Contents

RADIANT Swift Deployer - ACTUAL SOURCE CODE - Part 2

File: Services/AIAssistantService.swift

```swift
import Foundation
import Security

/// AI Assistant Service for Claude API integration
/// Provides intelligent deployment guidance and troubleshooting
actor AIAssistantService {

    // MARK: - Types

    enum AIError: Error, LocalizedError {
        case noApiKey
        case invalidResponse
        case networkError(String)
        case rateLimited
        case apiError(String)

        var errorDescription: String? {
            switch self {
            case .noApiKey:
                return "No API key configured. Add your Claude API key in Settings."
            case .invalidResponse:
                return "Invalid response from AI service"
            case .networkError(let message):
                return "Network error: \(message)"
            case .rateLimited:
                return "Rate limited. Please wait a moment."
            case .apiError(let message):
                return "AI API error: \(message)"
            }
        }
    }

    struct Message: Codable {
        let role: String
        let content: String
    }
```

```swift
struct ChatRequest: Codable {
    let model: String
    let max_tokens: Int
    let messages: [Message]
    let system: String?
}

struct ChatResponse: Codable {
    struct Content: Codable {
        let type: String
        let text: String
    }
    let id: String
    let content: [Content]
}

// MARK: - Properties

private let keychainService = "com.radiant.deployer"
private let apiKeyAccount = "claude-api-key"
private let baseURL = "https://api.anthropic.com/v1"
private let model = "claude-3-5-sonnet-20241022"

private var isConnected = false
private var lastConnectionCheck: Date?
private let connectionCheckInterval: TimeInterval = 60

private let systemPrompt = """
You are the RADIANT Deployer AI Assistant, helping users deploy and manage AWS infrastructu

Your capabilities:
- Explain deployment steps and configurations
- Troubleshoot deployment errors
- Recommend optimal tier configurations
- Explain AWS resource costs
- Guide users through credential setup

Be concise, technical, and helpful. If you don't know something, say so.
Always prioritize security best practices.
"""

// MARK: - Keychain Management

func saveApiKey(_ apiKey: String) throws {
    let data = apiKey.data(using: .utf8)!

    let query: [String: Any] = [
        kSecClass as String: kSecClassGenericPassword,
        kSecAttrService as String: keychainService,
        kSecAttrAccount as String: apiKeyAccount,
    ]

    // Delete existing
    SecItemDelete(query as CFDictionary)

    // Add new
    var newQuery = query
    newQuery[kSecValueData as String] = data

    let status = SecItemAdd(newQuery as CFDictionary, nil)
    guard status == errSecSuccess else {
        throw AIError.apiError("Failed to save API key: \(status)")
    }
}

func getApiKey() -> String? {
    let query: [String: Any] = [
        kSecClass as String: kSecClassGenericPassword,
        kSecAttrService as String: keychainService,
        kSecAttrAccount as String: apiKeyAccount,
        kSecReturnData as String: true,
    ]

    var result: AnyObject?
    let status = SecItemCopyMatching(query as CFDictionary, &result)

    guard status == errSecSuccess,
          let data = result as? Data,
          let apiKey = String(data: data, encoding: .utf8) else {
        return nil
    }

    return apiKey
}

func deleteApiKey() {
    let query: [String: Any] = [
        kSecClass as String: kSecClassGenericPassword,
        kSecAttrService as String: keychainService,
        kSecAttrAccount as String: apiKeyAccount,
    ]
    SecItemDelete(query as CFDictionary)
}

// MARK: - Connection Management
```

```swift
func checkConnection() async -> Bool {
    // Use cached result if recent
    if let lastCheck = lastConnectionCheck,
       Date().timeIntervalSince(lastCheck) < connectionCheckInterval {
        return isConnected
    }

    guard let apiKey = getApiKey(), !apiKey.isEmpty else {
        isConnected = false
        return false
    }

    do {
        // Simple test request
        let _ = try await sendMessage("Hello", context: nil)
        isConnected = true
        lastConnectionCheck = Date()
        return true
    } catch {
        isConnected = false
        lastConnectionCheck = Date()
        return false
    }
}

// MARK: - Chat Methods

func sendMessage(_ message: String, context: [Message]?) async throws -> String {
    guard let apiKey = getApiKey() else {
        throw AIError.noApiKey
    }

    var messages = context ?? []
    messages.append(Message(role: "user", content: message))

    let request = ChatRequest(
        model: model,
        max_tokens: 1024,
        messages: messages,
        system: systemPrompt
    )

    var urlRequest = URLRequest(url: URL(string: "\(baseURL)/messages")!)
