Contents

SECTION 1: FOUNDATION & SWIFT DEPLOYMENT APP (v2.0.0)

â• â

> **Dependencies:** Section 0 (Shared Types) **Creates:** Monorepo structure, Swift macOS deployment app

1.1 OVERVIEW

This section creates: 1. **Monorepo foundation** - Already partially in Section 0 2. **Swift macOS Deployment App** - Complete GUI for deploying RADIANT

The Swift app is the primary interface for administrators. Users should NEVER need to open a terminal.

1.2 CDK INFRASTRUCTURE PACKAGE SETUP

Since types are now in @radiant/shared, the infrastructure package imports them:

packages/infrastructure/package.json

```json
{
  "name": "@radiant/infrastructure",
  "version": "2.2.0",
  "private": true,
  "scripts": {
    "build": "tsc",
    "cdk": "cdk",
    "synth": "cdk synth",
    "deploy": "cdk deploy",
    "destroy": "cdk destroy"
  },
  "dependencies": {
    "@radiant/shared": "workspace:*",
    "aws-cdk-lib": "^2.120.0",
    "constructs": "^10.3.0",
    "source-map-support": "^0.5.21"
  },
  "devDependencies": {
    "@types/node": "^20.10.0",
    "aws-cdk": "^2.120.0",
    "typescript": "^5.3.0"
  }
}
```

packages/infrastructure/lib/config/tags.ts

```typescript
/**
 * CDK-specific tagging utilities
 * Types imported from @radiant/shared
 */

import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';

export interface RadiantTags {
  project: string;
  environment: string;
  appId: string;
  tier: number;
  version?: string;
  costCenter?: string;
}

export function applyTags(scope: Construct, tags: RadiantTags): void {
  cdk.Tags.of(scope).add('Project', tags.project);
  cdk.Tags.of(scope).add('Environment', tags.environment);
  cdk.Tags.of(scope).add('AppId', tags.appId);
  cdk.Tags.of(scope).add('Tier', tags.tier.toString());
  cdk.Tags.of(scope).add('Version', tags.version || '2.2.0');
  cdk.Tags.of(scope).add('ManagedBy', 'CDK');

  if (tags.costCenter) {
    cdk.Tags.of(scope).add('CostCenter', tags.costCenter);
  }
}

export function getResourceName(
  appId: string,
  environment: string,
  resource: string,
  suffix?: string
): string {
  const base = \`\${appId}-\${environment}-\${resource}\`;
  return suffix ? \`\${base}-\${suffix}\` : base;
}
```

> **NOTE:** The `tiers.ts` and `regions.ts` files are NOT created here. CDK stacks import these directly from `@radiant/shared`:
>
> ```typescript
> import { getTierConfig, TIER_CONFIGS, REGIONS } from '@radiant/shared';
> ```

1.3 SWIFT DEPLOYMENT APP

AI Implementation Notes

```
                    SWIFT FILE CREATION ORDER (for AI)


  PHASE 1: Project Setup (create first)

  1. Package.swift (or Xcode project)
  2. Info.plist
  3. RadiantDeployer.entitlements

  PHASE 2: Core Files (no dependencies)

  4. RadiantDeployerApp.swift (entry point)
  5. Models/ManagedApp.swift
  6. Models/Credentials.swift
  7. Models/Deployment.swift

  PHASE 3: State Management

  8. AppState.swift (depends on: Models)

  PHASE 4: Services (depend on Models)

  9.  Services/CredentialService.swift
  10. Services/CDKService.swift
  11. Services/AWSService.swift
  12. Services/APIService.swift

  PHASE 5: Views (depend on AppState, Services)

  13. Views/MainView.swift
  14. Views/AppsView.swift
  15. Views/DeployView.swift
  16. Views/SettingsView.swift
  17. Views/ProvidersView.swift
  18. Views/ModelsView.swift
```

Platform Requirements

```
  Platform: macOS 13.0+ (Ventura)
  Swift: 5.9+
  Xcode: 15.0+
```

```
  Architecture: Universal (arm64 + x86_64)
```

RadiantDeployer/Package.swift

```swift
// swift-tools-version: 5.9
// RADIANT Deployer - macOS Swift Package
// Platform: macOS 13.0+

import PackageDescription

let package = Package(
    name: "RadiantDeployer",
    platforms: [
        .macOS(.v13)
    ],
    products: [
        .executable(name: "RadiantDeployer", targets: ["RadiantDeployer"])
    ],
    dependencies: [
        // SQLCipher for encrypted credential storage
        .package(url: "https://github.com/nicklockwood/GRDB.swift.git", from: "6.24.0"),
    ],
    targets: [
        .executableTarget(
            name: "RadiantDeployer",
            dependencies: [
                .product(name: "GRDB", package: "GRDB.swift"),
            ],
            path: "Sources",
            resources: [
                .copy("Resources/Infrastructure"),
                .copy("Resources/NodeRuntime"),
            ]
        ),
        .testTarget(
            name: "RadiantDeployerTests",
            dependencies: ["RadiantDeployer"],
            path: "Tests"
        ),
    ]
)
```

RadiantDeployer/Info.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1
<plist version="1.0">
<dict>
    <key>CFBundleDevelopmentRegion</key>
    <string>en</string>
    <key>CFBundleExecutable</key>
    <string>RadiantDeployer</string>
    <key>CFBundleIconFile</key>
    <string>AppIcon</string>
    <key>CFBundleIdentifier</key>
    <string>com.radiant.deployer</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>6.0</string>
    <key>CFBundleName</key>
    <string>RADIANT Deployer</string>
    <key>CFBundlePackageType</key>
    <string>APPL</string>
    <key>CFBundleShortVersionString</key>
    <string>4.17.0</string>
    <key>CFBundleVersion</key>
    <string>1</string>
    <key>LSMinimumSystemVersion</key>
    <string>13.0</string>
    <key>NSHumanReadableCopyright</key>
    <string>Copyright © 2024 RADIANT. All rights reserved.</string>
    <key>NSMainStoryboardFile</key>
    <string></string>
    <key>NSPrincipalClass</key>
    <string>NSApplication</string>
    <key>LSApplicationCategoryType</key>
    <string>public.app-category.developer-tools</string>
</dict>
</plist>
```

RadiantDeployer/RadiantDeployer.entitlements

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1
<plist version="1.0">
<dict>
    <!-- App Sandbox (required for Mac App Store, optional otherwise) -->
    <key>com.apple.security.app-sandbox</key>
    <false/>
```

```
    <!-- Network access for AWS API calls -->
    <key>com.apple.security.network.client</key>
    <true/>

    <!-- File access for CDK operations -->
    <key>com.apple.security.files.user-selected.read-write</key>
    <true/>

    <!-- Keychain access for credential storage -->
    <key>com.apple.security.keychain</key>
    <true/>

    <!-- Process execution for CDK/Node commands -->
    <key>com.apple.security.cs.allow-unsigned-executable-memory</key>
    <true/>
</dict>
</plist>
```

RadiantDeployer/RadiantDeployerApp.swift

```swift
// MARK: - RADIANT Deployer App Entry Point
// Platform: macOS 13.0+
// Swift: 5.9+
// Dependencies: SwiftUI (system), AppState.swift

import SwiftUI

/// Main application entry point for RADIANT Deployer
/// This is the first file Xcode will look for when launching the app
@main
struct RadiantDeployerApp: App {
    // MARK: - State

    /// App-lifetime state object - created once, shared across all views
    @StateObject private var appState = AppState()

    // MARK: - Body

    var body: some Scene {
        WindowGroup {
            MainView()
                .environmentObject(appState)
                .frame(minWidth: 1200, minHeight: 800)
        }
        .windowStyle(.hiddenTitleBar)
        .commands {
```

```
            CommandGroup(replacing: .newItem) { }
        }

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

// MARK: - Preview

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

RadiantDeployer/AppState.swift

```
import SwiftUI
import Combine

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

    // 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 apiService = APIService()

    // MARK: - Initialization
```

```
    init() {
        Task {
            await loadInitialData()
        }
    }

