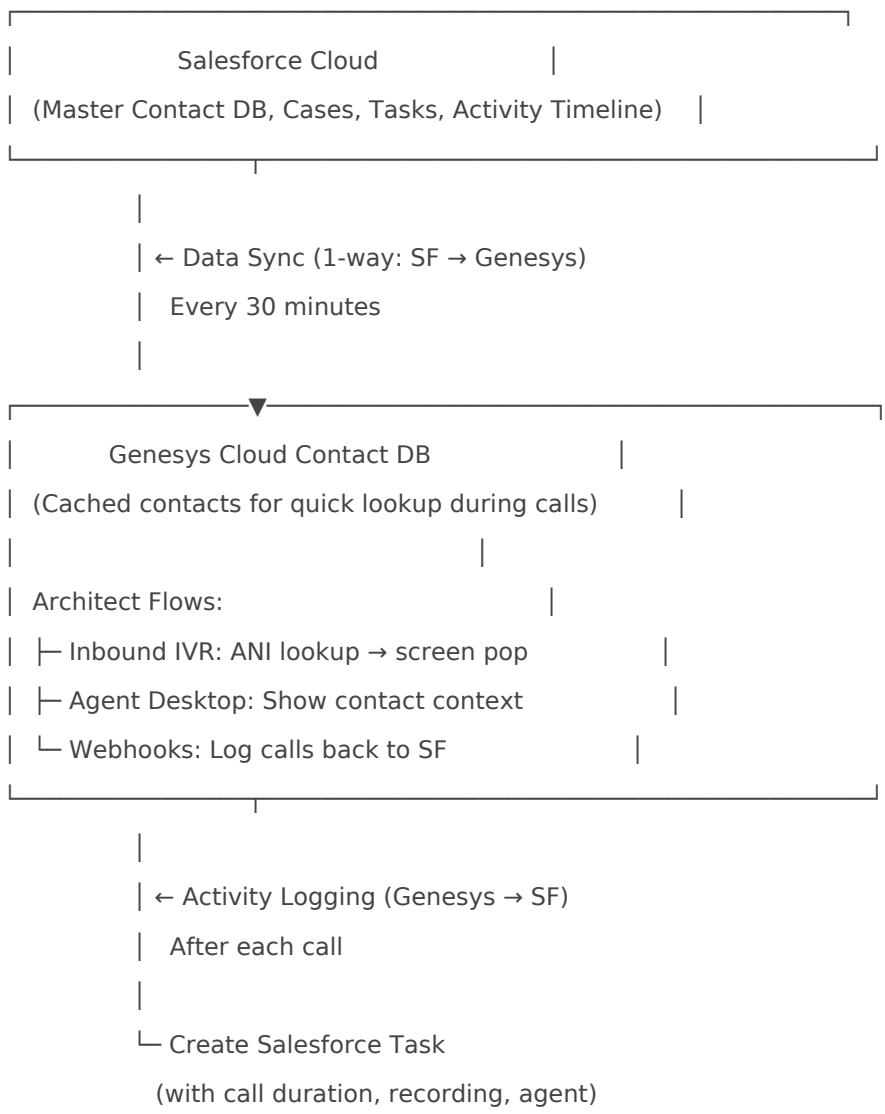
✓ Agent satisfaction > 4/5 (easy to use)

✓ Salesforce Task creation 99%+ success (activity logging)

✓ Contact data fresh (synced daily)

✓ Zero GDPR compliance violations

Architecture



Phase 1: Screen Pop (Week 1-2)