    urlRequest.httpMethod = "POST"
    urlRequest.setValue("application/json", forHTTPHeaderField: "Content-Type")
    urlRequest.setValue(apiKey, forHTTPHeaderField: "x-api-key")
    urlRequest.setValue("2023-06-01", forHTTPHeaderField: "anthropic-version")
    urlRequest.httpBody = try JSONEncoder().encode(request)

    let (data, response) = try await URLSession.shared.data(for: urlRequest)

    guard let httpResponse = response as? HTTPURLResponse else {
        throw AIError.invalidResponse
    }

    if httpResponse.statusCode == 429 {
        throw AIError.rateLimited
    }

    guard httpResponse.statusCode == 200 else {
        let errorMessage = String(data: data, encoding: .utf8) ?? "Unknown error"
        throw AIError.apiError(errorMessage)
    }

    let chatResponse = try JSONDecoder().decode(ChatResponse.self, from: data)
    return chatResponse.content.first?.text ?? ""
}

// MARK: - Specialized Methods

func explainDeploymentStep(_ step: String, tier: Int) async throws -> String {
    let prompt = """
    Explain this deployment step for a Tier \(tier) RADIANT deployment:

    Step: \(step)

    Keep the explanation concise (2-3 sentences) and mention any important considerations.
    """
    return try await sendMessage(prompt, context: nil)
}

func troubleshootError(_ error: String, context: String) async throws -> String {
    let prompt = """
    Help troubleshoot this deployment error:

    Error: \(error)

    Context: \(context)

    Provide:
    1. Likely cause
    2. Recommended fix
    3. Prevention tip
    """
    return try await sendMessage(prompt, context: nil)
}

func recommendTier(requirements: [String: Any]) async throws -> String {
    let requirementsJson = try JSONSerialization.data(withJSONObject: requirements)
    let requirementsString = String(data: requirementsJson, encoding: .utf8) ?? "{}"

    let prompt = """
    Based on these requirements, recommend the best RADIANT subscription tier (1-7):

    Requirements: \(requirementsString)

    Tiers:
    1: Free - 10K tokens/month
    2: Starter - 100K tokens/month
    3: Pro - 500K tokens/month
    4: Business - 2M tokens/month
    5: Enterprise - 10M tokens/month
    6: Enterprise Plus - 50M tokens/month
    7: Unlimited - Custom

    Recommend a tier and explain why.
    """
    return try await sendMessage(prompt, context: nil)
}

func estimateCosts(tier: Int, region: String, features: [String]) async throws -> String {
    let prompt = """
    Estimate monthly AWS costs for:
    - RADIANT Tier: \(tier)
    - Region: \(region)
    - Features: \(features.joined(separator: ", "))

    Break down by:
    1. Compute (Lambda, ECS)
    2. Database (Aurora)
    3. Storage (S3)
    4. Network (API Gateway, CloudFront)
    5. AI (if self-hosted models)

    Give rough estimates and note variables.
    """
    return try await sendMessage(prompt, context: nil)
}

func generateMigrationPlan(from: String, to: String) async throws -> String {
    let prompt = """
    Generate a migration plan from \(from) to \(to) for RADIANT.
