Contents

RADIANT v4.18.0 - Swift Deployer App Source Export

Component: macOS Native Deployer Application **Language**: Swift 5.9, SwiftUI **Files**: 36 source files **Platform**: macOS 13.0+

Architecture Narrative

The Swift Deployer is a **native macOS application** that provides a graphical interface for deploying and managing RADIANT infrastructure on AWS. It follows Apple's Human Interface Guidelines and implements modern macOS patterns including NavigationSplitView, Liquid Glass effects, and full menu bar integration.

Key Capabilities

1. **One-Click Deployment** - Deploy complete RADIANT infrastructure with a single click
2. **AI-Powered Assistant** - Claude-powered assistant for guidance and troubleshooting
3. **Snapshot Management** - Create, restore, and manage deployment snapshots
4. **Multi-Region Support** - Deploy across multiple AWS regions
5. **1Password Integration** - Secure credential management
6. **Real-Time Monitoring** - Track deployment progress and health

Component Structure

```
Sources/RadiantDeployer/
   RadiantDeployerApp.swift      # App entry point
   AppState.swift                # Global application state
   Config/
       RadiantConfig.swift       # Configuration management
   Models/                       # Data models
       Configuration.swift
       Credentials.swift
       Deployment.swift
```

```
    DomainConfiguration.swift
    InstallationParameters.swift
    ManagedApp.swift
Services/                    # Business logic services
    AIAssistantService.swift
    AIRegistryService.swift
    APIService.swift
    AWSService.swift
    CDKService.swift
    CredentialService.swift
    DatabaseService.swift
    DeploymentService.swift
    DNSService.swift
    HealthCheckService.swift
    LocalStorageManager.swift
    MultiRegionService.swift
    OnePasswordService.swift
    PackageService.swift
    SeedDataService.swift
    SnapshotService.swift
    TimeoutService.swift
Views/                       # SwiftUI views
    MainView.swift
    SettingsView.swift
Components/                  # Reusable UI components
     AppCommands.swift
     DataTableComponents.swift
     DetailViewComponents.swift
     MacOSComponents.swift
```

Source Files

1. RadiantDeployerApp.swift

Purpose: Main application entry point. Defines the SwiftUI App structure with main window, settings, and menu commands.

```swift
import SwiftUI

@main
struct RadiantDeployerApp: App {
    @StateObject private var appState = AppState()

    var body: some Scene {
        WindowGroup("Radiant Deployer") {
            MainView()
                .environmentObject(appState)
```

```
                .frame(minWidth: 1200, minHeight: 800)
        }
        .windowStyle(.titleBar)
        .windowToolbarStyle(.unified(showsTitle: true))
        .commands {
            RadiantCommands(appState: appState)
        }

        Settings {
            SettingsView()
                .environmentObject(appState)
        }
    }
}

#Preview {
    MainView()
        .environmentObject(AppState())
        .frame(width: 1200, height: 800)
}
```