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

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

    private func loadApps() async throws -> [ManagedApp] {
        // Load from local storage or API
        return ManagedApp.defaults
    }
}

// MARK: - Navigation
enum NavigationTab: String, CaseIterable, Identifiable, Sendable {
    case apps = "Apps"
    case deploy = "Deploy"
    case providers = "Providers"
    case models = "Models"
    case settings = "Settings"

    var id: String { rawValue }

    var icon: String {
        switch self {
        case .apps: return "square.grid.2x2"
        case .deploy: return "arrow.up.circle"
        case .providers: return "building.2"
        case .models: return "cpu"
        case .settings: return "gearshape"
        }
    }
}

// MARK: - Environment
enum Environment: String, CaseIterable, Identifiable, Sendable {
    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
    }
}
```

RadiantDeployer/Models/ManagedApp.swift

```
// MARK: - ManagedApp Model
// Platform: macOS 13.0+
// Dependencies: Foundation

import Foundation

// MARK: - Constants

/// Domain placeholder - replace with your actual domain during setup
let DOMAIN_PLACEHOLDER = "YOUR_DOMAIN.com"
```

```swift
// MARK: - ManagedApp

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

    /// Check if domain has been configured
    var isDomainConfigured: Bool {
        !domain.contains(DOMAIN_PLACEHOLDER)
    }

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

        subscript(env: Environment) -> 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"
        }
    }
}

// MARK: - Defaults
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(
            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)
            )
        )
    ]
}
```

RadiantDeployer/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]
}
```

RadiantDeployer/Models/Deployment.swift

```
import Foundation

// MARK: - Version Constant

/// Current RADIANT version - matches @radiant/shared/constants/version.ts
let RADIANT_VERSION = "4.17.0"

// MARK: - Deployment Progress

struct DeploymentProgress: Identifiable, Sendable {
    let id = UUID()
    var phase: DeploymentPhase
    var progress: Double // 0.0 - 1.0
    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"
        }
    }
}
```

```swift
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]?

    /// Create result with current RADIANT version
    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 {
    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"
        }
    }
}
```

RadiantDeployer/Services/CredentialService.swift

```
import Foundation
import SQLite3

/// Manages encrypted credential storage using SQLCipher
actor CredentialService {
    private var db: OpaquePointer?
    private let dbPath: URL
    private let encryptionKey: String

    init() {
        let appSupport = FileManager.default.urls(for: .applicationSupportDirectory, in: .userl
        let radiantDir = appSupport.appendingPathComponent("RadiantDeployer", isDirectory: true

        try? FileManager.default.createDirectory(at: radiantDir, withIntermediateDirectories: t
```

```
        dbPath = radiantDir.appendingPathComponent("credentials.db")

        // In production, derive from macOS Keychain
        encryptionKey = Self.getOrCreateEncryptionKey()
    }

    private static func getOrCreateEncryptionKey() -> String {
        let service = "YOUR_ORG_IDENTIFIER.radiant-deployer"
        let account = "db-encryption-key"

        // Try to get existing key
        let query: [String: Any] = [
            kSecClass as String: kSecClassGenericPassword,
            kSecAttrService as String: service,
            kSecAttrAccount as String: account,
            kSecReturnData as String: true
        ]

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

        if status == errSecSuccess, let data = result as? Data, let key = String(data: data, e
            return key
        }

        // Generate new key
        let newKey = UUID().uuidString + UUID().uuidString
        let keyData = newKey.data(using: .utf8)!

        let addQuery: [String: Any] = [
            kSecClass as String: kSecClassGenericPassword,
            kSecAttrService as String: service,
            kSecAttrAccount as String: account,
            kSecValueData as String: keyData,
            kSecAttrAccessible as String: kSecAttrAccessibleWhenUnlockedThisDeviceOnly
        ]

        SecItemAdd(addQuery as CFDictionary, nil)

        return newKey
    }

    private func openDatabase() throws {
        guard db == nil else { return }

        let flags = SQLITE_OPEN_CREATE | SQLITE_OPEN_READWRITE | SQLITE_OPEN_FULLMUTEX

        guard sqlite3_open_v2(dbPath.path, &db, flags, nil) == SQLITE_OK else {
```

```
        throw CredentialError.databaseOpenFailed
    }

    // Set encryption key (SQLCipher)
    let keySQL = "PRAGMA key = '\(encryptionKey)';"
    guard sqlite3_exec(db, keySQL, nil, nil, nil) == SQLITE_OK else {
        throw CredentialError.encryptionFailed
    }

    // Create tables
    let createSQL = """
        CREATE TABLE IF NOT EXISTS credentials (
            id TEXT PRIMARY KEY,
            name TEXT NOT NULL,
            access_key_id TEXT NOT NULL,
            secret_access_key TEXT NOT NULL,
            region TEXT NOT NULL,
            account_id TEXT,
            environment TEXT NOT NULL,
            created_at TEXT NOT NULL,
            last_validated_at TEXT,
            is_valid INTEGER
        );
    """

    guard sqlite3_exec(db, createSQL, nil, nil, nil) == SQLITE_OK else {
        throw CredentialError.tableCreationFailed
    }
}

func loadCredentials() async throws -> [CredentialSet] {
    try openDatabase()

    let query = "SELECT * FROM credentials ORDER BY created_at DESC;"
    var statement: OpaquePointer?

    guard sqlite3_prepare_v2(db, query, -1, &statement, nil) == SQLITE_OK else {
        throw CredentialError.queryFailed
    }

    defer { sqlite3_finalize(statement) }

    var credentials: [CredentialSet] = []
    let dateFormatter = ISO8601DateFormatter()

    while sqlite3_step(statement) == SQLITE_ROW {
        let id = String(cString: sqlite3_column_text(statement, 0))
        let name = String(cString: sqlite3_column_text(statement, 1))
```

```
            let accessKeyId = String(cString: sqlite3_column_text(statement, 2))
            let secretAccessKey = String(cString: sqlite3_column_text(statement, 3))
            let region = String(cString: sqlite3_column_text(statement, 4))
            let accountId = sqlite3_column_text(statement, 5).map { String(cString: $0) }
            let environment = String(cString: sqlite3_column_text(statement, 6))
            let createdAtStr = String(cString: sqlite3_column_text(statement, 7))
            let lastValidatedAtStr = sqlite3_column_text(statement, 8).map { String(cString: $
            let isValid = sqlite3_column_type(statement, 9) != SQLITE_NULL ? sqlite3_column_in

            credentials.append(CredentialSet(
                id: id,
                name: name,
                accessKeyId: accessKeyId,
                secretAccessKey: secretAccessKey,
                region: region,
                accountId: accountId,
                environment: CredentialEnvironment(rawValue: environment) ?? .shared,
                createdAt: dateFormatter.date(from: createdAtStr) ?? Date(),
                lastValidatedAt: lastValidatedAtStr.flatMap { dateFormatter.date(from: $0) },
                isValid: isValid
            ))
        }

        return credentials
    }

    func saveCredential(_ credential: CredentialSet) async throws {
        try openDatabase()

        let dateFormatter = ISO8601DateFormatter()

        let upsertSQL = """
            INSERT OR REPLACE INTO credentials
            (id, name, access_key_id, secret_access_key, region, account_id, environment, creat
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
        """

        var statement: OpaquePointer?
        guard sqlite3_prepare_v2(db, upsertSQL, -1, &statement, nil) == SQLITE_OK else {
            throw CredentialError.saveFailed
        }

        defer { sqlite3_finalize(statement) }

        sqlite3_bind_text(statement, 1, credential.id, -1, nil)
        sqlite3_bind_text(statement, 2, credential.name, -1, nil)
        sqlite3_bind_text(statement, 3, credential.accessKeyId, -1, nil)
        sqlite3_bind_text(statement, 4, credential.secretAccessKey, -1, nil)
```

```
        sqlite3_bind_text(statement, 5, credential.region, -1, nil)

        if let accountId = credential.accountId {
            sqlite3_bind_text(statement, 6, accountId, -1, nil)
        } else {
            sqlite3_bind_null(statement, 6)
        }

        sqlite3_bind_text(statement, 7, credential.environment.rawValue, -1, nil)
        sqlite3_bind_text(statement, 8, dateFormatter.string(from: credential.createdAt), -1, n

        if let lastValidated = credential.lastValidatedAt {
            sqlite3_bind_text(statement, 9, dateFormatter.string(from: lastValidated), -1, nil)
        } else {
            sqlite3_bind_null(statement, 9)
        }

        if let isValid = credential.isValid {
            sqlite3_bind_int(statement, 10, isValid ? 1 : 0)
        } else {
            sqlite3_bind_null(statement, 10)
        }

        guard sqlite3_step(statement) == SQLITE_DONE else {
            throw CredentialError.saveFailed
        }
    }

    func deleteCredential(id: String) async throws {
        try openDatabase()

        let deleteSQL = "DELETE FROM credentials WHERE id = ?;"
        var statement: OpaquePointer?

        guard sqlite3_prepare_v2(db, deleteSQL, -1, &statement, nil) == SQLITE_OK else {
            throw CredentialError.deleteFailed
        }

        defer { sqlite3_finalize(statement) }

        sqlite3_bind_text(statement, 1, id, -1, nil)

        guard sqlite3_step(statement) == SQLITE_DONE else {
            throw CredentialError.deleteFailed
        }
    }

    deinit {
```

```
        if db != nil {
            sqlite3_close(db)
        }
    }
}

enum CredentialError: Error, LocalizedError {
    case databaseOpenFailed
    case encryptionFailed
    case tableCreationFailed
    case queryFailed
    case saveFailed
    case deleteFailed
    case validationFailed(String)

    var errorDescription: String? {
        switch self {
        case .databaseOpenFailed: return "Failed to open credential database"
        case .encryptionFailed: return "Failed to set database encryption"
        case .tableCreationFailed: return "Failed to create database tables"
        case .queryFailed: return "Failed to query credentials"
        case .saveFailed: return "Failed to save credential"
        case .deleteFailed: return "Failed to delete credential"
        case .validationFailed(let reason): return "Credential validation failed: \(reason)"
        }
    }
}
```