    Include:
    1. Pre-migration checklist
    2. Migration steps
    3. Rollback procedure
    4. Verification steps
    5. Estimated downtime
    """
    return try await sendMessage(prompt, context: nil)
}

// MARK: - PROMPT-33 Required Methods

/// Error translation result
struct ErrorTranslation: Sendable {
    let userFriendlyMessage: String
    let technicalDetails: String
    let suggestedAction: String
    let severity: String // "critical", "high", "medium", "low"
}

/// Recovery recommendation result
struct RecoveryRecommendation: Sendable {
    let action: String // "rollback", "retry", "manual"
    let confidence: Double // 0.0-1.0
    let reason: String
    let steps: [String]
    let alternativeActions: [String]
}

/// AI assessment result
struct AIAssessment: Sendable {
    let riskLevel: String // "low", "medium", "high", "critical"
    let migrationComplexity: String
    let estimatedDuration: String
    let warnings: [String]
    let recommendations: [String]
}

/// Generated release notes
struct GeneratedReleaseNotes: Sendable {
    let summary: String
    let features: [String]
    let fixes: [String]
    let breakingChanges: [String]
    let upgradeNotes: String
}

/// Explain a deployment event with context
```

```
func explain(context: String, event: String) async throws -> String {
    let prompt = """
    Explain this deployment event in plain language:

    Context: \(context)
    Event: \(event)

    Keep the explanation concise (2-3 sentences) and user-friendly.
    """

    do {
        return try await sendMessage(prompt, context: nil)
    } catch {
        return fallbackExplanation(for: event)
    }
}

/// Translate technical error to user-friendly message
func translateError(error: Error, context: String) async throws -> ErrorTranslation {
    let prompt = """
    Translate this technical error to a user-friendly message:

    Error: \(error.localizedDescription)
    Context: \(context)

    Respond in JSON format:
    {
      "userFriendlyMessage": "...",
      "technicalDetails": "...",
      "suggestedAction": "...",
      "severity": "critical|high|medium|low"
    }
    """

    do {
        let response = try await sendMessage(prompt, context: nil)
        // Parse JSON response
        if let data = response.data(using: .utf8),
           let json = try? JSONSerialization.jsonObject(with: data) as? [String: String] {
            return ErrorTranslation(
                userFriendlyMessage: json["userFriendlyMessage"] ?? error.localizedDescrip
                technicalDetails: json["technicalDetails"] ?? "",
                suggestedAction: json["suggestedAction"] ?? "Contact support",
                severity: json["severity"] ?? "medium"
            )
        }
        return fallbackErrorTranslation(error: error)
    } catch {
        return fallbackErrorTranslation(error: error)
    }
}

/// Recommend recovery action for a deployment failure
func recommendRecovery(failure: String, snapshotAvailable: Bool) async throws -> RecoveryR
    let prompt = """
    Recommend a recovery action for this deployment failure:

    Failure: \(failure)
    Snapshot Available: \(snapshotAvailable)

    Respond in JSON format:
    {
      "action": "rollback|retry|manual",
      "confidence": 0.0-1.0,
      "reason": "...",
      "steps": ["step1", "step2"],
      "alternativeActions": ["alt1", "alt2"]
    }
    """

    do {
        let response = try await sendMessage(prompt, context: nil)
        if let data = response.data(using: .utf8),
           let json = try? JSONSerialization.jsonObject(with: data) as? [String: Any] {
            return RecoveryRecommendation(
                action: json["action"] as? String ?? (snapshotAvailable ? "rollback" : "ma
                confidence: json["confidence"] as? Double ?? 0.5,
                reason: json["reason"] as? String ?? "Based on failure analysis",
                steps: json["steps"] as? [String] ?? [],
                alternativeActions: json["alternativeActions"] as? [String] ?? []
            )
        }
        return fallbackRecoveryRecommendation(snapshotAvailable: snapshotAvailable)
    } catch {
        return fallbackRecoveryRecommendation(snapshotAvailable: snapshotAvailable)
    }
}

/// Generate release notes from conventional commits
func generateReleaseNotes(commits: [String]) async throws -> GeneratedReleaseNotes {
    let commitList = commits.joined(separator: "\n")
    let prompt = """
    Generate release notes from these commits:

    \(commitList)
    Respond in JSON format:
    {
      "summary": "...",
      "features": ["..."],
      "fixes": ["..."],
      "breakingChanges": ["..."],
      "upgradeNotes": "..."