Goal: When customer calls, agent sees their record

Step 1: Create Salesforce Apex Endpoint

```
// ContactLookupService.cls

@RestResource(urlMapping='/contact-lookup')
global class ContactLookupService {
  @HttpPost
  global static Response lookup(String phoneNumber) {
    Response response = new Response();

    try {
      // Normalize phone (remove formatting)
      String normalizedPhone = normalizePhone(phoneNumber);

      // Search for contact
      List<Contact> contacts = [
        SELECT Id, FirstName, LastName, Email, Phone,
               AccountId, Account.Name
        FROM Contact
        WHERE Phone = :normalizedPhone
           OR MobilePhone = :normalizedPhone
        LIMIT 1
      ];

      if (contacts.isEmpty()) {
        response.success = false;
        return response;
      }

      Contact contact = contacts[0];
      response.success = true;
      response.contactId = contact.Id;
      response.firstName = contact.FirstName;
```

```
      response.lastName = contact.LastName;
      response.email = contact.Email;
      response.accountId = contact.AccountId;
      response.accountName = contact.Account?.Name;

    } catch (Exception e) {
      response.success = false;
      response.error = e.getMessage();
    }

    return response;
  }

  private static String normalizePhone(String phone) {
    // Remove all non-digits
    String digits = phone.replaceAll('[^0-9]', '');

    // Add +1 if US number (10 digits)
    if (digits.length() == 10) {
      digits = '1' + digits;
    }

    return '+' + digits;
  }

  global class Response {
    public Boolean success;
    public String contactId;
    public String firstName;
    public String lastName;
    public String email;
    public String accountId;
    public String accountName;
    public String error;
  }
}
```

Step 2: Create Genesys Data Action

In **Architect → Data Actions**:

```
Name: lookup-contact-by-phone
Method: POST
URL: https://your-instance.salesforce.com/services/apexrest/contact-lookup

Input:
  phoneNumber: ${interaction.caller.phoneNumber}

Output:
  success: ${dataAction.response.success}
  contactId: ${dataAction.response.contactId}
  firstName: ${dataAction.response.firstName}
  lastName: ${dataAction.response.lastName}
  accountId: ${dataAction.response.accountId}
  accountName: ${dataAction.response.accountName}
```

Step 3: Create Architect Inbound Flow

```
START
  │
  ├─ Play: "Thank you for calling TechSupport..."
  │
  ├─ Data Action: lookup-contact-by-phone
  │   Input: ANI = ${interaction.caller.phoneNumber}
  │
  ├─ Decision: ${dataAction.result.success}?
  │   ├─ YES:
  │   │  ├─ Set interaction attributes:
  │   │  │  ├─ contact_id = ${dataAction.result.contactId}
  │   │  │  ├─ contact_name = ${dataAction.result.firstName} ${dataAction.result.lastName}
  │   │  │  ├─ account_id = ${dataAction.result.accountId}
  │   │  │  └─ account_name = ${dataAction.result.accountName}
  │   │  │
  │   │  └─ Transfer to Support Queue
  │   │
  │   └─ NO:
  │      ├─ Play: "Please hold while we locate your account..."
  │      └─ Transfer to Support Queue (no attributes)
```

```
  |
  └─ DISCONNECT
```

Expected Result

```
Customer dials: +1-512-555-1234
  ↓
Agent receives call
  ↓
Agent's Genesys desktop shows:
  Contact Name: John Doe
  Account: Acme Corp
  Email: john@acmecorp.com
  Last 3 Cases: [list]
  Contact History: [last 5 calls]
```

Phase 2: Contact Sync (Week 2-3)

Goal: Keep Genesys contact list in sync with Salesforce

Step 1: Create Sync Job

```
// sync-job.js (runs every 30 minutes)

const SalesforceGenesysSync = require('./lib/sync');

async function syncDaily() {
  const sync = new SalesforceGenesysSync({
    sfInstance: process.env.SALESFORCE_INSTANCE,
    sfToken: process.env.SALESFORCE_TOKEN,
    gzToken: process.env.GENESYS_TOKEN
  });
```

```
  const result = await sync.runFullSync();

  console.log(`✓ Sync complete: ${result.created} created, ${result.updated} updated`);

  if (result.errors.length > 0) {
    await sendAlert('warning', result);
  }
}

syncDaily().catch(error => {
  console.error('Sync failed:', error);
  process.exit(1);
});
```