RadiantDeployer/Services/CDKService.swift

```
import Foundation
import Combine

/// Manages CDK deployments from bundled infrastructure code
actor CDKService {
    private var currentProcess: Process?
    private let bundledNodePath: URL
    private let bundledInfraPath: URL

    init() {
        let bundle = Bundle.main
        bundledNodePath = bundle.resourceURL!.appendingPathComponent("NodeRuntime/bin/node")
        bundledInfraPath = bundle.resourceURL!.appendingPathComponent("Infrastructure")
    }

    func deploy(
        app: ManagedApp,
        environment: Environment,
```

```
    credentials: CredentialSet,
    onPhase: @escaping (DeploymentPhase) -> Void,
    onLog: @escaping (LogEntry) -> Void,
    onProgress: @escaping (Double) -> Void
) async throws -> DeploymentResult {
    let startTime = Date()
    var outputs: DeploymentOutputs?
    var errors: [String] = []

    // Create temporary working directory
    let workDir = FileManager.default.temporaryDirectory.appendingPathComponent(UUID().uui
    try FileManager.default.createDirectory(at: workDir, withIntermediateDirectories: true

    defer {
        try? FileManager.default.removeItem(at: workDir)
    }

    // Copy infrastructure to working directory
    let infraWorkDir = workDir.appendingPathComponent("infrastructure")
    try FileManager.default.copyItem(at: bundledInfraPath, to: infraWorkDir)

    do {
        // Phase 1: Validate credentials
        onPhase(.validating)
        onLog(LogEntry(timestamp: Date(), level: .info, message: "Validating AWS credential
        try await validateCredentials(credentials)
        onProgress(0.05)
        onLog(LogEntry(timestamp: Date(), level: .success, message: "Credentials validated

        // Phase 2: Install dependencies
        onPhase(.bootstrapping)
        onLog(LogEntry(timestamp: Date(), level: .info, message: "Installing CDK dependenci
        try await runCommand("npm", args: ["ci"], in: infraWorkDir, credentials: credential
        onProgress(0.10)

        // Phase 3: Bootstrap CDK
        onLog(LogEntry(timestamp: Date(), level: .info, message: "Bootstrapping CDK...", me
        try await runCDK(
            command: "bootstrap",
            args: ["aws://\(credentials.accountId ?? "")/\(credentials.region)"],
            in: infraWorkDir,
            credentials: credentials,
            context: buildContext(app: app, environment: environment),
            onLog: onLog
        )
        onProgress(0.15)

        // Phase 4: Synthesize
```

```
onPhase(.synthesizing)
onLog(LogEntry(timestamp: Date(), level: .info, message: "Synthesizing CloudFormati
try await runCDK(
    command: "synth",
    args: ["--all"],
    in: infraWorkDir,
    credentials: credentials,
    context: buildContext(app: app, environment: environment),
    onLog: onLog
)
onProgress(0.20)

// Phase 5-11: Deploy stacks
let stacks = [
    ("Foundation", DeploymentPhase.deployingFoundation, 0.30),
    ("Networking", DeploymentPhase.deployingNetworking, 0.40),
    ("Security", DeploymentPhase.deploySecurity, 0.50),
    ("Data", DeploymentPhase.deployingData, 0.60),
    ("AI", DeploymentPhase.deployingAI, 0.70),
    ("API", DeploymentPhase.deployingAPI, 0.80),
    ("Admin", DeploymentPhase.deployingAdmin, 0.90)
]

for (stackName, phase, progress) in stacks {
    onPhase(phase)
    onLog(LogEntry(timestamp: Date(), level: .info, message: "Deploying \(stackNam

    let stackOutputs = try await runCDK(
        command: "deploy",
        args: ["\(app.id)-\(environment.rawValue)-\(stackName.lowercased())", "--re
        in: infraWorkDir,
        credentials: credentials,
        context: buildContext(app: app, environment: environment),
        onLog: onLog
    )

    onProgress(progress)
    onLog(LogEntry(timestamp: Date(), level: .success, message: "\(stackName) stac
}

// Phase 12: Run migrations
onPhase(.runningMigrations)
onLog(LogEntry(timestamp: Date(), level: .info, message: "Running database migrati
try await runMigrations(app: app, environment: environment, credentials: credential
onProgress(0.95)

// Phase 13: Verify
onPhase(.verifying)
```

```
            onLog(LogEntry(timestamp: Date(), level: .info, message: "Verifying deployment...")
            outputs = try await verifyDeployment(app: app, environment: environment, workDir: w
            onProgress(1.0)

            onPhase(.complete)
            onLog(LogEntry(timestamp: Date(), level: .success, message: "Deployment complete!")

        } catch {
            errors.append(error.localizedDescription)
            onPhase(.failed)
            onLog(LogEntry(timestamp: Date(), level: .error, message: "Deployment failed: \(er
        }

        return DeploymentResult.create(
            appId: app.id,
            environment: environment.rawValue,
            success: errors.isEmpty,
            startedAt: startTime,
            outputs: outputs,
            errors: errors.isEmpty ? nil : errors
        )
    }

    private func validateCredentials(_ credentials: CredentialSet) async throws {
        // Find AWS CLI - check multiple locations for compatibility
        let awsPaths = [
            "/opt/homebrew/bin/aws",      // Homebrew on Apple Silicon
            "/usr/local/bin/aws",          // Homebrew on Intel
            "/usr/bin/aws",                // System install
        ]

        guard let awsPath = awsPaths.first(where: { FileManager.default.fileExists(atPath: $0)
            throw DeploymentError.awsCliNotFound
        }

        let process = Process()
        process.executableURL = URL(fileURLWithPath: awsPath)
        process.arguments = ["sts", "get-caller-identity"]
        process.environment = buildAWSEnvironment(credentials)

        let pipe = Pipe()
        process.standardOutput = pipe
        process.standardError = pipe

        try process.run()
        process.waitUntilExit()

        guard process.terminationStatus == 0 else {
```

```
            throw DeploymentError.credentialValidationFailed
        }
    }

    private func runCommand(
        _ command: String,
        args: [String],
        in directory: URL,
        credentials: CredentialSet,
        onLog: @escaping (LogEntry) -> Void
    ) async throws {
        let process = Process()
        process.executableURL = URL(fileURLWithPath: "/usr/bin/env")
        process.arguments = [command] + args
        process.currentDirectoryURL = directory
        process.environment = buildAWSEnvironment(credentials)

        let pipe = Pipe()
        process.standardOutput = pipe
        process.standardError = pipe

        pipe.fileHandleForReading.readabilityHandler = { handle in
            let data = handle.availableData
            if let output = String(data: data, encoding: .utf8), !output.isEmpty {
                onLog(LogEntry(timestamp: Date(), level: .debug, message: output.trimmingCharac
            }
        }

        try process.run()
        process.waitUntilExit()

        pipe.fileHandleForReading.readabilityHandler = nil

        guard process.terminationStatus == 0 else {
            throw DeploymentError.commandFailed(command, Int(process.terminationStatus))
        }
    }

    @discardableResult
    private func runCDK(
        command: String,
        args: [String],
        in directory: URL,
        credentials: CredentialSet,
        context: [String: String],
        onLog: @escaping (LogEntry) -> Void
    ) async throws -> [String: Any]? {
        var allArgs = [command] + args
```

```
        for (key, value) in context {
            allArgs.append("--context")
            allArgs.append("\(key)=\(value)")
        }

        try await runCommand("npx", args: ["cdk"] + allArgs, in: directory, credentials: creder

        // Read outputs if available
        let outputsFile = directory.appendingPathComponent("outputs.json")
        if FileManager.default.fileExists(atPath: outputsFile.path),
           let data = try? Data(contentsOf: outputsFile),
           let json = try? JSONSerialization.jsonObject(with: data) as? [String: Any] {
            return json
        }

        return nil
    }

    private func buildContext(app: ManagedApp, environment: Environment) -> [String: String] {
        let tier = app.environments[environment].tier
        return [
            "appId": app.id,
            "appName": app.name,
            "domain": app.domain,
            "environment": environment.rawValue.lowercased(),
            "tier": String(tier)
        ]
    }

    private func buildAWSEnvironment(_ credentials: CredentialSet) -> [String: String] {
        var env = ProcessInfo.processInfo.environment
        env["AWS_ACCESS_KEY_ID"] = credentials.accessKeyId
        env["AWS_SECRET_ACCESS_KEY"] = credentials.secretAccessKey
        env["AWS_DEFAULT_REGION"] = credentials.region
        env["AWS_REGION"] = credentials.region
        return env
    }

    private func runMigrations(
        app: ManagedApp,
        environment: Environment,
        credentials: CredentialSet,
        onLog: @escaping (LogEntry) -> Void
    ) async throws {
        // Invoke migration Lambda
        onLog(LogEntry(timestamp: Date(), level: .info, message: "Invoking migration Lambda...")
        // Implementation depends on how migrations are structured
```

```
    }

    private func verifyDeployment(
        app: ManagedApp,
        environment: Environment,
        workDir: URL
    ) async throws -> DeploymentOutputs {
        // Read outputs from CDK deployment
        let outputsFile = workDir.appendingPathComponent("outputs.json")

        guard FileManager.default.fileExists(atPath: outputsFile.path),
              let data = try? Data(contentsOf: outputsFile),
              let json = try? JSONSerialization.jsonObject(with: data) as? [String: Any] else {
            throw DeploymentError.outputsNotFound
        }

        // Parse outputs (structure depends on CDK output format)
        // This is a simplified version
        return DeploymentOutputs(
            apiUrl: json["ApiUrl"] as? String ?? "",
            graphqlUrl: json["GraphQLUrl"] as? String ?? "",
            dashboardUrl: json["DashboardUrl"] as? String ?? "",
            cognitoUserPoolId: json["CognitoUserPoolId"] as? String ?? "",
            cognitoClientId: json["CognitoClientId"] as? String ?? "",
            cognitoDomain: json["CognitoDomain"] as? String ?? "",
            auroraEndpoint: json["AuroraEndpoint"] as? String ?? "",
            s3MediaBucket: json["S3MediaBucket"] as? String ?? "",
            cloudfrontDistribution: json["CloudFrontDistribution"] as? String ?? ""
        )
    }

    func cancel() {
        currentProcess?.terminate()
    }
}

enum DeploymentError: Error, LocalizedError {
    case credentialValidationFailed
    case commandFailed(String, Int)
    case outputsNotFound
    case verificationFailed(String)
    case awsCliNotFound
    case nodeNotFound
    case cdkBootstrapFailed

    var errorDescription: String? {
        switch self {
        case .credentialValidationFailed:
```

```
            return "AWS credential validation failed"
        case .commandFailed(let cmd, let code):
            return "Command '\(cmd)' failed with exit code \(code)"
        case .outputsNotFound:
            return "Deployment outputs not found"
        case .verificationFailed(let reason):
            return "Deployment verification failed: \(reason)"
        case .awsCliNotFound:
            return "AWS CLI not found. Install via: brew install awscli"
        case .nodeNotFound:
            return "Node.js not found in bundled resources"
        case .cdkBootstrapFailed:
            return "CDK bootstrap failed. Check AWS permissions."
        }
    }
}
```

