

Contents

SECTION 3: CDK AI & API STACKS (v2.1.0) 1

[illegible]RADIANT v2.2.0 - Prompt 3: CDK AI & API Stacks 2

OVERVIEW 2

ARCHITECTURE OVERVIEW 2

STACK DEPENDENCIES 3

PART 1: UPDATE CDK ENTRY POINT 4

packages/infrastructure/bin/radiant.ts (Complete)	4
---	---

PART 2: AUTH STACK (COGNITO) 9

packages/infrastructure/lib/stacks/auth.stack.ts	9
--	---

PART 3: AI STACK (LITELLM + SAGEMAKER) 19

packages/infrastructure/lib/stacks/ai.stack.ts	19
--	----

PART 4: API STACK (REST + GRAPHQL) 31

packages/infrastructure/lib/stacks/api.stack.ts	31
---	----

PART 5: ADMIN STACK 41

[packages/infrastructure/lib/stacks/admin.stack.ts](#) 41

PART 6: GraphQL SCHEMA	51
------------------------	----

graphql/schema.graphql	51
--	----

PART 7: UPDATE STACK EXPORTS 58

packages/infrastructure/lib/stacks/index.ts (Updated)	58
---	----

DEPLOYMENT COMMANDS 59

Deploy All Stacks	59
-----------------------------	----

Deploy Individual Stacks	59
------------------------------------	----

Verify Deployment	60
-----------------------------	----

ESTIMATED COSTS BY TIER 60

NEXT PROMPTS 61

[illegible]

END OF SECTION 3 **61**

[illegible][illegible]

SECTION 3: CDK AI & API STACKS (v2.1.0)

[illegible]

Dependencies: Sections 0-2 **Creates:** Cognito, LiteLLM, API Gateway, AppSync, Admin stacks

Type Imports

```
import {
  TierConfig,
  getTierConfig,
  EXTERNAL_PROVIDERS,
  PROVIDER_ENDPOINTS,
} from '@radiant/shared';
```

RADIANT v2.2.0 - Prompt 3: CDK AI & API Stacks

Prompt 3 of 9 | Target Size: ~50KB | Version: 3.7.0 | December 2024

OVERVIEW

This prompt creates the AI services and API layer stacks:

1. **Auth Stack** - Cognito User Pool, Identity Pool, MFA configuration
2. **AI Stack** - LiteLLM on ECS Fargate, SageMaker endpoints for self-hosted models
3. **API Stack** - API Gateway REST API, AppSync GraphQL, Lambda integrations
4. **Admin Stack** - Admin dashboard hosting, admin-specific APIs

All stacks are tier-aware with automatic scaling based on infrastructure tier (1-5).

ARCHITECTURE OVERVIEW

Architectural Diagram illustrating the components and layers of the RADIANT v2.2.0 system:

```
graph TD
    subgraph CLIENT_LAYER [CLIENT LAYER]
        direction LR
        W[Web Apps]
        M[Mobile Apps]
        D[Desktop Apps]
        A[Admin Dashboard]
    end

    subgraph CLOUDFRONT [CLOUDFRONT (CDN)]
        direction LR
        T[TLS 1.3]
        G[Geographic Routing]
        C[Caching]
        P[DDoS Protection]
    end

    subgraph API_STACK [API STACK]
        direction LR
        S[Admin SPA]
        R[REST API GW]
        A2[AppSync]
    end

    subgraph AI_STACK [AI STACK]
        direction LR
        S2[S3/CF]
        A3[API/v2/*]
        G2[GraphQL]
    end

    CLIENT_LAYER --> CLOUDFRONT
    CLOUDFRONT --> API_STACK
    API_STACK --> AI_STACK
```