---

2. AppState.swift

Purpose: Global application state management using SwiftUI's `@Published` properties. Manages navigation, credentials, deployment state, and service references.

```
import SwiftUI
import Combine

@MainActor
final class AppState: ObservableObject {
    // MARK: - Navigation
    @Published var selectedTab: NavigationTab = .dashboard
    @Published var selectedApp: ManagedApp?
    @Published var selectedEnvironment: DeployEnvironment = .dev

    // MARK: - UI State
    @Published var showInspector: Bool = false
    @Published var showAIAssistant: Bool = false
    @Published var sidebarWidth: CGFloat = 240
    @Published var inspectorWidth: CGFloat = 280
    @Published var columnVisibility: NavigationSplitViewVisibility = .all

    // MARK: - Data
    @Published var apps: [ManagedApp] = []
    @Published var credentials: [CredentialSet] = []
    @Published var isLoading = false
```

```
@Published var error: AppError?

// MARK: - Deployment
@Published var isDeploying = false
@Published var deploymentProgress: DeploymentProgress?
@Published var deploymentLogs: [LogEntry] = []

// MARK: - Services
let credentialService = CredentialService()
let cdkService = CDKService()
let awsService = AWSService()
let aiRegistryService = AIRegistryService()

// MARK: - Radiant Connection
@Published var radiantBaseURL: String?
@Published var radiantAuthToken: String?
@Published var isConnectedToRadiant = false

// MARK: - 1Password Status
@Published var onePasswordConfigured = true
@Published var onePasswordStatus: CredentialService.OnePasswordStatus?

// MARK: - Debug/Testing
@Published var bypassOnePassword = true

// MARK: - Initialization
init() {
    Task {
        await loadInitialData()
    }
}

func loadInitialData() async {
    isLoading = true
    defer { isLoading = false }

    if bypassOnePassword {
        onePasswordConfigured = true
        apps = ManagedApp.defaults
        return
    }

    onePasswordStatus = await credentialService.checkOnePasswordStatus()
    onePasswordConfigured = onePasswordStatus?.installed == true && onePasswordStatus?.sig

    guard onePasswordConfigured else {
        apps = ManagedApp.defaults
        return
```

```
        }

        do {
            credentials = try await credentialService.loadCredentials()
            apps = try await loadApps()
        } catch {
            self.error = AppError(message: "Failed to load data", underlying: error)
        }
    }

    func refreshOnePasswordStatus() async {
        if bypassOnePassword {
            onePasswordConfigured = true
            return
        }

        onePasswordStatus = await credentialService.checkOnePasswordStatus()
        onePasswordConfigured = onePasswordStatus?.installed == true && onePasswordStatus?.sig

        if onePasswordConfigured {
            do {
                credentials = try await credentialService.loadCredentials()
            } catch {
                self.error = AppError(message: "Failed to load credentials", underlying: error
            }
        }
    }

    private func loadApps() async throws -> [ManagedApp] {
        return ManagedApp.defaults
    }

    // MARK: - Commands

    func refreshAllStatus() async {
        isLoading = true
        defer { isLoading = false }

        do {
            credentials = try await credentialService.loadCredentials()
            apps = try await loadApps()
        } catch {
            self.error = AppError(message: "Failed to refresh status", underlying: error)
        }
    }

    func runHealthCheck() async {
        isLoading = true
```

```
        defer { isLoading = false }

        guard let credential = credentials.first else {
            isConnectedToRadiant = false
            return
        }

        let credentialsValid = await AWSService.shared.checkCredentialsValid(credential)
        let apiHealthy = await AWSService.shared.checkAPIHealth(credential: credential)
        let dbHealthy = await AWSService.shared.checkDatabaseHealth(credential: credential)

        isConnectedToRadiant = credentialsValid && apiHealthy && dbHealthy
    }
}

// MARK: - Navigation
enum NavigationTab: String, CaseIterable, Identifiable, Sendable {
    case dashboard = "Dashboard"
    case apps = "Apps"
    case deploy = "Deploy"
    case instances = "Instances"
    case snapshots = "Snapshots"
    case packages = "Packages"
    case history = "History"
    case providers = "Providers"
    case models = "Models"
    case selfHosted = "Self-Hosted"
    case domains = "Domains"
    case email = "Email"
    case multiRegion = "Multi-Region"
    case abTesting = "A/B Testing"
    case security = "Security"
    case compliance = "Compliance"
    case costs = "Costs"
    case monitoring = "Monitoring"
    case settings = "Settings"

    var id: String { rawValue }

    var icon: String {
        switch self {
        case .dashboard: return "square.grid.2x2"
        case .apps: return "app.badge"
        case .deploy: return "arrow.up.circle"
        case .instances: return "server.rack"
        case .snapshots: return "clock.arrow.circlepath"
        case .packages: return "shippingbox"
        case .history: return "clock"
```

```
        case .providers: return "building.2"
        case .models: return "cpu"
        case .selfHosted: return "memorychip"
        case .domains: return "globe.americas"
        case .email: return "envelope"
        case .multiRegion: return "globe"
        case .abTesting: return "flask"
        case .security: return "shield.lefthalf.filled"
        case .compliance: return "checkmark.shield"
        case .costs: return "dollarsign.circle"
        case .monitoring: return "waveform.path.ecg.rectangle"
        case .settings: return "gearshape"
        }
    }

    var color: Color {
        switch self {
        case .dashboard: return .blue
        case .apps: return .purple
        case .deploy: return .green
        case .instances: return .orange
        case .snapshots: return .cyan
        case .packages: return .indigo
        case .history: return .brown
        case .providers: return .teal
        case .models: return .pink
        case .selfHosted: return .mint
        case .domains: return .cyan
        case .email: return .orange
        case .multiRegion: return .blue
        case .abTesting: return .purple
        case .security: return .red
        case .compliance: return .green
        case .costs: return .yellow
        case .monitoring: return .teal
        case .settings: return .gray
        }
    }

    static var mainTabs: [NavigationTab] { [.dashboard, .apps, .deploy] }
    static var operationTabs: [NavigationTab] { [.instances, .snapshots, .packages, .history] }
    static var aiTabs: [NavigationTab] { [.providers, .models, .selfHosted] }
    static var configTabs: [NavigationTab] { [.domains, .email] }
    static var advancedTabs: [NavigationTab] { [.multiRegion, .abTesting] }
    static var securityTabs: [NavigationTab] { [.security, .compliance] }
    static var systemTabs: [NavigationTab] { [.costs, .monitoring, .settings] }
}
```

```swift
// MARK: - DeployEnvironment
enum DeployEnvironment: String, CaseIterable, Identifiable, Sendable, Codable {
    case dev = "Development"
    case staging = "Staging"
    case prod = "Production"

    var id: String { rawValue }

    var shortName: String {
        switch self {
        case .dev: return "DEV"
        case .staging: return "STAGING"
        case .prod: return "PROD"
        }
    }

    var color: Color {
        switch self {
        case .dev: return .blue
        case .staging: return .orange
        case .prod: return .green
        }
    }
}

// MARK: - Error
struct AppError: Identifiable, Sendable {
    let id = UUID()
    let message: String
    let underlying: (any Error)?

    var localizedDescription: String {
        if let underlying = underlying {
            return "\(message): \(underlying.localizedDescription)"
        }
        return message
    }
}
```