RadiantDeployer/Services/AWSService.swift

```
import Foundation

/// Direct AWS API interactions (for validation, health checks, etc.)
actor AWSService {

    func validateCredentials(_ credentials: CredentialSet) async throws -> AWSAccount {
        let process = Process()
        process.executableURL = URL(fileURLWithPath: "/usr/local/bin/aws")
        process.arguments = ["sts", "get-caller-identity", "--output", "json"]

        var env = ProcessInfo.processInfo.environment
        env["AWS_ACCESS_KEY_ID"] = credentials.accessKeyId
        env["AWS_SECRET_ACCESS_KEY"] = credentials.secretAccessKey
        env["AWS_DEFAULT_REGION"] = credentials.region
        process.environment = env

        let pipe = Pipe()
        process.standardOutput = pipe
        process.standardError = Pipe()

        try process.run()
        process.waitUntilExit()

        guard process.terminationStatus == 0 else {
            throw AWSError.credentialValidationFailed
        }

        let data = pipe.fileHandleForReading.readDataToEndOfFile()
        guard let json = try? JSONSerialization.jsonObject(with: data) as? [String: Any],
```

```
            let accountId = json["Account"] as? String else {
            throw AWSError.invalidResponse
        }

        // Get account alias
        let alias = try? await getAccountAlias(credentials: credentials)

        return AWSAccount(
            accountId: accountId,
            accountAlias: alias,
            regions: ["us-east-1", "us-west-2", "eu-west-1"] // Could be dynamic
        )
    }

    private func getAccountAlias(credentials: CredentialSet) async throws -> String? {
        let process = Process()
        process.executableURL = URL(fileURLWithPath: "/usr/local/bin/aws")
        process.arguments = ["iam", "list-account-aliases", "--output", "json"]

        var env = ProcessInfo.processInfo.environment
        env["AWS_ACCESS_KEY_ID"] = credentials.accessKeyId
        env["AWS_SECRET_ACCESS_KEY"] = credentials.secretAccessKey
        env["AWS_DEFAULT_REGION"] = credentials.region
        process.environment = env

        let pipe = Pipe()
        process.standardOutput = pipe
        process.standardError = Pipe()

        try process.run()
        process.waitUntilExit()

        guard process.terminationStatus == 0 else { return nil }

        let data = pipe.fileHandleForReading.readDataToEndOfFile()
        guard let json = try? JSONSerialization.jsonObject(with: data) as? [String: Any],
              let aliases = json["AccountAliases"] as? [String],
              let alias = aliases.first else {
            return nil
        }

        return alias
    }

    func checkHealth(app: ManagedApp, environment: Environment, credentials: CredentialSet) asy
        // Check API Gateway health endpoint
        guard let apiUrl = app.environments[environment].apiUrl else {
            return .unknown
```

```
        }

        guard let url = URL(string: "\(apiUrl)/health") else {
            return .unknown
        }

        var request = URLRequest(url: url)
        request.timeoutInterval = 10

        do {
            let (_, response) = try await URLSession.shared.data(for: request)
            guard let httpResponse = response as? HTTPURLResponse else {
                return .unknown
            }

            switch httpResponse.statusCode {
            case 200: return .healthy
            case 503: return .degraded
            default: return .unhealthy
            }
        } catch {
            return .unhealthy
        }
    }
}

enum AWSError: Error, LocalizedError {
    case credentialValidationFailed
    case invalidResponse
    case apiError(String)

    var errorDescription: String? {
        switch self {
        case .credentialValidationFailed:
            return "AWS credential validation failed"
        case .invalidResponse:
            return "Invalid response from AWS"
        case .apiError(let message):
            return "AWS API error: \(message)"
        }
    }
}
```