Step 2: Deploy as Lambda (AWS)

```
# serverless.yml

service: techsupport-crm-sync

provider:
  name: aws
  runtime: nodejs18.x
  environment:
    SALESFORCE_INSTANCE: ${env:SALESFORCE_INSTANCE}
    SALESFORCE_TOKEN: ${env:SALESFORCE_TOKEN}
    GENESYS_TOKEN: ${env:GENESYS_TOKEN}

functions:
  sync:
    handler: sync-job.syncDaily
    events:
      - schedule:
          rate: rate(30 minutes)  # Every 30 minutes
          enabled: true

resources:
  Resources:
```

```
  SyncLogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: /aws/lambda/techsupport-crm-sync
      RetentionInDays: 30
```

Deploy: `serverless deploy`

Step 3: Monitor Sync

```
Logs:
  2026-03-14 10:00:00 ✓ Fetched 2345 contacts from Salesforce
  2026-03-14 10:00:05 ✓ Found 2100 existing Genesys contacts
  2026-03-14 10:00:30 ✓ Created: 50, Updated: 200, Skipped: 5
  2026-03-14 10:00:31 Duration: 31 seconds

Metrics:
  - Success rate: 99.7%
  - Avg duration: 32 sec
  - Contacts synced: 250/day
```

Phase 3: Activity Logging (Week 3-4)

Goal: After call, log details in Salesforce Task

Step 1: Enable Genesys Webhooks

In **Genesys Admin → Integrations → Webhooks**:

Event: conversation.ended

URL: https://your-backend.com/webhook/call-ended

Payload: Include all details (recording ID, agent, duration)

Retries: 3 times

Step 2: Create Webhook Handler

```
// webhook-handler.js

app.post('/webhook/call-ended', async (req, res) => {
  const {
    conversationId,
    callerId,
    durationSeconds,
    recordingId,
    agentName,
    queueName,
    attributes
  } = req.body;

  try {
    // 1. Find matching Salesforce contact
    const contact = await findContactByPhone(callerId);
    if (!contact) {
      console.warn(`Contact not found for ${callerId}`);
      return res.status(200).json({ message: 'Contact not found' });
    }

    // 2. Get recording URL
    const recordingUrl = await getRecordingUrl(recordingId);

    // 3. Create Salesforce Task
    const task = {
      Subject: `Call with ${contact.Name}`,
      Description: `
Call Details:
  Duration: ${Math.floor(durationSeconds / 60)} min
  Agent: ${agentName}
```

```
  Queue: ${queueName}
  Recording: ${recordingUrl || 'Not available'}
    `.trim(),
    WhoId: contact.Id,
    WhatId: contact.AccountId,
    ActivityDate: new Date().toISOString().split('T')[0],
    CallType: 'Inbound',
    Status: 'Completed',
    Type: 'Call'
  };

  const taskResult = await createSalesforceTask(task);
  console.log(`✓ Task created: ${taskResult.id}`);

  // 4. Update Contact's LastActivityDate
  await updateSalesforceContact(contact.Id, {
    LastActivityDate: new Date().toISOString().split('T')[0]
  });

  res.status(200).json({ taskId: taskResult.id });

 } catch (error) {
  console.error('Webhook error:', error);
  res.status(500).json({ error: error.message });
 }
});
```

Step 3: Verify Logging

```
In Salesforce Contact record:
  Activity Timeline shows:
    - Call with John Doe (10 min)
      Agent: Sarah Smith
      Queue: Support
      Recording: [link]
      [Add note/next steps]
```

Phase 4: GDPR Compliance (Week 4)

Goal: Handle data deletion requests

Step 1: Create GDPR Deletion Procedure

```
// gdpr-delete.js

async function handleGDPRDeletion(email) {
  console.log(`□□ GDPR Deletion: ${email}`);

  // 1. Find contact
  const sfContact = await findSalesforceContactByEmail(email);
  const gzContact = await findGenesysContactByEmail(email);

  // 2. Delete from both
  if (sfContact) {
    await deleteSalesforceContact(sfContact.Id);
  }

  if (gzContact) {
    await deleteGenesysContact(gzContact.id);
  }

  // 3. Delete recordings
  const recordings = await findRecordingsByPhone(email);
  for (const rec of recordings) {
    await deleteRecording(rec.id);
  }

  // 4. Log deletion
  await logGDPRDeletion({
    email,
    deletedAt: new Date(),
```

```
    deletedFrom: ['salesforce', 'genesys', 'recordings']
  });

  console.log(`  Deleted: ${email}`);
}
```