---

3. Config/RadiantConfig.swift

Purpose: Centralized configuration with environment variable support. Manages AWS settings, paths, timeouts, and feature flags.

```swift
// RADIANT v4.18.0 - Configuration
// Centralized configuration with environment variable support
```

```swift
import Foundation
import os.log

// MARK: - Logger

enum RadiantLogger {
    private static let subsystem = "com.radiant.deployer"

    static let general = Logger(subsystem: subsystem, category: "general")
    static let deployment = Logger(subsystem: subsystem, category: "deployment")
    static let packages = Logger(subsystem: subsystem, category: "packages")
    static let seeds = Logger(subsystem: subsystem, category: "seeds")
    static let aws = Logger(subsystem: subsystem, category: "aws")
    static let credentials = Logger(subsystem: subsystem, category: "credentials")

    static func error(_ message: String, error: Error? = nil, category: Logger = general) {
        if let error = error {
            category.error("\(message): \(error.localizedDescription)")
        } else {
            category.error("\(message)")
        }
    }

    static func warning(_ message: String, category: Logger = general) {
        category.warning("\(message)")
    }

    static func info(_ message: String, category: Logger = general) {
        category.info("\(message)")
    }

    static func debug(_ message: String, category: Logger = general) {
        category.debug("\(message)")
    }
}

// MARK: - Configuration

struct RadiantConfig: Sendable {

    static let shared = RadiantConfig()

    // MARK: - AWS Configuration
    let releasesBucket: String
    let seedsPrefix: String
    let packagesPrefix: String
    let defaultRegion: String
```

```
    // MARK: - Paths
    let packageCacheDirectory: URL
    let seedsCacheDirectory: URL
    let awsCliPath: String
    let cdkCliPath: String

    // MARK: - Timeouts
    let networkTimeout: TimeInterval
    let cdkDeploymentTimeout: TimeInterval
    let healthCheckTimeout: TimeInterval

    // MARK: - Feature Flags
    let verboseLogging: Bool
    let dryRunMode: Bool

    private init() {
        let env = ProcessInfo.processInfo.environment

        self.releasesBucket = env["RADIANT_RELEASES_BUCKET"] ?? "radiant-releases-us-east-1"
        self.seedsPrefix = env["RADIANT_SEEDS_PREFIX"] ?? "seeds/"
        self.packagesPrefix = env["RADIANT_PACKAGES_PREFIX"] ?? "packages/"
        self.defaultRegion = env["AWS_DEFAULT_REGION"] ?? "us-east-1"

        let appSupport: URL
        if let appSupportDir = FileManager.default.urls(for: .applicationSupportDirectory, in:
            appSupport = appSupportDir
        } else {
            appSupport = FileManager.default.homeDirectoryForCurrentUser.appendingPathComponent
        }
        let radiantDir = appSupport.appendingPathComponent("RadiantDeployer")

        self.packageCacheDirectory = radiantDir.appendingPathComponent("packages")
        self.seedsCacheDirectory = radiantDir.appendingPathComponent("seeds")
        self.awsCliPath = env["AWS_CLI_PATH"] ?? "/usr/local/bin/aws"
        self.cdkCliPath = env["CDK_CLI_PATH"] ?? "/usr/local/bin/cdk"

        self.networkTimeout = TimeInterval(env["RADIANT_NETWORK_TIMEOUT"] ?? "30") ?? 30
        self.cdkDeploymentTimeout = TimeInterval(env["RADIANT_CDK_TIMEOUT"] ?? "3600") ?? 3600
        self.healthCheckTimeout = TimeInterval(env["RADIANT_HEALTH_TIMEOUT"] ?? "60") ?? 60

        self.verboseLogging = env["RADIANT_VERBOSE"] == "true"
        self.dryRunMode = env["RADIANT_DRY_RUN"] == "true"

        try? FileManager.default.createDirectory(at: packageCacheDirectory, withIntermediateDi
        try? FileManager.default.createDirectory(at: seedsCacheDirectory, withIntermediateDire
    }

    func releasesS3URI(path: String = "") -> String {
```

```
        "s3://\(releasesBucket)/\(path)"
    }

    func instanceBucket(appId: String, environment: String) -> String {
        "radiant-\(appId)-\(environment)-deployments"
    }
}

// MARK: - Credential Sanitization

extension RadiantConfig {

    static func sanitize(_ string: String) -> String {
        var result = string

        let patterns = [
            "A[KBS]IA[A-Z0-9]{16}",
            "[A-Za-z0-9/+=]{40}",
            "(?i)(api[_-]?key|apikey|secret|password|token)[\"':\\s=]+[A-Za-z0-9/+=_-]{16,}",
        ]

        for pattern in patterns {
            if let regex = try? NSRegularExpression(pattern: pattern, options: []) {
                result = regex.stringByReplacingMatches(
                    in: result,
                    range: NSRange(result.startIndex..., in: result),
                    withTemplate: "[REDACTED]"
                )
            }
        }

        return result
    }

    static func sanitizeError(_ error: Error) -> String {
        sanitize(error.localizedDescription)
    }
}
```