The diagram shows a multi-tier architecture. The **CLIENT LAYER** (Web Apps, Mobile Apps, Desktop Apps, Admin Dashboard) interacts with the **CLOUDFRONT (CDN)** layer, which includes TLS 1.3, Geographic Routing, Caching, and DDoS Protection. The **API STACK** (Admin SPA, REST API GW, AppSync) and **AI STACK** (S3/CF, API/v2/*, GraphQL) are also shown, with the API Stack interacting with the AI Stack.


```

// =====
// STACK PREFIX
// =====

const stackPrefix = `${appId}-${environment}`;

const env = {
  account: process.env.CDK_DEFAULT_ACCOUNT,
  region: primaryRegion
};

const commonProps = {
  appId,
  appName,
  environment,
  tier,
  tierConfig,
  domain,
};

// =====
// FOUNDATION STACKS (From Prompt 2)
// =====

// 1. Foundation Stack
const foundationStack = new FoundationStack(app, `${stackPrefix}-foundation`, {
  env,
  ...commonProps,
  description: `RADIANT Foundation - ${appName} ${environment}`,
});

// 2. Networking Stack
const networkingStack = new NetworkingStack(app, `${stackPrefix}-networking`, {
  env,
  ...commonProps,
  description: `RADIANT Networking - ${appName} ${environment}`,
});
networkingStack.addDependency(foundationStack);

// 3. Security Stack
const securityStack = new SecurityStack(app, `${stackPrefix}-security`, {
  env,
  ...commonProps,
  vpc: networkingStack.vpc,
  description: `RADIANT Security - ${appName} ${environment}`,
});
securityStack.addDependency(networkingStack);

```

```

// 4. Data Stack
const dataStack = new DataStack(app, `${stackPrefix}-data`, {
  env,
  ...commonProps,
  vpc: networkingStack.vpc,
  databaseSecurityGroup: securityStack.databaseSecurityGroup,
  lambdaSecurityGroup: securityStack.lambdaSecurityGroup,
  dataKey: securityStack.dataKey,
  isPrimary: true,
  enableGlobal: enableMultiRegion,
  description: `RADIANT Data Layer - ${appName} ${environment}`,
});
dataStack.addDependency(securityStack);

// 5. Storage Stack
const storageStack = new StorageStack(app, `${stackPrefix}-storage`, {
  env,
  ...commonProps,
  vpc: networkingStack.vpc,
  mediaKey: securityStack.mediaKey,
  hostedZoneId,
  enableReplication: enableMultiRegion,
  description: `RADIANT Storage - ${appName} ${environment}`,
});
storageStack.addDependency(securityStack);

// =====
// AI & API STACKS (Prompt 3)
// =====

// 6. Auth Stack (Cognito)
const authStack = new AuthStack(app, `${stackPrefix}-auth`, {
  env,
  ...commonProps,
  description: `RADIANT Auth - ${appName} ${environment}`,
});
authStack.addDependency(securityStack);

// 7. AI Stack (LiteLLM + SageMaker)
const aiStack = new AIStack(app, `${stackPrefix}-ai`, {
  env,
  ...commonProps,
  vpc: networkingStack.vpc,
  ecsSecurityGroup: securityStack.ecsSecurityGroup,
  lambdaSecurityGroup: securityStack.lambdaSecurityGroup,
  albSecurityGroup: securityStack.albSecurityGroup,
  secretsKey: securityStack.secretsKey,
  description: `RADIANT AI Services - ${appName} ${environment}`,
});

