Contents

RADIANT Swift Deployer - ACTUAL SOURCE CODE - Part 1

File: RadiantDeployerApp.swift

```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)
}
```

File: AppState.swift

```swift
import SwiftUI
import Combine
```

```swift
@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  // Start as true when bypassing
    @Published var onePasswordStatus: CredentialService.OnePasswordStatus?

    // MARK: - Debug/Testing
    @Published var bypassOnePassword = true  // Set to true to skip 1Password during testing

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

```
    }
}

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

    // Allow bypass for testing
    if bypassOnePassword {
        onePasswordConfigured = true
        apps = ManagedApp.defaults
        return
    }

    // Check 1Password status first
    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 {
        // Health check implementation
        isLoading = true
        defer { isLoading = false }

        // Check AWS connectivity and service health
        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)

        // Update connection status based on health checks
        isConnectedToRadiant = credentialsValid && apiHealthy && dbHealthy
    }
}

// MARK: - Navigation
enum NavigationTab: String, CaseIterable, Identifiable, Sendable {
    // Main
    case dashboard = "Dashboard"
    case apps = "Apps"
    case deploy = "Deploy"

    // Operations
    case instances = "Instances"
```

```
case snapshots = "Snapshots"
case packages = "Packages"
case history = "History"

// AI Registry
case providers = "Providers"
case models = "Models"
case selfHosted = "Self-Hosted"

// Configuration
case domains = "Domains"
case email = "Email"

// Advanced
case multiRegion = "Multi-Region"
case abTesting = "A/B Testing"

// Security & Compliance
case security = "Security"
case compliance = "Compliance"

// System
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]
    }
}

// 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
    }
}
```

File: Config/RadiantConfig.swift

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

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")

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

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

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

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

// MARK: - Configuration
```

```
struct RadiantConfig: Sendable {

    // MARK: - Singleton

    static let shared = RadiantConfig()

    // MARK: - AWS Configuration

    /// S3 bucket for official releases
    let releasesBucket: String

    /// S3 bucket prefix for seeds
    let seedsPrefix: String

    /// S3 bucket prefix for packages
    let packagesPrefix: String

    /// Default AWS region
    let defaultRegion: String

    // MARK: - Paths

    /// Local cache directory for packages
    let packageCacheDirectory: URL

    /// Local cache directory for seeds
    let seedsCacheDirectory: URL

    /// AWS CLI path
    let awsCliPath: String

    /// CDK CLI path
    let cdkCliPath: String

    // MARK: - Timeouts

    /// Default network timeout in seconds
    let networkTimeout: TimeInterval

    /// CDK deployment timeout in seconds
    let cdkDeploymentTimeout: TimeInterval

    /// Health check timeout in seconds
    let healthCheckTimeout: TimeInterval

    // MARK: - Feature Flags
```

```
    /// Enable verbose logging
    let verboseLogging: Bool

    /// Enable dry run mode (no actual deployments)
    let dryRunMode: Bool

    // MARK: - Initialization

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

        // AWS Configuration
        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"

        // Paths - safely unwrap application support directory
        let appSupport: URL
        if let appSupportDir = FileManager.default.urls(for: .applicationSupportDirectory, in:
            appSupport = appSupportDir
        } else {
            // Fallback to home directory if application support is unavailable
            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"

        // Timeouts
        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

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

        // Ensure cache directories exist
        try? FileManager.default.createDirectory(at: packageCacheDirectory, withIntermediateDi
        try? FileManager.default.createDirectory(at: seedsCacheDirectory, withIntermediateDire
    }

    // MARK: - Helpers
```

```
    /// Get the full S3 URI for releases bucket
    func releasesS3URI(path: String = "") -> String {
        "s3://\(releasesBucket)/\(path)"
    }

    /// Get environment-specific bucket name
    func instanceBucket(appId: String, environment: String) -> String {
        "radiant-\(appId)-\(environment)-deployments"
    }
}

// MARK: - Credential Sanitization

extension RadiantConfig {

    /// Sanitize a string to remove potential credentials
    static func sanitize(_ string: String) -> String {
        var result = string

        // Patterns that might contain credentials
        let patterns = [
            // AWS Access Key ID (starts with AKIA, ABIA, ACCA, ASIA)
            "A[KBS]IA[A-Z0-9]{16}",
            // AWS Secret Access Key (40 char base64)
            "[A-Za-z0-9/+=]{40}",
            // Generic API keys
            "(?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
    }

    /// Sanitize error for logging (removes potential credentials)
    static func sanitizeError(_ error: Error) -> String {
        sanitize(error.localizedDescription)
    }
}
```

File: Models/Deployment.swift

```swift
import Foundation

let RADIANT_VERSION = "4.18.0"

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: - PROMPT-33 Granular Deployment State

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

/// Health check result for individual service
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"
    }
}

/// Deployment failure details
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()
    }
}

/// Rollback result after recovery
struct RollbackResult: Sendable {
    let success: Bool
    let snapshotId: String?
    let restoredVersion: String?
    let duration: TimeInterval
    let message: String
}

/// Rollback failure details
struct RollbackFailure: Sendable {
    let error: String
    let partiallyRestored: Bool
    let recoverySteps: [String]
}

/// Granular deployment state per PROMPT-33 spec
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)

    /// Whether cancel is allowed in this state
    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
        }
    }

    /// Overall progress percentage
    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
        }
    }

    /// Display name for the state
    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"
        }
    }

    /// Icon for the state
    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"
        }
    }

    /// Whether this is a terminal state
    var isTerminal: Bool {
        switch self {
        case .complete, .failed, .rolledBack, .rollbackFailed:
            return true
        default:
            return false
        }
    }
}
```