---

4. Models/Deployment.swift

Purpose: Deployment-related data models including progress tracking, results, and granular deployment state machine.

```
import Foundation

let RADIANT_VERSION = "4.18.0"
```

```swift
struct DeploymentProgress: Identifiable, Sendable {
    let id = UUID()
    var phase: DeploymentPhase
    var progress: Double
    var currentStack: String?
    var message: String?
    var startedAt: Date
    var estimatedCompletion: Date?
}

enum DeploymentPhase: String, CaseIterable, Sendable {
    case idle = "Idle"
    case validating = "Validating Credentials"
    case bootstrapping = "Bootstrapping CDK"
    case synthesizing = "Synthesizing Stacks"
    case deployingFoundation = "Deploying Foundation"
    case deployingNetworking = "Deploying Networking"
    case deploySecurity = "Deploying Security"
    case deployingData = "Deploying Data Layer"
    case deployingAI = "Deploying AI Services"
    case deployingAPI = "Deploying API Layer"
    case deployingAdmin = "Deploying Admin Dashboard"
    case runningMigrations = "Running Migrations"
    case seedingData = "Seeding Initial Data"
    case verifying = "Verifying Deployment"
    case complete = "Complete"
    case failed = "Failed"

    var progress: Double {
        switch self {
        case .idle: return 0.0
        case .validating: return 0.05
        case .bootstrapping: return 0.10
        case .synthesizing: return 0.15
        case .deployingFoundation: return 0.25
        case .deployingNetworking: return 0.35
        case .deploySecurity: return 0.45
        case .deployingData: return 0.55
        case .deployingAI: return 0.65
        case .deployingAPI: return 0.75
        case .deployingAdmin: return 0.85
        case .runningMigrations: return 0.90
        case .seedingData: return 0.95
        case .verifying: return 0.98
        case .complete: return 1.0
        case .failed: return 0.0
        }
```

```
    }

    var icon: String {
        switch self {
        case .idle: return "circle"
        case .validating: return "checkmark.shield"
        case .bootstrapping: return "arrow.up.circle"
        case .synthesizing: return "doc.text"
        case .deployingFoundation: return "building"
        case .deployingNetworking: return "network"
        case .deploySecurity: return "lock.shield"
        case .deployingData: return "cylinder"
        case .deployingAI: return "cpu"
        case .deployingAPI: return "server.rack"
        case .deployingAdmin: return "rectangle.3.group"
        case .runningMigrations: return "arrow.triangle.2.circlepath"
        case .seedingData: return "leaf"
        case .verifying: return "checkmark.circle"
        case .complete: return "checkmark.circle.fill"
        case .failed: return "xmark.circle.fill"
        }
    }
}

struct DeploymentResult: Identifiable, Codable, Sendable {
    let id: String
    let appId: String
    let environment: String
    let version: String
    let success: Bool
    let startedAt: Date
    let completedAt: Date
    let outputs: DeploymentOutputs?
    let errors: [String]?

    static func create(
        appId: String,
        environment: String,
        success: Bool,
        startedAt: Date,
        outputs: DeploymentOutputs? = nil,
        errors: [String]? = nil
    ) -> DeploymentResult {
        DeploymentResult(
            id: UUID().uuidString,
            appId: appId,
            environment: environment,
            version: RADIANT_VERSION,
```

```
            success: success,
            startedAt: startedAt,
            completedAt: Date(),
            outputs: outputs,
            errors: errors
        )
    }
}

struct DeploymentOutputs: Codable, Sendable {
    let apiUrl: String
    let graphqlUrl: String
    let dashboardUrl: String
    let cognitoUserPoolId: String
    let cognitoClientId: String
    let cognitoDomain: String
    let auroraEndpoint: String
    let s3MediaBucket: String
    let cloudfrontDistribution: String
}

struct LogEntry: Identifiable, Sendable {
    let id = UUID()
    let timestamp: Date
    let level: LogLevel
    let message: String
    let metadata: [String: String]?
}

enum LogLevel: String, Sendable, Codable {
    case debug, info, warn, error, success

    var color: String {
        switch self {
        case .debug: return "gray"
        case .info: return "blue"
        case .warn: return "orange"
        case .error: return "red"
        case .success: return "green"
        }
    }

    var icon: String {
        switch self {
        case .debug: return "ant"
        case .info: return "info.circle"
        case .warn: return "exclamationmark.triangle"
        case .error: return "xmark.circle"
```

```
        case .success: return "checkmark.circle"
        }
    }
}

// MARK: - Granular Deployment State

enum PreparationStep: String, Sendable {
    case validatingPackage = "Validating Package"
    case checkingCompatibility = "Checking Compatibility"
    case acquiringLock = "Acquiring Deployment Lock"
    case loadingConfiguration = "Loading Configuration"
}

struct HealthCheckResult: Sendable, Identifiable {
    let id = UUID()
    let service: String
    let status: HealthStatus
    let responseTime: TimeInterval?
    let message: String?

    enum HealthStatus: String, Sendable {
        case healthy = "Healthy"