```

```

});
aiStack.addDependency(securityStack);
aiStack.addDependency(networkingStack);

// 8. API Stack (REST + GraphQL)
const apiStack = new APIStack(app, `${stackPrefix}-api`, {
  env,
  ...commonProps,
  vpc: networkingStack.vpc,
  lambdaSecurityGroup: securityStack.lambdaSecurityGroup,
  userPool: authStack.userPool,
  userPoolClient: authStack.userPoolClient,
  litellmUrl: aiStack.litellmUrl,
  auroraCluster: dataStack.cluster,
  auroraSecret: dataStack.dbSecret,
  usageTable: dataStack.usageTable,
  sessionsTable: dataStack.sessionsTable,
  cacheTable: dataStack.cacheTable,
  mediaBucket: storageStack.mediaBucket,
  lambdaRole: securityStack.lambdaExecutionRole,
  webAcl: securityStack.webAcl,
  description: `RADIANT API Layer - ${appName} ${environment}`,
});
apiStack.addDependency(authStack);
apiStack.addDependency(aiStack);
apiStack.addDependency(dataStack);
apiStack.addDependency(storageStack);

// 9. Admin Stack
const adminStack = new AdminStack(app, `${stackPrefix}-admin`, {
  env,
  ...commonProps,
  vpc: networkingStack.vpc,
  adminUserPool: authStack.adminUserPool,
  adminUserPoolClient: authStack.adminUserPoolClient,
  restApi: apiStack.api,
  graphqlApi: apiStack.graphqlApi,
  assetsBucket: storageStack.assetsBucket,
  assetsDistribution: storageStack.assetsDistribution,
  hostedZoneId,
  description: `RADIANT Admin Dashboard - ${appName} ${environment}`,
});
adminStack.addDependency(apiStack);

// =====
// TAGGING
// =====

```

```

cdk.Tags.of(app).add('Project', 'RADIANT');
cdk.Tags.of(app).add('AppId', appId);
cdk.Tags.of(app).add('Environment', environment);
cdk.Tags.of(app).add('Tier', String(tier));
cdk.Tags.of(app).add('ManagedBy', 'CDK');
cdk.Tags.of(app).add('Version', '2.2.0');

```

PART 2: AUTH STACK (COGNITO)

packages/infrastructure/lib/stacks/auth.stack.ts

```

import * as cdk from 'aws-cdk-lib';
import * as cognito from 'aws-cdk-lib/aws-cognito';
import * as iam from 'aws-cdk-lib/aws-iam';
import * as ssm from 'aws-cdk-lib/aws-ssm';
import { Construct } from 'constructs';
import { TierConfig, TierLevel } from '../config/tiers';
import { applyTags } from '../config/tags';

export interface AuthStackProps extends cdk.StackProps {
  appId: string;
  appName: string;
  environment: string;
  tier: TierLevel;
  tierConfig: TierConfig;
  domain: string;
}

/**
 * Auth Stack
 *
 * Creates Cognito User Pools for both end-users and administrators.
 * Configures MFA, password policies, and Identity Pool for AWS access.
 */
export class AuthStack extends cdk.Stack {
  // User Pool (End Users)
  public readonly userPool: cognito.UserPool;
  public readonly userPoolClient: cognito.UserPoolClient;
  public readonly userPoolDomain: cognito.UserPoolDomain;

  // Admin User Pool (Administrators)
  public readonly adminUserPool: cognito.UserPool;
  public readonly adminUserPoolClient: cognito.UserPoolClient;
  public readonly adminUserPoolDomain: cognito.UserPoolDomain;

  // Identity Pool
  public readonly identityPool: cognito.CfnIdentityPool;

```

```

// Domain
public readonly cognitoDomain: string;

constructor(scope: Construct, id: string, props: AuthStackProps) {
    super(scope, id, props);

    const { tierConfig } = props;
    const resourcePrefix = `${props.appId}-${props.environment}`;

    // =====
    // END-USER COGNITO USER POOL
    // =====

    this.userPool = new cognito.UserPool(this, 'UserPool', {
        userPoolName: `${resourcePrefix}-users`,

        // Sign-in configuration
        signInAliases: {
            email: true,
            username: false,
        },

        // Self-registration
        selfSignUpEnabled: true,

        // Verification
        autoVerify: {
            email: true,
        },

        // User verification
        userVerification: {
            emailSubject: `Welcome to ${props.appName} - Verify your email`,
            emailBody: `
                <h2>Welcome to ${props.appName}!</h2>
                <p>Thank you for signing up. Please verify your email address by entering this code:
                <h1 style="color: #4F46E5; letter-spacing: 4px;">{####}</h1>
                <p>This code expires in 24 hours.</p>
                <p>If you didn't create this account, please ignore this email.</p>
            `,
            emailStyle: cognito.VerificationEmailStyle.CODE,
        },

        // Password policy
        passwordPolicy: {
            minLength: 12,
            requireLowercase: true,