    }
    """

    do {
        let response = try await sendMessage(prompt, context: nil)
        if let data = response.data(using: .utf8),
           let json = try? JSONSerialization.jsonObject(with: data) as? [String: Any] {
            return GeneratedReleaseNotes(
                summary: json["summary"] as? String ?? "Release notes",
                features: json["features"] as? [String] ?? [],
                fixes: json["fixes"] as? [String] ?? [],
                breakingChanges: json["breakingChanges"] as? [String] ?? [],
                upgradeNotes: json["upgradeNotes"] as? String ?? ""
            )
        }
        return fallbackReleaseNotes(commits: commits)
    } catch {
        return fallbackReleaseNotes(commits: commits)
    }
}

/// Assess deployment risk and complexity
func assessDeployment(packageVersion: String, currentVersions: [String: String]) async thr
    let versionsJson = currentVersions.map { "\($0.key): \($0.value)" }.joined(separator: '
    let prompt = """
    Assess this deployment:

    Target Package: \(packageVersion)
    Current Versions: \(versionsJson)

    Respond in JSON format:
    {
      "riskLevel": "low|medium|high|critical",
      "migrationComplexity": "simple|moderate|complex",
      "estimatedDuration": "...",
      "warnings": ["..."],
      "recommendations": ["..."]
    }
    """

    do {
```

```swift
            let response = try await sendMessage(prompt, context: nil)
            if let data = response.data(using: .utf8),
               let json = try? JSONSerialization.jsonObject(with: data) as? [String: Any] {
                return AIAssessment(
                    riskLevel: json["riskLevel"] as? String ?? "medium",
                    migrationComplexity: json["migrationComplexity"] as? String ?? "moderate",
                    estimatedDuration: json["estimatedDuration"] as? String ?? "15-30 minutes"
                    warnings: json["warnings"] as? [String] ?? [],
                    recommendations: json["recommendations"] as? [String] ?? []
                )
            }
            return fallbackAssessment()
        } catch {
            return fallbackAssessment()
        }
    }

    // MARK: - Fallback Methods (when AI unavailable)

    func fallbackExplanation(for event: String) -> String {
        switch event.lowercased() {
        case let e where e.contains("snapshot"):
            return "Creating a backup of current system state before making changes."
        case let e where e.contains("migration"):
            return "Applying database schema changes to update the system."
        case let e where e.contains("health"):
            return "Verifying all services are responding correctly after deployment."
        case let e where e.contains("maintenance"):
            return "Temporarily pausing user requests to safely apply updates."
        case let e where e.contains("rollback"):
            return "Reverting to the previous system state due to an issue."
        default:
            return "Processing deployment step: \(event)"
        }
    }

    func fallbackErrorTranslation(error: Error) -> ErrorTranslation {
        let message = error.localizedDescription

        if message.contains("timeout") {
            return ErrorTranslation(
                userFriendlyMessage: "The operation took too long to complete.",
                technicalDetails: message,
                suggestedAction: "Check network connectivity and try again.",
                severity: "medium"
            )
        } else if message.contains("permission") || message.contains("access") {
            return ErrorTranslation(
                userFriendlyMessage: "Insufficient permissions to complete this action.",
                technicalDetails: message,
                suggestedAction: "Verify AWS credentials and IAM permissions.",
                severity: "high"
            )
        } else if message.contains("connection") || message.contains("network") {
            return ErrorTranslation(
                userFriendlyMessage: "Unable to connect to AWS services.",
                technicalDetails: message,
                suggestedAction: "Check your internet connection and AWS region settings.",
                severity: "high"
            )
        }

        return ErrorTranslation(
            userFriendlyMessage: "An error occurred during the operation.",
            technicalDetails: message,
            suggestedAction: "Review the error details and try again.",
            severity: "medium"
        )
    }

    func fallbackRecoveryRecommendation(snapshotAvailable: Bool) -> RecoveryRecommendation {
        if snapshotAvailable {
            return RecoveryRecommendation(
                action: "rollback",
                confidence: 0.8,
                reason: "A snapshot is available for safe rollback",
                steps: ["Initiate rollback", "Restore from snapshot", "Verify system health"],
                alternativeActions: ["Retry deployment", "Manual intervention"]
            )
        }
        return RecoveryRecommendation(
            action: "manual",
            confidence: 0.5,
            reason: "No snapshot available - manual intervention recommended",
            steps: ["Review error logs", "Assess system state", "Contact support if needed"],
            alternativeActions: ["Retry deployment with fixes"]
        )
    }

    func fallbackReleaseNotes(commits: [String]) -> GeneratedReleaseNotes {
        var features: [String] = []
        var fixes: [String] = []
        var breaking: [String] = []

        for commit in commits {
            if commit.starts(with: "feat!") || commit.contains("!:") {
                breaking.append(commit)
            } else if commit.starts(with: "feat") {
                features.append(commit)
            } else if commit.starts(with: "fix") {
                fixes.append(commit)
            }
        }

        return GeneratedReleaseNotes(
            summary: "Release containing \(commits.count) changes",
            features: features,
            fixes: fixes,
            breakingChanges: breaking,
            upgradeNotes: breaking.isEmpty ? "" : "This release contains breaking changes. Rev
        )
    }

    func fallbackAssessment() -> AIAssessment {
        return AIAssessment(
            riskLevel: "medium",
            migrationComplexity: "moderate",
            estimatedDuration: "15-30 minutes",
            warnings: ["AI assessment unavailable - using default values"],
            recommendations: ["Create a snapshot before proceeding", "Monitor deployment close
        )
    }
}

// MARK: - Singleton Access

extension AIAssistantService {
    static let shared = AIAssistantService()
}
```

File: Models/ManagedApp.swift

```
import Foundation

let DOMAIN_PLACEHOLDER = "YOUR_DOMAIN.com"

struct ManagedApp: Identifiable, Codable, Hashable, Sendable {
    let id: String
    var name: String
    var domain: String
    var description: String?
