ServiceNow-TEAM Integration Implementation Approach

Overview

This document outlines the implementation approach for automatically creating TEAM access requests based on ServiceNow ticket creation. The integration will enable seamless temporary elevated access management triggered by ServiceNow incident/request workflows.

Current TEAM Architecture Analysis

Core Components

- **Frontend**: React application with AWS Amplify hosting
- **Backend**: AWS AppSync GraphQL API with Lambda resolvers
- **Authentication**: AWS Cognito User Pools with IAM Identity Center integration
- **Database**: DynamoDB tables for requests, policies, approvers, and settings
- **Workflow**: Step Functions for grant/revoke/approval processes
- **Notifications**: SNS for email/SMS notifications

Key Lambda Functions

- `teamRouter` : Main orchestration function handling request lifecycle
- `teamStatus` : Status management and updates
- `teamNotifications` : Notification handling
- `teamGetPermissionSets` : IAM Identity Center permission set retrieval

GraphQL Schema

- `Requests` table with fields: email, accountId, role, duration, justification, status, ticketNo

Implementation Architecture

1. ServiceNow Integration Components

A. ServiceNow Business Rule

```
// ServiceNow Business Rule: TEAM Access Request Trigger
(function executeRule(current, previous) {
    // Trigger conditions
    if (current.category == 'aws_access' && current.state == 'New') {
        var teamIntegration = new TeamAccessIntegration();
        teamIntegration.createAccessRequest(current);
    }
})(current, previous);
```

B. ServiceNow Script Include

```
var TeamAccessIntegration = Class.create();
TeamAccessIntegration.prototype = {
    initialize: function() {
        this.teamApiEndpoint = gs.getProperty('team.api.endpoint');
        this.teamApiKey = gs.getProperty('team.api.key');
    },

    createAccessRequest: function(ticket) {
        var payload = this._buildRequestPayload(ticket);
        var response = this._callTeamAPI(payload);

        if (response.success) {
            ticket.u_team_request_id = response.requestId;
            ticket.update();
        }
    },

    _buildRequestPayload: function(ticket) {
        return {
            email: ticket.caller_id.email,
            accountId: ticket.u_aws_account_id,
            role: ticket.u_permission_set,
            duration: parseInt(ticket.u_access_duration),
            justification: ticket.short_description,
            ticketNo: ticket.number,
            requester: ticket.caller_id.user_name
```

```
        };
    }
};
```

2. TEAM API Enhancement

A. New Lambda Function: `teamServiceNowIntegration`

```
import json
import boto3
import os
from datetime import datetime, timedelta

def lambda_handler(event, context):
    """
    Handle ServiceNow integration requests
    """
    try:
        # Parse request
        body = json.loads(event['body'])

        # Validate ServiceNow authentication
        if not validate_servicenow_auth(event['headers']):
            return error_response(401, "Unauthorized")

        # Create TEAM request
        request_id = create_team_request(body)

        return success_response({
            'requestId': request_id,
            'status': 'created'
        })

    except Exception as e:
        return error_response(500, str(e))

def create_team_request(payload):
    """Create TEAM access request from ServiceNow data"""

    # Validate required fields
    required_fields = ['email', 'accountId', 'role', 'duration',
'justification', 'ticketNo']
    for field in required_fields:
        if field not in payload:
            raise ValueError(f"Missing required field: {field}")
```

```
    # Generate request ID
    request_id = generate_request_id()

    # Calculate end time
    start_time = datetime.utcnow()
    end_time = start_time + timedelta(hours=payload['duration'])

    # Prepare DynamoDB item
    request_item = {
        'id': request_id,
        'email': payload['email'],
        'accountId': payload['accountId'],
        'role': payload['role'],
        'duration': payload['duration'],
        'justification': payload['justification'],
        'ticketNo': payload['ticketNo'],
        'status': 'pending_approval',
        'startTime': start_time.isoformat(),
        'endTime': end_time.isoformat(),
        'createdAt': start_time.isoformat(),
        'source': 'servicenow'
    }

    # Save to DynamoDB
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table(os.environ['REQUESTS_TABLE_NAME'])
    table.put_item(Item=request_item)

    # Trigger approval workflow
    trigger_approval_workflow(request_item)

    return request_id
```

B. API Gateway Integration

```
# CloudFormation template addition
ServiceNowIntegrationAPI:
  Type: AWS::ApiGateway::RestApi
  Properties:
    Name: TEAM-ServiceNow-Integration

ServiceNowResource:
  Type: AWS::ApiGateway::Resource
  Properties:
    RestApiId: !Ref ServiceNowIntegrationAPI
```

```
      ParentId: !GetAtt ServiceNowIntegrationAPI.RootResourceId
      PathPart: servicenow

  ServiceNowMethod:
    Type: AWS::ApiGateway::Method
    Properties:
      RestApiId: !Ref ServiceNowIntegrationAPI
      ResourceId: !Ref ServiceNowResource
      HttpMethod: POST
      AuthorizationType: AWS_IAM
      Integration:
        Type: AWS_PROXY
        IntegrationHttpMethod: POST
        Uri: !Sub 'arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-
31/functions/${TeamServiceNowIntegrationFunction.Arn}/invocations'
```