```

```

    requireUppercase: true,
    requireDigits: true,
    requireSymbols: true,
    tempPasswordValidity: cdk.Duration.days(7),
},

// MFA configuration
mfa: tierConfig.tier >= 3
  ? cognito.Mfa.OPTIONAL
  : cognito.Mfa.OFF,
mfaSecondFactor: {
  sms: true,
  otp: true,
},

// Account recovery
accountRecovery: cognito.AccountRecovery.EMAIL_ONLY,

// Standard attributes
standardAttributes: {
  email: {
    required: true,
    mutable: true,
  },
  fullname: {
    required: false,
    mutable: true,
  },
},

// Custom attributes
customAttributes: {
  tenantId: new cognito.StringAttribute({ mutable: false }),
  role: new cognito.StringAttribute({ mutable: true }),
  createdAt: new cognito.StringAttribute({ mutable: false }),
},

// Device tracking
deviceTracking: {
  challengeRequiredOnNewDevice: tierConfig.tier >= 3,
  deviceOnlyRememberedOnUserPrompt: true,
},

// Advanced security (Tier 3+)
advancedSecurityMode: tierConfig.tier >= 3
  ? cognito.AdvancedSecurityMode.ENFORCED
  : cognito.AdvancedSecurityMode.OFF,

```

```

// Removal policy
removalPolicy: props.environment === 'prod'
  ? cdk RemovalPolicy.RETAIN
  : cdk RemovalPolicy.DESTROY,
});

// User Pool Client
this.userPoolClient = this.userPool.addClient('UserPoolClient', {
  userPoolClientName: `${resourcePrefix}-web-client`,

  // OAuth configuration
  oAuth: {
    flows: {
      authorizationCodeGrant: true,
      implicitCodeGrant: false,
    },
    scopes: [
      cognito.OAuthScope.EMAIL,
      cognito.OAuthScope.OPENID,
      cognito.OAuthScope.PROFILE,
    ],
    callbackUrls: [
      `https://${props.domain}/auth/callback`,
      `https://admin.${props.domain}/auth/callback`,
      'http://localhost:3000/auth/callback',
    ],
    logoutUrls: [
      `https://${props.domain}/auth/logout`,
      `https://admin.${props.domain}/auth/logout`,
      'http://localhost:3000/auth/logout',
    ],
  },

  // Token configuration
  accessTokenValidity: cdk.Duration.hours(1),
  idTokenValidity: cdk.Duration.hours(1),
  refreshTokenValidity: cdk.Duration.days(30),

  // Auth flows
  authFlows: {
    userPassword: true,
    userSrp: true,
    custom: false,
  },

  // Prevent user existence errors
  preventUserExistenceErrors: true,

```

```

    // Generate client secret for server-side apps
    generateSecret: false,
  });

  // User Pool Domain
  const domainPrefix = `${props.appId}-${props.environment}-${this.account}`.toLowerCase();
  this.userPoolDomain = this.userPool.addDomain('UserPoolDomain', {
    cognitoDomain: {
      domainPrefix,
    },
  });
  this.cognitoDomain = `${domainPrefix}.auth.${this.region}.amazoncognito.com`;

  // =====
  // ADMIN COGNITO USER POOL (Separate for enhanced security)
  // =====

  this.adminUserPool = new cognito.UserPool(this, 'AdminUserPool', {
    userPoolName: `${resourcePrefix}-admins`,

    // Sign-in
    signInAliases: {
      email: true,
      username: false,
    },

    // No self-registration for admins
    selfSignUpEnabled: false,

    // Verification
    autoVerify: {
      email: true,
    },

    // Invitation settings
    userInvitation: {
      emailSubject: `You're invited to ${props.appName} Admin`,
      emailBody: `
        <h2>Welcome to ${props.appName} Admin!</h2>
        <p>You have been invited to join as an administrator.</p>
        <p>Your temporary password is: <strong>{####}</strong></p>
        <p>Please sign in at <a href="https://admin.${props.domain}">admin.${props.domain}</a>
        <p>This invitation expires in 7 days.</p>
      `,
    },
  });

  // Strict password policy for admins
  passwordPolicy: {