        case unhealthy = "Unhealthy"
        case timeout = "Timeout"
        case pending = "Pending"
    }
}

struct DeploymentFailure: Sendable {
    let phase: String
    let error: String
    let technicalDetails: String?
    let isRetryable: Bool
    let timestamp: Date

    init(phase: String, error: Error, isRetryable: Bool = false) {
        self.phase = phase
        self.error = error.localizedDescription
        self.technicalDetails = String(describing: error)
        self.isRetryable = isRetryable
        self.timestamp = Date()
    }
}

struct RollbackResult: Sendable {
    let success: Bool
    let snapshotId: String?
```

```
    let restoredVersion: String?
    let duration: TimeInterval
    let message: String
}

struct RollbackFailure: Sendable {
    let error: String
    let partiallyRestored: Bool
    let recoverySteps: [String]
}

enum DeploymentState: Sendable {
    case idle
    case preparing(PreparationStep)
    case creatingSnapshot(progress: Double)
    case enablingMaintenance
    case deployingInfrastructure(progress: Double, message: String)
    case runningMigrations(current: Int, total: Int, stepName: String)
    case deployingLambda(progress: Double)
    case deployingDashboard(progress: Double)
    case runningHealthChecks(results: [HealthCheckResult])
    case disablingMaintenance
    case verifying
    case complete(DeploymentResult)
    case failed(DeploymentFailure)
    case cancelling(fromState: String)
    case rollingBack(progress: Double, step: String)
    case rolledBack(RollbackResult)
    case rollbackFailed(RollbackFailure)

    var canCancel: Bool {
        switch self {
        case .idle, .complete, .failed, .cancelling, .rollingBack, .rolledBack, .rollbackFailed
            return false
        case .preparing, .creatingSnapshot, .enablingMaintenance, .deployingInfrastructure,
             .runningMigrations, .deployingLambda, .deployingDashboard, .runningHealthChecks,
             .disablingMaintenance, .verifying:
            return true
        }
    }

    var progress: Double {
        switch self {
        case .idle: return 0.0
        case .preparing: return 0.05
        case .creatingSnapshot(let p): return 0.05 + p * 0.10
        case .enablingMaintenance: return 0.15
        case .deployingInfrastructure(let p, _): return 0.15 + p * 0.40
```

```
        case .runningMigrations(let c, let t, _): return 0.55 + (Double(c) / Double(max(t, 1)))
        case .deployingLambda(let p): return 0.70 + p * 0.10
        case .deployingDashboard(let p): return 0.80 + p * 0.10
        case .runningHealthChecks: return 0.90
        case .disablingMaintenance: return 0.95
        case .verifying: return 0.98
        case .complete: return 1.0
        case .failed: return 0.0
        case .cancelling: return 0.0
        case .rollingBack(let p, _): return p
        case .rolledBack: return 1.0
        case .rollbackFailed: return 0.0
        }
    }

    var displayName: String {
        switch self {
        case .idle: return "Ready"
        case .preparing(let step): return step.rawValue
        case .creatingSnapshot: return "Creating Snapshot"
        case .enablingMaintenance: return "Enabling Maintenance Mode"
        case .deployingInfrastructure(_, let msg): return msg.isEmpty ? "Deploying Infrastructu
        case .runningMigrations(let c, let t, let name): return "Migration \(c)/\(t): \(name)"
        case .deployingLambda: return "Deploying Lambda Functions"
        case .deployingDashboard: return "Deploying Admin Dashboard"
        case .runningHealthChecks: return "Running Health Checks"
        case .disablingMaintenance: return "Disabling Maintenance Mode"
        case .verifying: return "Verifying Deployment"
        case .complete: return "Deployment Complete"
        case .failed(let f): return "Failed: \(f.phase)"
        case .cancelling(let from): return "Cancelling (\(from))"
        case .rollingBack(_, let step): return "Rolling Back: \(step)"
        case .rolledBack: return "Rolled Back Successfully"
        case .rollbackFailed: return "Rollback Failed"
        }
    }

    var icon: String {
        switch self {
        case .idle: return "circle"
        case .preparing: return "gearshape"
        case .creatingSnapshot: return "camera"
        case .enablingMaintenance: return "wrench.and.screwdriver"
        case .deployingInfrastructure: return "building.2"
        case .runningMigrations: return "arrow.triangle.2.circlepath"
        case .deployingLambda: return "function"
        case .deployingDashboard: return "rectangle.3.group"
        case .runningHealthChecks: return "heart.text.square"
```

```
        case .disablingMaintenance: return "checkmark.seal"
        case .verifying: return "checkmark.shield"
        case .complete: return "checkmark.circle.fill"
        case .failed: return "xmark.circle.fill"
        case .cancelling: return "stop.circle"
        case .rollingBack: return "arrow.uturn.backward.circle"
        case .rolledBack: return "arrow.uturn.backward.circle.fill"
        case .rollbackFailed: return "exclamationmark.triangle.fill"
        }
    }

    var isTerminal: Bool {
        switch self {
        case .complete, .failed, .rolledBack, .rollbackFailed:
            return true
        default:
            return false
        }
    }
}
```