RadiantDeployer/Services/APIService.swift

```
import Foundation

/// Communicates with deployed RADIANT APIs
```

```swift
actor APIService {
    private let session = URLSession.shared
    private var authToken: String?

    func setAuthToken(_ token: String) {
        self.authToken = token
    }

    func fetchProviders(baseUrl: String) async throws -> [Provider] {
        let url = URL(string: "\(baseUrl)/api/v2/providers")!
        return try await fetch(url: url)
    }

    func fetchModels(baseUrl: String) async throws -> [Model] {
        let url = URL(string: "\(baseUrl)/api/v2/models")!
        return try await fetch(url: url)
    }

    private func fetch<T: Decodable>(url: URL) async throws -> T {
        var request = URLRequest(url: url)
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")

        if let token = authToken {
            request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
        }

        let (data, response) = try await session.data(for: request)

        guard let httpResponse = response as? HTTPURLResponse,
              (200...299).contains(httpResponse.statusCode) else {
            throw APIError.requestFailed
        }

        return try JSONDecoder().decode(T.self, from: data)
    }
}

// Placeholder types for API responses
struct Provider: Codable, Identifiable {
    let id: String
    let name: String
    let status: String
}

struct Model: Codable, Identifiable {
    let id: String
    let name: String
    let providerId: String
```

```
}

enum APIError: Error {
    case requestFailed
    case decodingFailed
}
```

PART 4: SWIFT VIEWS

RadiantDeployer/Views/MainView.swift

```
import SwiftUI

struct MainView: View {
    @EnvironmentObject var appState: AppState

    var body: some View {
        NavigationSplitView {
            Sidebar()
        } detail: {
            DetailView()
        }
        .navigationSplitViewStyle(.balanced)
        .alert(item: $appState.error) { error in
            Alert(
                title: Text("Error"),
                message: Text(error.localizedDescription),
                dismissButton: .default(Text("OK"))
            )
        }
    }
}

struct Sidebar: View {
    @EnvironmentObject var appState: AppState

    var body: some View {
        List(selection: $appState.selectedTab) {
            Section("Navigation") {
                ForEach(NavigationTab.allCases) { tab in
                    Label(tab.rawValue, systemImage: tab.icon)
                        .tag(tab)
                }
            }

            if !appState.apps.isEmpty {
                Section("Apps") {
```

```
                    ForEach(appState.apps) { app in
                        AppRow(app: app)
                    }
                }
            }
        }
        .listStyle(.sidebar)
        .frame(minWidth: 220)
        .toolbar {
            ToolbarItem(placement: .automatic) {
                Button(action: { /* Add app */ }) {
                    Image(systemName: "plus")
                }
            }
        }
    }
}

struct AppRow: View {
    @EnvironmentObject var appState: AppState
    let app: ManagedApp

    var body: some View {
        HStack {
            VStack(alignment: .leading, spacing: 2) {
                Text(app.name)
                    .font(.headline)
                Text(app.domain)
                    .font(.caption)
                    .foregroundColor(.secondary)
            }

            Spacer()

            HStack(spacing: 4) {
                EnvironmentDot(status: app.environments.dev)
                EnvironmentDot(status: app.environments.staging)
                EnvironmentDot(status: app.environments.prod)
            }
        }
        .padding(.vertical, 4)
        .contentShape(Rectangle())
        .onTapGesture {
            appState.selectedApp = app
            appState.selectedTab = .deploy
        }
    }
}
```

```
struct EnvironmentDot: View {
    let status: EnvironmentStatus

    var body: some View {
        Circle()
            .fill(status.deployed ? Color(status.healthStatus.color) : Color.gray.opacity(0.3))
            .frame(width: 8, height: 8)
    }
}

struct DetailView: View {
    @EnvironmentObject var appState: AppState

    var body: some View {
        Group {
            switch appState.selectedTab {
            case .apps:
                AppsView()
            case .deploy:
                DeployView()
            case .providers:
                ProvidersView()
            case .models:
                ModelsView()
            case .settings:
                SettingsView()
            }
        }
    }
}
```

RadiantDeployer/Views/AppsView.swift

```
import SwiftUI

struct AppsView: View {
    @EnvironmentObject var appState: AppState
    @State private var showingAddApp = false

    var body: some View {
        ScrollView {
            LazyVGrid(columns: [GridItem(.adaptive(minimum: 300))], spacing: 20) {
                ForEach(appState.apps) { app in
                    AppCard(app: app)
                }

                AddAppCard(action: { showingAddApp = true })
```

```
            }
            .padding()
        }
        .navigationTitle("Applications")
        .sheet(isPresented: $showingAddApp) {
            AddAppSheet()
        }
    }
}

struct AppCard: View {
    @EnvironmentObject var appState: AppState
    let app: ManagedApp

    var body: some View {
        VStack(alignment: .leading, spacing: 16) {
            HStack {
                VStack(alignment: .leading) {
                    Text(app.name)
                        .font(.title2)
                        .fontWeight(.semibold)
                    Text(app.domain)
                        .font(.subheadline)
                        .foregroundColor(.secondary)
                }

                Spacer()

                Menu {
                    Button("Edit") { }
                    Button("View Logs") { }
                    Divider()
                    Button("Delete", role: .destructive) { }
                } label: {
                    Image(systemName: "ellipsis.circle")
                        .font(.title3)
                }
            }

            Divider()

            HStack(spacing: 20) {
                EnvironmentStatusView(name: "DEV", status: app.environments.dev, color: .blue)
                EnvironmentStatusView(name: "STAGING", status: app.environments.staging, color
                EnvironmentStatusView(name: "PROD", status: app.environments.prod, color: .gre
            }

            HStack {
```

```
                Button("Deploy") {
                    appState.selectedApp = app
                    appState.selectedTab = .deploy
                }
                .buttonStyle(.borderedProminent)

                Button("Dashboard") {
                    if let url = URL(string: app.environments.prod.dashboardUrl ?? "") {
                        NSWorkspace.shared.open(url)
                    }
                }
                .buttonStyle(.bordered)
                .disabled(app.environments.prod.dashboardUrl == nil)
            }
        }
        .padding()
        .background(Color(.windowBackgroundColor))
        .cornerRadius(12)
        .shadow(radius: 2)
    }
}

struct EnvironmentStatusView: View {
    let name: String
    let status: EnvironmentStatus
    let color: Color

    var body: some View {
        VStack(spacing: 4) {
            Text(name)
                .font(.caption)
                .fontWeight(.medium)
                .foregroundColor(.secondary)

            Circle()
                .fill(status.deployed ? Color(status.healthStatus.color) : Color.gray.opacity(0
                .frame(width: 12, height: 12)

            if status.deployed, let version = status.version {
                Text("v\(version)")
                    .font(.caption2)
                    .foregroundColor(.secondary)
            }
        }
    }
}

struct AddAppCard: View {
```

```
    let action: () -> Void

    var body: some View {
        Button(action: action) {
            VStack(spacing: 12) {
                Image(systemName: "plus.circle")
                    .font(.system(size: 40))
                    .foregroundColor(.accentColor)
                Text("Add Application")
                    .font(.headline)
            }
            .frame(maxWidth: .infinity, minHeight: 150)
            .background(Color(.windowBackgroundColor).opacity(0.5))
            .cornerRadius(12)
            .overlay(
                RoundedRectangle(cornerRadius: 12)
                    .stroke(style: StrokeStyle(lineWidth: 2, dash: [8]))
                    .foregroundColor(.secondary.opacity(0.3))
            )
        }
        .buttonStyle(.plain)
    }
}

struct AddAppSheet: View {
    @Environment(\.dismiss) var dismiss
    @State private var appId = ""
    @State private var appName = ""
    @State private var domain = ""

    var body: some View {
        VStack(spacing: 20) {
            Text("Add New Application")
                .font(.title2)
                .fontWeight(.semibold)

            Form {
                TextField("App ID (lowercase, no spaces)", text: $appId)
                TextField("App Name", text: $appName)
                TextField("Domain (e.g., myapp.YOUR_DOMAIN.com)", text: $domain)
            }
            .formStyle(.grouped)

            HStack {
                Button("Cancel") { dismiss() }
                    .buttonStyle(.bordered)

                Button("Add Application") {
```

```
                    // Add app logic
                    dismiss()
                }
                .buttonStyle(.borderedProminent)
                .disabled(appId.isEmpty || appName.isEmpty || domain.isEmpty)
            }
        }
        .padding()
        .frame(width: 400)
    }
}
```