```

```

    minLength: 16,
    requireLowercase: true,
    requireUppercase: true,
    requireDigits: true,
    requireSymbols: true,
    tempPasswordValidity: cdk.Duration.days(7),
},

// MFA REQUIRED for production admins
mfa: props.environment === 'prod'
    ? cognito.Mfa.REQUIRED
    : cognito.Mfa.OPTIONAL,
mfaSecondFactor: {
    sms: false, // TOTP only for admins
    otp: true,
},

// Account recovery
accountRecovery: cognito.AccountRecovery.EMAIL_ONLY,

// Standard attributes
standardAttributes: {
    email: {
        required: true,
        mutable: true,
    },
    fullname: {
        required: true,
        mutable: true,
    },
},

// Custom attributes
customAttributes: {
    adminRole: new cognito.StringAttribute({ mutable: true }),
    invitedBy: new cognito.StringAttribute({ mutable: false }),
    invitedAt: new cognito.StringAttribute({ mutable: false }),
},

// Always track admin devices
deviceTracking: {
    challengeRequiredOnNewDevice: true,
    deviceOnlyRememberedOnUserPrompt: true,
},

// Always enforce advanced security for admins
advancedSecurityMode: cognito.AdvancedSecurityMode.ENFORCED,

```

```

    // Never delete admin pool
    removalPolicy: cdk.RemovalPolicy.RETAIN,
  });

  // Admin User Pool Groups
  new cognito.CfnUserPoolGroup(this, 'SuperAdminsGroup', {
    userPoolId: this.adminUserPool.userPoolId,
    groupName: 'super_admin',
    description: 'Full platform access including user management',
    precedence: 1,
  });

  new cognito.CfnUserPoolGroup(this, 'AdminsGroup', {
    userPoolId: this.adminUserPool.userPoolId,
    groupName: 'admin',
    description: 'Can deploy and manage assigned applications',
    precedence: 2,
  });

  new cognito.CfnUserPoolGroup(this, 'OperatorsGroup', {
    userPoolId: this.adminUserPool.userPoolId,
    groupName: 'operator',
    description: 'Read-only access with audit log viewing',
    precedence: 3,
  });

  new cognito.CfnUserPoolGroup(this, 'AuditorsGroup', {
    userPoolId: this.adminUserPool.userPoolId,
    groupName: 'auditor',
    description: 'Billing and audit log access only',
    precedence: 4,
  });

  // Admin User Pool Client
  this.adminUserPoolClient = this.adminUserPool.addClient('AdminUserPoolClient', {
    userPoolClientName: `${resourcePrefix}-admin-client`,

    oAuth: {
      flows: {
        authorizationCodeGrant: true,
        implicitCodeGrant: false,
      },
      scopes: [
        cognito.OAuthScope.EMAIL,
        cognito.OAuthScope.OPENID,
        cognito.OAuthScope.PROFILE,
      ],
      callbackUrls: [

```

```

        `https://admin.${props.domain}/auth/callback`,
        'http://localhost:3001/auth/callback',
    ],
    logoutUrls: [
        `https://admin.${props.domain}/auth/logout`,
        'http://localhost:3001/auth/logout',
    ],
},

// Shorter token validity for admins
accessTokenValidity: cdk.Duration.minutes(30),
idTokenValidity: cdk.Duration.minutes(30),
refreshTokenValidity: cdk.Duration.days(7),

authFlows: {
    userPassword: false, // SRP only for admins
    userSrp: true,
    custom: false,
},

preventUserExistenceErrors: true,
generateSecret: false,
});

// Admin User Pool Domain
const adminDomainPrefix = `${props.appId}-${props.environment}-admin-${this.account}`.toLowerCase();
this.adminUserPoolDomain = this.adminUserPool.addDomain('AdminUserPoolDomain', {
    cognitoDomain: {
        domainPrefix: adminDomainPrefix,
    },
});

// =====
// IDENTITY POOL (Federated Identities)
// =====

this.identityPool = new cognito.CfnIdentityPool(this, 'IdentityPool', {
    identityPoolName: `${resourcePrefix.replace(/-/g, '_')}_identity`,

    allowUnauthenticatedIdentities: false,

    cognitoIdentityProviders: [
        {
            clientId: this.userPoolClient.userPoolClientId,
            providerName: this.userPool.userPoolProviderName,
            serverSideTokenCheck: true,
        },
    ],
},