    var createdAt: Date
    var updatedAt: Date
    var environments: EnvironmentStatuses

    var isDomainConfigured: Bool {
        !domain.contains(DOMAIN_PLACEHOLDER)
    }

    struct EnvironmentStatuses: Codable, Hashable, Sendable {
        var dev: EnvironmentStatus
        var staging: EnvironmentStatus
        var prod: EnvironmentStatus

        subscript(env: DeployEnvironment) -> EnvironmentStatus {
            get {
                switch env {
                case .dev: return dev
                case .staging: return staging
                case .prod: return prod
                }
            }
            set {
                switch env {
                case .dev: dev = newValue
                case .staging: staging = newValue
                case .prod: prod = newValue
                }
            }
        }
    }
}

struct EnvironmentStatus: Codable, Hashable, Sendable {
    var deployed: Bool
    var version: String?
    var tier: Int
    var lastDeployedAt: Date?
    var healthStatus: HealthStatus
    var apiUrl: String?
    var dashboardUrl: String?
}

enum HealthStatus: String, Codable, Sendable {
    case healthy, degraded, unhealthy, unknown

    var color: String {
        switch self {
        case .healthy: return "green"
        case .degraded: return "orange"
        case .unhealthy: return "red"
        case .unknown: return "gray"
        }
    }
}

extension ManagedApp {
    static let defaults: [ManagedApp] = [
        ManagedApp(
            id: "thinktank",
            name: "Think Tank",
            domain: "thinktank.\(DOMAIN_PLACEHOLDER)",
            description: "AI-powered brainstorming and ideation platform",
            createdAt: Date(),
            updatedAt: Date(),
            environments: .init(
                dev: .init(deployed: false, tier: 1, healthStatus: .unknown),
                staging: .init(deployed: false, tier: 2, healthStatus: .unknown),
                prod: .init(deployed: false, tier: 3, healthStatus: .unknown)
            )
        ),
        ManagedApp(
            id: "launchboard",
            name: "Launch Board",
            domain: "launchboard.\(DOMAIN_PLACEHOLDER)",
            description: "Project launch management and tracking",
            createdAt: Date(),
            updatedAt: Date(),
            environments: .init(
                dev: .init(deployed: false, tier: 1, healthStatus: .unknown),
                staging: .init(deployed: false, tier: 2, healthStatus: .unknown),
                prod: .init(deployed: false, tier: 3, healthStatus: .unknown)
            )
        ),
        ManagedApp(
            id: "alwaysme",
            name: "Always Me",
            domain: "alwaysme.\(DOMAIN_PLACEHOLDER)",
            description: "Personal AI assistant and memory",
            createdAt: Date(),
            updatedAt: Date(),
            environments: .init(
                dev: .init(deployed: false, tier: 1, healthStatus: .unknown),
                staging: .init(deployed: false, tier: 2, healthStatus: .unknown),
                prod: .init(deployed: false, tier: 3, healthStatus: .unknown)
            )
        ),
        ManagedApp(
```

```swift
            id: "mechanicalmaker",
            name: "Mechanical Maker",
            domain: "mechanicalmaker.\(DOMAIN_PLACEHOLDER)",
            description: "AI-assisted mechanical design and CAD",
            createdAt: Date(),
            updatedAt: Date(),
            environments: .init(
                dev: .init(deployed: false, tier: 1, healthStatus: .unknown),
                staging: .init(deployed: false, tier: 2, healthStatus: .unknown),