Step 2: Document Privacy Policy

Update website:

```
We collect:
  - Name, email, phone (for customer service)
  - Call recordings (for 7 years per compliance)

You can:
  - Request deletion: privacy@techsupport.com
  - We'll delete within 30 days
```

Go-Live Plan

Week 1: Preparation

- ☐ Test screen pop in dev environment
- ☐ Train agents on new features
- ☐ Set up monitoring

Week 2: Screen Pop

- ☐ Deploy Salesforce Apex
- ☐ Deploy Genesys Data Action
- ☐ Deploy Architect Flow
- ☐ Pilot with 5 agents
- ☐ Full rollout if successful

Week 3: Contact Sync

- [ ] Deploy sync job to Lambda
- [ ] Monitor logs for errors
- [ ] Verify contacts are syncing
- [ ] Set up Slack alerts

Week 4: Activity Logging

- [ ] Deploy webhook handler
- [ ] Configure Genesys webhooks
- [ ] Test call logging
- [ ] Verify Salesforce tasks created

Week 5: GDPR & Training

- [ ] Document privacy procedures
- [ ] Train staff on GDPR
- [ ] Set up GDPR deletion process
- [ ] Final full-system testing

Expected Results

Before Integration

Agent experience:

- Customer calls
- Agent types customer name into Salesforce search (30 sec)
- Wait for results
- Navigate to account/cases
- Get context, THEN handle call

Average handle time: 8 minutes

Agent satisfaction: 3/5

Customer satisfaction: 3.5/5

After Integration

Agent experience:

- Customer calls
- Record auto-pops (1 sec)
- Agent sees name, account, last cases
- Agent handles call with context immediately
- Call logged to Salesforce automatically

Average handle time: 7 minutes (12.5% improvement)

Agent satisfaction: 4.5/5

Customer satisfaction: 4.2/5

Monitoring & Maintenance

Weekly Checks

☐ Screen pop success rate > 99%

☐ Contact sync duration < 2 min

☐ Activity logging > 99% success

☐ GDPR deletion requests: 0

☐ Errors: < 5 per week

Monthly Review

☐ Agent feedback on usability

☐ Performance trends

☐ Cost analysis

☐ Compliance status

☐ Planned improvements

Budget Estimate

Implementation:

- ├─ Salesforce Apex: $3,000 (5 days)
- ├─ Genesys Architect: $2,000 (3 days)
- ├─ Sync Job (Lambda): $1,500 (2 days)
- ├─ Webhook Handler: $1,500 (2 days)
- ├─ Testing & QA: $2,000 (3 days)
- └─ Total Dev: $10,000

Operations (annual):

- ├─ AWS Lambda: $50/month ($600/year)
- ├─ Genesys API calls: $200/month ($2,400/year)
- ├─ Salesforce: Included in license
- ├─ Monitoring: $100/month ($1,200/year)
- └─ Total Ops: $4,200/year

ROI:

- ├─ Productivity gain: 12.5% = ~200 agents × 30 min/day
- ├─ Annual value: 200 × 250 working days × 0.5 hours × $25/hr = $625,000
- ├─ Cost: $10,000 dev + $4,200 ops = $14,200
- └─ Payback: < 1 week

Related Topics

- Chapter 12: Screen Pop Architecture & Implementation
- Chapter 12: Contact Sync Patterns
- Chapter 12: Activity Logging & Webhooks
- Chapter 12: GDPR & Data Governance
- Chapter 11: API Endpoints Reference

Revision #1
Created 15 March 2026 00:46:17 by Cesar Gzz
Updated 15 March 2026 00:46:27 by Cesar Gzz