```

```

});

// Authenticated Role
const authenticatedRole = new iam.Role(this, 'AuthenticatedRole', {
  assumedBy: new iam.FederatedPrincipal(
    'cognito-identity.amazonaws.com',
    {
      StringEquals: {
        'cognito-identity.amazonaws.com:aud': this.identityPool.ref,
      },
      'ForAnyValue:StringLike': {
        'cognito-identity.amazonaws.com:amr': 'authenticated',
      },
    },
    'sts:AssumeRoleWithWebIdentity'
  ),
  description: 'Role for authenticated users',
});

// Basic permissions for authenticated users
authenticatedRole.addToPolicy(new iam.PolicyStatement({
  actions: [
    's3:GetObject',
    's3:PutObject',
  ],
  resources: [
    `arn:aws:s3:::${props.appId}-${props.environment}-media-*/*`,
  ],
  conditions: {
    StringLike: {
      's3:prefix': ['uploads/${cognito-identity.amazonaws.com:sub}/*'],
    },
  },
}));

// Attach roles to Identity Pool
new cognito.CfnIdentityPoolRoleAttachment(this, 'IdentityPoolRoles', {
  identityPoolId: this.identityPool.ref,
  roles: {
    authenticated: authenticatedRole.roleArn,
  },
});

// =====
// SSM PARAMETERS
// =====

new ssm.StringParameter(this, 'UserPoolIdParam', {

```

```

        parameterName: `/radiant/${props.appId}/${props.environment}/auth/user-pool-id`,
        stringValue: this.userPool.userPoolId,
    });

    new ssm.StringParameter(this, 'UserPoolClientIdParam', {
        parameterName: `/radiant/${props.appId}/${props.environment}/auth/user-pool-client-id`,
        stringValue: this.userPoolClient.userPoolClientId,
    });

    new ssm.StringParameter(this, 'AdminUserPoolIdParam', {
        parameterName: `/radiant/${props.appId}/${props.environment}/auth/admin-user-pool-id`,
        stringValue: this.adminUserPool.userPoolId,
    });

    new ssm.StringParameter(this, 'AdminUserPoolClientIdParam', {
        parameterName: `/radiant/${props.appId}/${props.environment}/auth/admin-user-pool-client-id`,
        stringValue: this.adminUserPoolClient.userPoolClientId,
    });

    new ssm.StringParameter(this, 'CognitoDomainParam', {
        parameterName: `/radiant/${props.appId}/${props.environment}/auth/cognito-domain`,
        stringValue: this.cognitoDomain,
    });

    new ssm.StringParameter(this, 'IdentityPoolIdParam', {
        parameterName: `/radiant/${props.appId}/${props.environment}/auth/identity-pool-id`,
        stringValue: this.identityPool.ref,
    });

    // =====
    // TAGS
    // =====

    applyTags(this, {
        appId: props.appId,
        environment: props.environment,
        tier: props.tier,
    });

    // =====
    // OUTPUTS
    // =====

    new cdk.CfnOutput(this, 'UserPoolId', {
        value: this.userPool.userPoolId,
        description: 'Cognito User Pool ID',
        exportName: `${resourcePrefix}-user-pool-id`,
    });