RadiantDeployer/Views/DeployView.swift

```
import SwiftUI

struct DeployView: View {
    @EnvironmentObject var appState: AppState
    @State private var selectedCredential: CredentialSet?

    var body: some View {
        HSplitView {
            // Left: Configuration
            DeployConfigPanel(selectedCredential: $selectedCredential)
                .frame(minWidth: 350, maxWidth: 450)

            // Right: Logs
            DeployLogPanel()
                .frame(minWidth: 500)
        }
        .navigationTitle("Deploy")
    }
}

struct DeployConfigPanel: View {
    @EnvironmentObject var appState: AppState
    @Binding var selectedCredential: CredentialSet?
    @State private var selectedTier: Int = 1

    var body: some View {
        ScrollView {
            VStack(alignment: .leading, spacing: 24) {
                // App Selection
                Section {
                    Picker("Application", selection: $appState.selectedApp) {
                        Text("Select an app").tag(nil as ManagedApp?)
                        ForEach(appState.apps) { app in
                            Text(app.name).tag(app as ManagedApp?)
```

```
            }
        }
        .pickerStyle(.menu)
    } header: {
        Text("Application")
            .font(.headline)
    }

    // Environment Selection
    Section {
        Picker("Environment", selection: $appState.selectedEnvironment) {
            ForEach(Environment.allCases) { env in
                HStack {
                    Circle()
                        .fill(env.color)
                        .frame(width: 8, height: 8)
                    Text(env.rawValue)
                }
                .tag(env)
            }
        }
        .pickerStyle(.segmented)
    } header: {
        Text("Environment")
            .font(.headline)
    }

    // Credentials
    Section {
        Picker("AWS Credentials", selection: $selectedCredential) {
            Text("Select credentials").tag(nil as CredentialSet?)
            ForEach(appState.credentials) { cred in
                Text("\(cred.name) (\(cred.environment.rawValue))")
                    .tag(cred as CredentialSet?)
            }
        }
        .pickerStyle(.menu)

        if let cred = selectedCredential {
            HStack {
                Label(cred.region, systemImage: "globe")
                Spacer()
                if let valid = cred.isValid {
                    Image(systemName: valid ? "checkmark.circle.fill" : "xmark.cir
                        .foregroundColor(valid ? .green : .red)
                }
            }
            .font(.caption)
```

```
            .foregroundColor(.secondary)
        }

        Button("Manage Credentials") {
            appState.selectedTab = .settings
        }
        .font(.caption)
    } header: {
        Text("AWS Credentials")
            .font(.headline)
    }

    // Tier Selection
    Section {
        Picker("Infrastructure Tier", selection: $selectedTier) {
            Text("1 - SEED (Development)").tag(1)
            Text("2 - STARTUP (Small Production)").tag(2)
            Text("3 - GROWTH (Medium Production)").tag(3)
            Text("4 - SCALE (Large + Multi-Region)").tag(4)
            Text("5 - ENTERPRISE (Full Compliance)").tag(5)
        }
        .pickerStyle(.menu)
    } header: {
        Text("Infrastructure Tier")
            .font(.headline)
    }

    Divider()

    // Deploy Button
    Button(action: startDeployment) {
        HStack {
            if appState.isDeploying {
                ProgressView()
                    .scaleEffect(0.8)
            } else {
                Image(systemName: "arrow.up.circle.fill")
            }
            Text(appState.isDeploying ? "Deploying..." : "Deploy")
        }
        .frame(maxWidth: .infinity)
    }
    .buttonStyle(.borderedProminent)
    .controlSize(.large)
    .disabled(!canDeploy)

    // Progress
    if let progress = appState.deploymentProgress {
```

```
                VStack(alignment: .leading, spacing: 8) {
                    HStack {
                        Image(systemName: progress.phase.icon)
                        Text(progress.phase.rawValue)
                        Spacer()
                        Text("\(Int(progress.progress * 100))%")
                    }
                    .font(.subheadline)

                    ProgressView(value: progress.progress)
                }
                .padding()
                .background(Color(.textBackgroundColor))
                .cornerRadius(8)
            }
        }
        .padding()
    }
}

private var canDeploy: Bool {
    appState.selectedApp != nil &&
    selectedCredential != nil &&
    !appState.isDeploying
}

private func startDeployment() {
    guard let app = appState.selectedApp,
          let credentials = selectedCredential else { return }

    appState.isDeploying = true
    appState.deploymentLogs = []

    Task {
        let result = try await appState.cdkService.deploy(
            app: app,
            environment: appState.selectedEnvironment,
            credentials: credentials,
            onPhase: { phase in
                Task { @MainActor in
                    appState.deploymentProgress = DeploymentProgress(
                        phase: phase,
                        progress: phase.progress,
                        currentStack: nil,
                        message: nil,
                        startedAt: Date()
                    )
                }
```

```
                },
                onLog: { log in
                    Task { @MainActor in
                        appState.deploymentLogs.append(log)
                    }
                },
                onProgress: { progress in
                    Task { @MainActor in
                        appState.deploymentProgress?.progress = progress
                    }
                }
            )

            await MainActor.run {
                appState.isDeploying = false

                if result.success {
                    // Update app status
                    if let index = appState.apps.firstIndex(where: { $0.id == app.id }) {
                        appState.apps[index].environments[appState.selectedEnvironment].deploye
                        appState.apps[index].environments[appState.selectedEnvironment].versio
                        appState.apps[index].environments[appState.selectedEnvironment].healthS
                        appState.apps[index].environments[appState.selectedEnvironment].apiUrl
                        appState.apps[index].environments[appState.selectedEnvironment].dashboa
                    }
                }
            }
        }
    }
}

struct DeployLogPanel: View {
    @EnvironmentObject var appState: AppState

    var body: some View {
        VStack(alignment: .leading, spacing: 0) {
            HStack {
                Text("Deployment Logs")
                    .font(.headline)
                Spacer()
                Button(action: { appState.deploymentLogs = [] }) {
                    Image(systemName: "trash")
                }
                .buttonStyle(.borderless)
                .disabled(appState.deploymentLogs.isEmpty)
            }
            .padding()
```

```
            Divider()

            ScrollViewReader { proxy in
                ScrollView {
                    LazyVStack(alignment: .leading, spacing: 4) {
                        ForEach(appState.deploymentLogs) { log in
                            LogRow(log: log)
                                .id(log.id)
                        }
                    }
                    .padding()
                }
                .onChange(of: appState.deploymentLogs.count) { _ in
                    if let lastLog = appState.deploymentLogs.last {
                        withAnimation {
                            proxy.scrollTo(lastLog.id, anchor: .bottom)
                        }
                    }
                }
            }
            .background(Color(.textBackgroundColor))
        }
    }
}

struct LogRow: View {
    let log: LogEntry

    private var timeString: String {
        let formatter = DateFormatter()
        formatter.dateFormat = "HH:mm:ss.SSS"
        return formatter.string(from: log.timestamp)
    }

    var body: some View {
        HStack(alignment: .top, spacing: 8) {
            Text(timeString)
                .font(.system(.caption, design: .monospaced))
                .foregroundColor(.secondary)

            Image(systemName: log.level.icon)
                .foregroundColor(Color(log.level.color))
                .font(.caption)

            Text(log.message)
                .font(.system(.caption, design: .monospaced))
                .foregroundColor(Color(log.level.color))
                .textSelection(.enabled)
```

```
        }
    }
}
```