---

5. Services/AIAssistantService.swift

Purpose: Claude API integration for intelligent deployment guidance. Provides error translation, recovery recommendations, and contextual explanations.

```
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
}

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,
    ]

    SecItemDelete(query as CFDictionary)

    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: - 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

    struct ErrorTranslation: Sendable {
        let userFriendlyMessage: String
        let technicalDetails: String
        let suggestedAction: String
        let severity: String
    }

    struct RecoveryRecommendation: Sendable {
        let action: String
        let confidence: Double
        let reason: String
        let steps: [String]
        let alternativeActions: [String]
    }

    struct AIAssessment: Sendable {
        let riskLevel: String
        let migrationComplexity: String
        let estimatedDuration: String
        let warnings: [String]
        let recommendations: [String]
    }

    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)
        }
    }
```

```
    func translateError(error: Error, context: String) async throws -> ErrorTranslation {
        // Implementation with fallback...
    }

    func recommendRecovery(failure: String, snapshotAvailable: Bool) async throws -> RecoveryR
        // Implementation with fallback...
    }

    func assessDeployment(packageVersion: String, currentVersions: [String: String]) async thr
        // Implementation with fallback...
    }

    // MARK: - Fallback Methods

    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)"
        }
    }
}

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

---

6. Services/DeploymentService.swift (excerpt)

Purpose: Core deployment orchestration. Handles mode detection (install/update/rollback), execution, snapshot management, and parameter merging.