```

```

new cdk.CfnOutput(this, 'UserPoolClientId', {
  value: this.userPoolClient.userPoolClientId,
  description: 'Cognito User Pool Client ID',
  exportName: `${resourcePrefix}-user-pool-client-id`,
});

new cdk.CfnOutput(this, 'CognitoDomainOutput', {
  value: this.cognitoDomain,
  description: 'Cognito Domain',
  exportName: `${resourcePrefix}-cognito-domain`,
});

new cdk.CfnOutput(this, 'IdentityPoolId', {
  value: this.identityPool.ref,
  description: 'Cognito Identity Pool ID',
  exportName: `${resourcePrefix}-identity-pool-id`,
});

new cdk.CfnOutput(this, 'AdminUserPoolId', {
  value: this.adminUserPool.userPoolId,
  description: 'Admin Cognito User Pool ID',
  exportName: `${resourcePrefix}-admin-user-pool-id`,
});

new cdk.CfnOutput(this, 'AdminUserPoolClientId', {
  value: this.adminUserPoolClient.userPoolClientId,
  description: 'Admin Cognito User Pool Client ID',
  exportName: `${resourcePrefix}-admin-user-pool-client-id`,
});
}
}

```

PART 3: AI STACK (LITELLM + SAGEMAKER)

packages/infrastructure/lib/stacks/ai.stack.ts

```

import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as ecs from 'aws-cdk-lib/aws-ecs';
import * as ecsPatterns from 'aws-cdk-lib/aws-ecs-patterns';
import * as logs from 'aws-cdk-lib/aws-logs';
import * as secretsmanager from 'aws-cdk-lib/aws-secretsmanager';
import * as iam from 'aws-cdk-lib/aws-iam';
import * as kms from 'aws-cdk-lib/aws-kms';
import * as ssm from 'aws-cdk-lib/aws-ssm';
import * as sagemaker from 'aws-cdk-lib/aws-sagemaker';

```

```

import * as applicationautoscaling from 'aws-cdk-lib/aws-applicationautoscaling';
import * as servicediscovery from 'aws-cdk-lib/aws-servicediscovery';
import { Construct } from 'constructs';
import { TierConfig, TierLevel } from '../config/tiers';
import { applyTags } from '../config/tags';

export interface AISTackProps extends cdk.StackProps {
  appId: string;
  appName: string;
  environment: string;
  tier: TierLevel;
  tierConfig: TierConfig;
  domain: string;
  vpc: ec2.Vpc;
  ecsSecurityGroup: ec2.SecurityGroup;
  lambdaSecurityGroup: ec2.SecurityGroup;
  albSecurityGroup: ec2.SecurityGroup;
  secretsKey: kms.Key;
}

/**
 * AI Stack
 *
 * Creates LiteLLM proxy on ECS Fargate for unified AI routing.
 * Deploys SageMaker endpoints for self-hosted models (Tier 3+).
 * Supports 20+ external providers and 30+ self-hosted models.
 */
export class AISTack extends cdk.Stack {
  // ECS Resources
  public readonly cluster: ec2.Cluster;
  public readonly litellmService: ecsPatterns.ApplicationLoadBalancedFargateService;
  public readonly litellmUrl: string;

  // Service Discovery
  public readonly namespace: servicediscovery.PrivateDnsNamespace;

  // Secrets
  public readonly providerSecretsArn: string;

  // SageMaker (Tier 3+)
  public readonly sagemakerRole?: iam.Role;
  public readonly sagemakerEndpoints: Map<string, sagemaker.CfnEndpoint> = new Map();

  constructor(scope: Construct, id: string, props: AISTackProps) {
    super(scope, id, props);

    const { tierConfig, vpc } = props;
    const resourcePrefix = `${props.appId}-${props.environment}`;