RadiantDeployer/Views/SettingsView.swift

```
import SwiftUI

struct SettingsView: View {
    @EnvironmentObject var appState: AppState

    var body: some View {
        TabView {
            CredentialsSettingsView()
                .tabItem {
                    Label("Credentials", systemImage: "key")
                }

            GeneralSettingsView()
                .tabItem {
                    Label("General", systemImage: "gearshape")
                }

            AboutView()
                .tabItem {
                    Label("About", systemImage: "info.circle")
                }
        }
        .padding()
    }
}

struct CredentialsSettingsView: View {
    @EnvironmentObject var appState: AppState
    @State private var showingAddCredential = false
    @State private var selectedCredential: CredentialSet?
    @State private var isValidating = false

    var body: some View {
        VStack(alignment: .leading, spacing: 20) {
            HStack {
                Text("AWS Credentials")
                    .font(.title2)
                    .fontWeight(.semibold)

                Spacer()

                Button(action: { showingAddCredential = true }) {
```

```
                Label("Add Credential", systemImage: "plus")
            }
            .buttonStyle(.borderedProminent)
        }

        List(selection: $selectedCredential) {
            ForEach(appState.credentials) { credential in
                CredentialRow(credential: credential, isValidating: isValidating && selecte
                    .tag(credential)
            }
            .onDelete(perform: deleteCredentials)
        }
        .listStyle(.inset)

        if let credential = selectedCredential {
            HStack {
                Button("Validate") {
                    validateCredential(credential)
                }
                .disabled(isValidating)

                Button("Delete", role: .destructive) {
                    deleteCredential(credential)
                }
            }
        }
    }
    .sheet(isPresented: $showingAddCredential) {
        AddCredentialSheet()
    }
}

private func validateCredential(_ credential: CredentialSet) {
    isValidating = true

    Task {
        do {
            let account = try await appState.awsService.validateCredentials(credential)

            await MainActor.run {
                if let index = appState.credentials.firstIndex(where: { $0.id == credential
                    appState.credentials[index].accountId = account.accountId
                    appState.credentials[index].isValid = true
                    appState.credentials[index].lastValidatedAt = Date()
                }
                isValidating = false
            }
```

```
                // Save updated credential
                try await appState.credentialService.saveCredential(appState.credentials.first
            } catch {
                await MainActor.run {
                    if let index = appState.credentials.firstIndex(where: { $0.id == credential
                        appState.credentials[index].isValid = false
                    }
                    isValidating = false
                    appState.error = AppError(message: "Credential validation failed", underly
                }
            }
        }
    }

    private func deleteCredential(_ credential: CredentialSet) {
        Task {
            try await appState.credentialService.deleteCredential(id: credential.id)
            await MainActor.run {
                appState.credentials.removeAll { $0.id == credential.id }
                selectedCredential = nil
            }
        }
    }

    private func deleteCredentials(at offsets: IndexSet) {
        for index in offsets {
            let credential = appState.credentials[index]
            deleteCredential(credential)
        }
    }
}

struct CredentialRow: View {
    let credential: CredentialSet
    let isValidating: Bool

    var body: some View {
        HStack {
            VStack(alignment: .leading, spacing: 4) {
                Text(credential.name)
                    .font(.headline)
                HStack {
                    Text(credential.accessKeyId)
                    Text("Ã¢â¬Â¢")
                    Text(credential.region)
                }
                .font(.caption)
                .foregroundColor(.secondary)
```

```
            }

            Spacer()

            if isValidating {
                ProgressView()
                    .scaleEffect(0.7)
            } else if let isValid = credential.isValid {
                Image(systemName: isValid ? "checkmark.circle.fill" : "xmark.circle.fill")
                    .foregroundColor(isValid ? .green : .red)
            }

            Text(credential.environment.rawValue)
                .font(.caption)
                .padding(.horizontal, 8)
                .padding(.vertical, 4)
                .background(Color.accentColor.opacity(0.1))
                .cornerRadius(4)
        }
        .padding(.vertical, 4)
    }
}

struct AddCredentialSheet: View {
    @EnvironmentObject var appState: AppState
    @Environment(\.dismiss) var dismiss

    @State private var name = ""
    @State private var accessKeyId = ""
    @State private var secretAccessKey = ""
    @State private var region = "us-east-1"
    @State private var environment: CredentialEnvironment = .dev
    @State private var isSaving = false

    var body: some View {
        VStack(spacing: 20) {
            Text("Add AWS Credential")
                .font(.title2)
                .fontWeight(.semibold)

            Form {
                TextField("Name (e.g., 'Production AWS')", text: $name)
                TextField("Access Key ID", text: $accessKeyId)
                SecureField("Secret Access Key", text: $secretAccessKey)

                Picker("Region", selection: $region) {
                    Text("US East (N. Virginia)").tag("us-east-1")
                    Text("US West (Oregon)").tag("us-west-2")
```

```
                Text("Europe (Ireland)").tag("eu-west-1")
                Text("Europe (Frankfurt)").tag("eu-central-1")
                Text("Asia Pacific (Tokyo)").tag("ap-northeast-1")
                Text("Asia Pacific (Singapore)").tag("ap-southeast-1")
            }

            Picker("Environment", selection: $environment) {
                ForEach(CredentialEnvironment.allCases, id: \.self) { env in
                    Text(env.rawValue).tag(env)
                }
            }
        }
        .formStyle(.grouped)

        HStack {
            Button("Cancel") { dismiss() }
                .buttonStyle(.bordered)

            Button("Add Credential") {
                saveCredential()
            }
            .buttonStyle(.borderedProminent)
            .disabled(!isValid || isSaving)
        }
    }
    .padding()
    .frame(width: 450)
}

private var isValid: Bool {
    !name.isEmpty && !accessKeyId.isEmpty && !secretAccessKey.isEmpty
}

private func saveCredential() {
    isSaving = true

    let credential = CredentialSet(
        id: UUID().uuidString,
        name: name,
        accessKeyId: accessKeyId,
        secretAccessKey: secretAccessKey,
        region: region,
        accountId: nil,
        environment: environment,
        createdAt: Date(),
        lastValidatedAt: nil,
        isValid: nil
    )
```

```
            Task {
                try await appState.credentialService.saveCredential(credential)

                await MainActor.run {
                    appState.credentials.append(credential)
                    isSaving = false
                    dismiss()
                }
            }
        }
    }

struct GeneralSettingsView: View {
    @AppStorage("autoValidateCredentials") private var autoValidate = true
    @AppStorage("showAdvancedOptions") private var showAdvanced = false
    @AppStorage("defaultTier") private var defaultTier = 1

    var body: some View {
        Form {
            Section {
                Toggle("Auto-validate credentials on launch", isOn: $autoValidate)
                Toggle("Show advanced deployment options", isOn: $showAdvanced)
            }

            Section {
                Picker("Default Infrastructure Tier", selection: $defaultTier) {
                    Text("1 - SEED").tag(1)
                    Text("2 - STARTUP").tag(2)
                    Text("3 - GROWTH").tag(3)
                    Text("4 - SCALE").tag(4)
                    Text("5 - ENTERPRISE").tag(5)
                }
            }
        }
        .formStyle(.grouped)
    }
}

struct AboutView: View {
    var body: some View {
        VStack(spacing: 20) {
            Image(systemName: "cloud.fill")
                .font(.system(size: 80))
                .foregroundColor(.accentColor)

            Text("RADIANT Deployer")
                .font(.title)
```

```
                .fontWeight(.bold)

            Text("Version 2.2.0")
                .foregroundColor(.secondary)

            Text("Production-grade multi-tenant AWS SaaS infrastructure deployment and manageme
                .multilineTextAlignment(.center)
                .foregroundColor(.secondary)

            Divider()

            VStack(alignment: .leading, spacing: 8) {
                Text("Features:")
                    .font(.headline)

                FeatureRow(icon: "server.rack", text: "Multi-environment deployment (Dev/Stagi
                FeatureRow(icon: "cpu", text: "20+ AI providers, 30+ self-hosted models")
                FeatureRow(icon: "lock.shield", text: "HIPAA & SOC 2 compliance")
                FeatureRow(icon: "person.2", text: "Two-person approval for production")
                FeatureRow(icon: "globe", text: "Multi-region global deployment")
            }

            Spacer()

            Text("Ã¸Â© 2024 Zynapses Inc.")
                .font(.caption)
                .foregroundColor(.secondary)
        }
        .padding()
    }
}

struct FeatureRow: View {
    let icon: String
    let text: String

    var body: some View {
        HStack {
            Image(systemName: icon)
                .foregroundColor(.accentColor)
                .frame(width: 20)
            Text(text)
        }
    }
}
```