                prod: .init(deployed: false, tier: 3, healthStatus: .unknown)
            )
        )
    ]
}
```

File: Models/Configuration.swift

```swift
import Foundation

/// Centralized configuration for RADIANT Deployer
/// All hardcoded values should be placed here for easy customization
struct Configuration {

    // MARK: - AWS Regions

    /// Default region for releases bucket
    static var releasesBucketRegion: String {
        ProcessInfo.processInfo.environment["RADIANT_RELEASES_REGION"] ?? "us-east-1"
    }

    /// Primary deployment region
    static var primaryRegion: String {
        ProcessInfo.processInfo.environment["RADIANT_PRIMARY_REGION"] ?? "us-east-1"
    }

    /// Available regions for deployment
    static let availableRegions: [String] = [
        "us-east-1",
        "us-west-2",
        "eu-west-1",
        "eu-central-1",
        "ap-northeast-1",
        "ap-southeast-1",
        "ap-south-1"
    ]
```

```swift
// MARK: - S3 Buckets

/// Releases bucket name pattern
static func releasesBucket(region: String = releasesBucketRegion) -> String {
    "radiant-releases-\(region)"
}

/// Media bucket name pattern
static func mediaBucket(region: String) -> String {
    "radiant-media-\(region)"
}

// MARK: - SES Configuration

/// SES mail-from MX record
static func sesMailFromMX(region: String = primaryRegion) -> String {
    "10 feedback-smtp.\(region).amazonses.com"
}

/// SES SPF record
static let sesSPFRecord = "v=spf1 include:amazonses.com ~all"

// MARK: - Version Information

/// Current RADIANT version
static let radiantVersion = "4.18.0"

/// Minimum deployer version required
static let minimumDeployerVersion = "4.18.0"

// MARK: - Timeouts (seconds)

static let cdkDeployTimeout = 1800
static let cdkBootstrapTimeout = 600
static let healthCheckTimeout = 30
static let migrationTimeout = 300
static let packageDownloadTimeout = 300

// MARK: - Defaults

/// Default VPC CIDR
static let defaultVpcCidr = "10.0.0.0/16"

/// Default Aurora instance class by tier
static func defaultAuroraInstance(tier: Int) -> String {
    switch tier {
    case 1: return "db.t3.medium"
    case 2: return "db.r6g.large"
```

```
        case 3: return "db.r6g.xlarge"
        case 4: return "db.r6g.2xlarge"
        default: return "db.t3.medium"
        }
    }

    // MARK: - Compatibility

    /// Supported deployment tiers
    static let supportedTiers = ["1", "2", "3", "4"]

    /// AWS CDK version requirement
    static let awsCdkVersion = "2.x"

    /// Node.js version requirement
    static let nodejsVersion = "20.x"

    /// PostgreSQL version requirement
    static let postgresqlVersion = "15"
}
```

File: Models/Credentials.swift

```
import Foundation

struct CredentialSet: Identifiable, Codable {
    let id: String
    var name: String
    var accessKeyId: String
    var secretAccessKey: String
    var region: String
    var accountId: String?
    var environment: CredentialEnvironment
    var createdAt: Date
    var lastValidatedAt: Date?
    var isValid: Bool?

    var maskedSecretKey: String {
        guard secretAccessKey.count > 8 else { return "********" }
        let prefix = String(secretAccessKey.prefix(4))
        let suffix = String(secretAccessKey.suffix(4))
        return "\(prefix)...\(suffix)"
    }
}

enum CredentialEnvironment: String, Codable, CaseIterable {
    case dev = "Development"
    case staging = "Staging"
    case prod = "Production"
    case shared = "Shared"
}

struct AWSAccount: Codable {
    let accountId: String
    let accountAlias: String?
    let regions: [String]
}
```