```
// RADIANT v4.18.0 - Deployment Service
// Handles deployment mode detection, execution, and snapshot management

import Foundation
```

```swift
enum DeploymentError: Error, LocalizedError {
    case noCredentials
    case instanceNotFound
    case snapshotNotFound(String)
    case invalidSnapshot(String)
    case invalidConfiguration(String)
    case parameterValidationFailed(String)
    case infrastructureDeploymentFailed(String)
    case migrationFailed(String)
    case rollbackFailed(String)
    case networkError(Error)
    case packageNotFound(String)
    case integrityCheckFailed
    case verificationFailed(String)
    case commandFailed(String, Int)

    var errorDescription: String? {
        switch self {
        case .noCredentials: return "No AWS credentials configured"
        case .instanceNotFound: return "No existing Radiant instance found"
        case .snapshotNotFound(let id): return "Snapshot not found: \(id)"
        case .invalidSnapshot(let reason): return "Invalid snapshot: \(reason)"
        case .invalidConfiguration(let reason): return "Invalid configuration: \(reason)"
        case .parameterValidationFailed(let reason): return "Parameter validation failed: \(rea
        case .infrastructureDeploymentFailed(let reason): return "Infrastructure deployment fai
        case .migrationFailed(let reason): return "Database migration failed: \(reason)"
        case .rollbackFailed(let reason): return "Rollback failed: \(reason)"
        case .networkError(let error): return "Network error: \(error.localizedDescription)"
        case .packageNotFound(let version): return "Deployment package not found: \(version)"
        case .integrityCheckFailed: return "Package integrity check failed"
        case .verificationFailed(let reason): return "Verification failed: \(reason)"
        case .commandFailed(let command, let exitCode): return "Command '\(command)' failed wit
        }
    }
}

enum MigrationMode: Sendable {
    case fresh       // Run ALL migrations (install)
    case incremental // Run only new migrations (update)
}

actor DeploymentService {

    private let awsService: AWSService
    private let cdkService: CDKService
    private let credentialService: CredentialService
    private var packageService: PackageService?
```

```
init() {
    self.awsService = AWSService()
    self.cdkService = CDKService()
    self.credentialService = CredentialService()
}

// MARK: - Mode Detection

func determineDeploymentMode(
    app: ManagedApp,
    environment: DeployEnvironment,
    targetVersion: String? = nil
) async throws -> DeploymentMode {

    let instanceExists = try await checkInstanceExists(app: app, environment: environment)

    if !instanceExists {
        return .install
    }

    let currentVersion = try await fetchCurrentVersion(app: app, environment: environment)
    let target = targetVersion ?? RADIANT_VERSION

    if compareVersions(target, currentVersion) > 0 {
        return .update
    } else if compareVersions(target, currentVersion) < 0 {
        return .rollback
    } else {
        return .update
    }
}

// MARK: - Install Mode Execution

func executeInstall(
    app: ManagedApp,
    environment: DeployEnvironment,
    package: DeploymentPackage,
    credentials: CredentialSet,
    onProgress: @escaping @Sendable (DeploymentProgress) -> Void
) async throws -> DeploymentExecutionResult {
    let startTime = Date()

    // STEP 1: Use DEFAULT parameters
    let envStatus = app.environments[environment]
    let tier = TierLevel(rawValue: envStatus.tier) ?? .seed
    let parameters = InstallationParameters.defaults(
        appId: app.id,
```

```
        environment: environment,
        tier: tier
    )

    // STEP 2: Deploy infrastructure with defaults
    try await deployInfrastructure(
        app: app,
        environment: environment,
        package: package,
        parameters: parameters,
        credentials: credentials,
        onProgress: onProgress
    )

    // STEP 3: Run initial migrations (creates tables)
    try await runMigrations(
        package: package,
        mode: .fresh,
        fromVersion: nil,
        onProgress: onProgress
    )

    // STEP 4: SEED the AI Registry (ONLY on fresh install)
    try await seedAIRegistry(
        package: package,
        onProgress: onProgress
    )

    // STEP 5: Create initial admin user
    try await createInitialAdmin(
        app: app,
        environment: environment,
        onProgress: onProgress
    )

    // STEP 6: Verify deployment
    let outputs = try await verifyDeployment(app: app, environment: environment)

    return DeploymentExecutionResult(
        mode: .install,
        success: true,
        version: package.manifest.version,
        rollbackSnapshotId: nil,
        outputs: outputs,
        errors: nil,
        duration: Date().timeIntervalSince(startTime)
    )
}
```

```
// MARK: - Update Mode Execution

func executeUpdate(
    app: ManagedApp,
    environment: DeployEnvironment,
    package: DeploymentPackage,
    credentials: CredentialSet,
    userChanges: ParameterChanges?,
    onProgress: @escaping @Sendable (DeploymentProgress) -> Void
) async throws -> DeploymentExecutionResult {
    let startTime = Date()