3. Enhanced Request Processing

A. Modified teamRouter Function

```
def handle_servicenow_request(event):
    """Enhanced handling for ServiceNow-originated requests"""

    request_data = event['arguments']['input']

    # Additional validation for ServiceNow requests
    if request_data.get('source') == 'servicenow':
        # Validate ServiceNow ticket exists
        if not validate_servicenow_ticket(request_data['ticketNo']):
            raise ValueError("Invalid ServiceNow ticket")

        # Auto-approve if configured
        if should_auto_approve(request_data):
            request_data['status'] = 'approved'
            trigger_grant_workflow(request_data)
        else:
            trigger_approval_workflow(request_data)

    return process_standard_request(request_data)

def should_auto_approve(request_data):
    """Determine if request should be auto-approved based on
policies"""

    # Check auto-approval policies
```

```
    policies = get_auto_approval_policies(
        request_data['accountId'],
        request_data['role']
    )

    for policy in policies:
        if (request_data['duration'] <= policy['max_duration'] and
            request_data['email'] in policy['allowed_users']):
            return True

    return False
```

4. Bidirectional Status Sync

A. TEAM to ServiceNow Status Updates

```
def update_servicenow_status(request_id, status, comment=None):
    """Update ServiceNow ticket with TEAM request status"""

    # Get request details
    request = get_request_by_id(request_id)
    if not request or not request.get('ticketNo'):
        return

    # Prepare ServiceNow update
    snow_payload = {
        'state': map_team_status_to_snow(status),
        'work_notes': f"TEAM Request {request_id}: {status}",
        'u_team_status': status
    }

    if comment:
        snow_payload['work_notes'] += f" - {comment}"

    # Call ServiceNow API
    update_servicenow_ticket(request['ticketNo'], snow_payload)

def map_team_status_to_snow(team_status):
    """Map TEAM status to ServiceNow state"""
    mapping = {
        'pending_approval': 'Pending Approval',
        'approved': 'Approved',
        'active': 'In Progress',
        'expired': 'Closed Complete',
        'revoked': 'Closed Complete',
```

```
            'rejected': 'Closed Incomplete'
        }
        return mapping.get(team_status, 'New')
```

B. ServiceNow to TEAM Status Updates

```
// ServiceNow Business Rule: TEAM Status Sync
(function executeRule(current, previous) {
    if (current.u_team_request_id &&
        current.state.changesTo('Cancelled')) {

        var teamIntegration = new TeamAccessIntegration();
        teamIntegration.revokeAccess(current.u_team_request_id,
                                     'Cancelled via ServiceNow');
    }
})(current, previous);
```

5. Security Implementation

A. Authentication & Authorization

```
def validate_servicenow_auth(headers):
    """Validate ServiceNow API authentication"""

    # Option 1: API Key validation
    api_key = headers.get('X-API-Key')
    if api_key != os.environ['SERVICENOW_API_KEY']:
        return False

    # Option 2: JWT token validation
    token = headers.get('Authorization', '').replace('Bearer ', '')
    if not validate_jwt_token(token):
        return False

    # Option 3: IP whitelist validation
    source_ip = headers.get('X-Forwarded-For', '').split(',')[0]
    if source_ip not in get_allowed_ips():
        return False

    return True

def validate_jwt_token(token):
    """Validate JWT token from ServiceNow"""
    try:
```

```
            import jwt
            payload = jwt.decode(
                token,
                os.environ['JWT_SECRET'],
                algorithms=['HS256']
            )
            return payload.get('iss') == 'servicenow'
        except:
            return False
```

B. Data Validation & Sanitization

```
def validate_request_data(data):
    """Validate and sanitize ServiceNow request data"""

    # Email validation
    if not re.match(r'^[^@]+@[^@]+\.[^@]+$', data['email']):
        raise ValueError("Invalid email format")

    # Account ID validation
    if not re.match(r'^\d{12}$', data['accountId']):
        raise ValueError("Invalid AWS account ID")

    # Duration limits
    if not (1 <= data['duration'] <= 168):  # 1 hour to 1 week
        raise ValueError("Duration must be between 1 and 168 hours")

    # Sanitize text fields
    data['justification'] = sanitize_text(data['justification'])
    data['ticketNo'] = sanitize_text(data['ticketNo'])

    return data
```

Configuration Requirements

ServiceNow Configuration

```
// System Properties
team.api.endpoint = https://api.team.company.com/servicenow
team.api.key = <secure-api-key>
```

```
team.auto.approve.roles = ReadOnlyAccess,ViewOnlyAccess
team.max.duration.hours = 24
```

TEAM Configuration

```
{
  "servicenow_integration": {
    "enabled": true,
    "api_key": "<secure-api-key>",
    "allowed_ips": ["10.0.0.0/8"],
    "auto_approval": {
      "enabled": true,
      "max_duration_hours": 8,
      "allowed_roles": ["ReadOnlyAccess"]
    }
  }
}
```