RadiantDeployer/Views/ProvidersView.swift

```
import SwiftUI

struct ProvidersView: View {
    @EnvironmentObject var appState: AppState

    var body: some View {
        VStack {
            Text("AI Providers")
                .font(.title)
            Text("Provider management is available in the Admin Dashboard after deployment.")
                .foregroundColor(.secondary)
        }
        .frame(maxWidth: .infinity, maxHeight: .infinity)
        .navigationTitle("Providers")
    }
}
```

RadiantDeployer/Views/ModelsView.swift

```
import SwiftUI

struct ModelsView: View {
    @EnvironmentObject var appState: AppState

    var body: some View {
        VStack {
            Text("AI Models")
                .font(.title)
            Text("Model management is available in the Admin Dashboard after deployment.")
                .foregroundColor(.secondary)
        }
        .frame(maxWidth: .infinity, maxHeight: .infinity)
        .navigationTitle("Models")
    }
}
```

PART 5: BUNDLE SCRIPTS

tools/scripts/bundle-infrastructure.sh

```
#!/bin/bash
set -e

echo "=== RADIANT Infrastructure Bundler ==="

# Paths
```

```
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
ROOT_DIR="$(cd "$SCRIPT_DIR/../.." && pwd)"
INFRA_SOURCE="$ROOT_DIR/packages/infrastructure"
SWIFT_APP="$ROOT_DIR/apps/swift-deployer"
BUNDLE_TARGET="$SWIFT_APP/Resources/Infrastructure"

echo "Source: $INFRA_SOURCE"
echo "Target: $BUNDLE_TARGET"

# Clean and create target
rm -rf "$BUNDLE_TARGET"
mkdir -p "$BUNDLE_TARGET"

# Copy CDK source
echo "Copying CDK source..."
cp -R "$INFRA_SOURCE/bin" "$BUNDLE_TARGET/"
cp -R "$INFRA_SOURCE/lib" "$BUNDLE_TARGET/"
cp -R "$INFRA_SOURCE/lambda" "$BUNDLE_TARGET/"
cp "$INFRA_SOURCE/package.json" "$BUNDLE_TARGET/"
cp "$INFRA_SOURCE/package-lock.json" "$BUNDLE_TARGET/" 2>/dev/null || true
cp "$INFRA_SOURCE/cdk.json" "$BUNDLE_TARGET/"
cp "$INFRA_SOURCE/tsconfig.json" "$BUNDLE_TARGET/"

# Copy migrations
if [ -d "$ROOT_DIR/migrations" ]; then
    echo "Copying migrations..."
    cp -R "$ROOT_DIR/migrations" "$BUNDLE_TARGET/"
fi

# Install production dependencies
echo "Installing production dependencies..."
cd "$BUNDLE_TARGET"
npm ci --production

echo "=== Bundle complete ==="
echo "Size: $(du -sh "$BUNDLE_TARGET" | cut -f1)"
```

tools/scripts/bundle-node.sh

```
#!/bin/bash
set -e

echo "=== Node.js Runtime Bundler ==="

# Paths
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
SWIFT_APP="$(cd "$SCRIPT_DIR/../../apps/swift-deployer" && pwd)"
NODE_TARGET="$SWIFT_APP/Resources/NodeRuntime"
```

```
# Node version
NODE_VERSION="20.10.0"

echo "Target: $NODE_TARGET"
echo "Node Version: $NODE_VERSION"

# Clean and create target
rm -rf "$NODE_TARGET"
mkdir -p "$NODE_TARGET"

# Download Node.js for macOS ARM64
ARCH=$(uname -m)
if [ "$ARCH" = "arm64" ]; then
    NODE_ARCH="arm64"
else
    NODE_ARCH="x64"
fi

NODE_URL="https://nodejs.org/dist/v${NODE_VERSION}/node-v${NODE_VERSION}-darwin-${NODE_ARCH}.ta
TEMP_DIR=$(mktemp -d)

echo "Downloading Node.js..."
curl -sL "$NODE_URL" | tar xz -C "$TEMP_DIR"

# Copy only necessary files
echo "Copying runtime..."
cp -R "$TEMP_DIR/node-v${NODE_VERSION}-darwin-${NODE_ARCH}/bin" "$NODE_TARGET/"
cp -R "$TEMP_DIR/node-v${NODE_VERSION}-darwin-${NODE_ARCH}/lib" "$NODE_TARGET/"

# Cleanup
rm -rf "$TEMP_DIR"

echo "=== Bundle complete ==="
echo "Size: $(du -sh "$NODE_TARGET" | cut -f1)"
```

BUILD INSTRUCTIONS

1. **Build Shared Package**:

   ```
   cd packages/shared
   npm install
   npm run build
   ```

2. **Bundle Node.js Runtime**:

   ```
   chmod +x tools/scripts/bundle-node.sh
   ./tools/scripts/bundle-node.sh
   ```

3. **Bundle Infrastructure** (after completing Prompts 2-3):

```
chmod +x tools/scripts/bundle-infrastructure.sh
./tools/scripts/bundle-infrastructure.sh
```

4. **Open in Xcode**:

```
cd apps/swift-deployer
open RadiantDeployer.xcodeproj
```

5. **Build and Run**:
    - Select "My Mac" as the destination
    - Press Ã¢Å’ËœR to build and run

DEPENDENCIES

Swift (Add to Xcode Project)

- SQLite3 (System framework)
- Security (System framework)

Node.js (bundled)

- aws-cdk: ^2.120.0
- aws-cdk-lib: ^2.120.0
- constructs: ^10.3.0

NEXT PROMPTS

Continue with: - **Prompt 2**: CDK Infrastructure Stacks (VPC, Database, Security, Storage) - **Prompt 3**: CDK AI & API Stacks (LiteLLM, SageMaker, API Gateway) - **Prompt 4**: Lambda Functions - Core - **Prompt 5**: Lambda Functions - Admin & Billing - **Prompt 6**: Self-Hosted Models & Mid-Level Services - **Prompt 7**: External Providers & Database Schema - **Prompt 8**: Admin Web Dashboard - **Prompt 9**: Assembly & Deployment Guide

---

End of Prompt 1: Foundation & Swift Deployment App RADIANT v2.2.0 - December 2024

â• â

END OF SECTION 1

â• â

â• â