    // STEP 1: READ current parameters FROM the running instance
    let currentParameters = try await fetchCurrentParameters(
        app: app,
        environment: environment,
        credentials: credentials
    )

    // STEP 2: Create snapshot for rollback capability
    let snapshotId = try await createPreUpdateSnapshot(
        app: app,
        environment: environment,
        currentParameters: currentParameters,
        package: package
    )

    // STEP 3: MERGE user changes with current parameters
    let updatedParameters = mergeParameters(
        current: currentParameters,
        changes: userChanges
    )

    // STEP 4: Validate parameter changes are safe
    try validateParameterChanges(
        from: currentParameters,
        to: updatedParameters,
        package: package
    )

    // STEP 5: Deploy infrastructure with MERGED parameters
    try await deployInfrastructure(
        app: app,
        environment: environment,
        package: package,
        parameters: updatedParameters.toInstallationParameters(),
        credentials: credentials,
```

```
            onProgress: onProgress
        )

        // STEP 6: Run INCREMENTAL migrations (NOT fresh)
        try await runMigrations(
            package: package,
            mode: .incremental,
            fromVersion: currentParameters.version,
            onProgress: onProgress
        )

        // STEP 7: DO NOT seed AI Registry (preserve admin customizations)

        // STEP 8: Verify deployment
        let outputs = try await verifyDeployment(app: app, environment: environment)

        return DeploymentExecutionResult(
            mode: .update,
            success: true,
            version: package.manifest.version,
            rollbackSnapshotId: snapshotId,
            outputs: outputs,
            errors: nil,
            duration: Date().timeIntervalSince(startTime)
        )
    }

    // MARK: - Rollback Mode Execution

    func executeRollback(
        app: ManagedApp,
        environment: DeployEnvironment,
        targetSnapshotId: String,
        credentials: CredentialSet,
        onProgress: @escaping @Sendable (DeploymentProgress) -> Void
    ) async throws -> DeploymentExecutionResult {
        // Load snapshot, validate, create safety snapshot, deploy with snapshot params
        // ...
    }

    // MARK: - Validation

    private func validateParameterChanges(
        from current: InstanceParameters,
        to updated: InstanceParameters,
        package: DeploymentPackage
    ) throws {
        // Cannot change region after install
```

```
        if current.region != updated.region {
            throw DeploymentError.parameterValidationFailed(
                "Region cannot be changed after initial installation."
            )
        }

        // Cannot downgrade tier below what features require
        if updated.enableSelfHostedModels && updated.tier < .growth {
            throw DeploymentError.parameterValidationFailed(
                "Self-hosted models require GROWTH tier or higher"
            )
        }

        if updated.enableMultiRegion && updated.tier < .scale {
            throw DeploymentError.parameterValidationFailed(
                "Multi-region requires SCALE tier or higher"
            )
        }

        // Validate Aurora capacity
        if updated.auroraMinCapacity > updated.auroraMaxCapacity {
            throw DeploymentError.parameterValidationFailed(
                "Aurora min capacity cannot exceed max capacity"
            )
        }
    }
}
```

Additional Files (Summary)

Due to the size of this export, the remaining Swift files are summarized below:

File	Lines	Purpose
`AWSService.swift`	~400	AWS SDK wrapper for EC2, S3, RDS, CloudFormation
`CDKService.swift`	~300	CDK CLI wrapper for synth/deploy/destroy
`CredentialService.swift`	~250	1Password and AWS credential management
`PackageService.swift`	~350	Deployment package download and verification
`SnapshotService.swift`	~200	Snapshot creation and restoration
`HealthCheckService.swift`	~150	Service health verification
`LocalStorageManager.swift`	~200	SQLCipher encrypted local storage
`TimeoutService.swift`	~100	Operation timeout management
`MultiRegionService.swift`	~250	Multi-region deployment coordination

File	Lines	Purpose
SeedDataService.swift	~300	AI Registry seeding
DNSService.swift	~150	Route 53 DNS management
DatabaseService.swift	~200	Aurora PostgreSQL operations
AIRegistryService.swift	~400	AI model registry management
Configuration.swift	~150	Configuration model
Credentials.swift	~100	Credential models
InstallationParameters.swift	~200	Installation parameter models
ManagedApp.swift	~150	Managed app model
DomainConfiguration.swift	~100	Domain configuration
MacOSComponents.swift	~300	Design tokens and components
AppCommands.swift	~150	Menu bar commands
DataTableComponents.swift	~200	Table/list components
DetailViewComponents.swift	~150	Detail view patterns

This concludes the Swift Deployer App source export. See other export files for remaining components.