```

```

// =====
// SERVICE DISCOVERY NAMESPACE
// =====

this.namespace = new servicediscovery.PrivateDnsNamespace(this, 'AiNamespace', {
  name: `${resourcePrefix}.ai.internal`,
  vpc,
  description: `Service discovery namespace for ${props.appName} AI services`,
});

// =====
// PROVIDER API KEYS SECRET
// =====

const providerSecrets = new secretsmanager.Secret(this, 'ProviderSecrets', {
  secretName: `${resourcePrefix}/ai/provider-keys`,
  description: `API keys for AI providers - ${props.appName}`,
  encryptionKey: props.secretsKey,
  generateSecretString: {
    secretStringTemplate: JSON.stringify({
      OPENAI_API_KEY: '',
      ANTHROPIC_API_KEY: '',
      GOOGLE_API_KEY: '',
      XAI_API_KEY: '',
      DEEPSEEK_API_KEY: '',
      PERPLEXITY_API_KEY: '',
      COHERE_API_KEY: '',
      MISTRAL_API_KEY: '',
      GROQ_API_KEY: '',
      TOGETHER_API_KEY: '',
      FIREWORKS_API_KEY: '',
      STABILITY_API_KEY: '',
      ELEVENLABS_API_KEY: '',
      RUNWAY_API_KEY: '',
      REPLICATE_API_TOKEN: '',
      MESHY_API_KEY: '',
      AZURE_OPENAI_API_KEY: '',
      AZURE_OPENAI_ENDPOINT: '',
      AWS_BEDROCK_REGION: props.env?.region || 'us-east-1',
    }),
    generateStringKey: 'LITELLM_MASTER_KEY',
    excludeCharacters: '"\'\\',
  },
});
this.providerSecretsArn = providerSecrets.secretArn;

// =====

```

```

// ECS CLUSTER
// =====

this.cluster = new ecs.Cluster(this, 'AiCluster', {
  clusterName: `${resourcePrefix}-ai`,
  vpc,
  containerInsights: tierConfig.tier >= 2,
  enableFargateCapacityProviders: true,
});

// =====
// LITELLM TASK DEFINITION
// =====

const litellmLogGroup = new logs.LogGroup(this, 'LitellmLogs', {
  logGroupName: `/radiant/${props.appId}/${props.environment}/litellm`,
  retention: tierConfig.tier >= 3
    ? logs.RetentionDays.THREE_MONTHS
    : logs.RetentionDays.TWO_WEEKS,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
});

const litellmTaskDef = new ecs.FargateTaskDefinition(this, 'LitellmTaskDef', {
  family: `${resourcePrefix}-litellm`,
  cpu: tierConfig.litellm.cpu,
  memoryLimitMiB: tierConfig.litellm.memory,
});

// Grant secrets access
providerSecrets.grantRead(litellmTaskDef.taskRole);

// LiteLLM Container
const litellmContainer = litellmTaskDef.addContainer('litellm', {
  image: ecs.ContainerImage.fromRegistry('ghcr.io/berriai/litellm:main-latest'),
  essential: true,
  logging: ecs.LogDrivers.awsLogs({
    streamPrefix: 'litellm',
    logGroup: litellmLogGroup,
  }),
  environment: {
    LITELLM_LOG: 'DEBUG',
    DATABASE_URL: '', // Will be set by Lambda or config
    STORE_MODEL_IN_DB: 'true',
    UI_USERNAME: 'admin',
  },
  secrets: {
    LITELLM_MASTER_KEY: ecs.Secret.fromSecretsManager(providerSecrets, 'LITELLM_MASTER_KEY'),
    OPENAI_API_KEY: ecs.Secret.fromSecretsManager(providerSecrets, 'OPENAI_API_KEY'),
  },
});

```

```

    ANTHROPIC_API_KEY: ecs.Secret.fromSecretsManager(providerSecrets, 'ANTHROPIC_API_KEY'),
    GOOGLE_API_KEY: ecs.Secret.fromSecretsManager(providerSecrets, 'GOOGLE_API_KEY'),
    XAI_API_KEY: ecs.Secret.fromSecretsManager(providerSecrets, 'XAI_API_KEY'),
    DEEPSEEK_API_KEY: ecs.Secret.fromSecretsManager(providerSecrets, 'DEEPSEEK_API_KEY'),
    PERPLEXITY_API_KEY: ecs.Secret.fromSecretsManager(providerSecrets, 'PERPLEXITY_API_KEY'),
    COHERE_API_KEY: ecs.Secret.fromSecretsManager(providerSecrets, 'COHERE_API_KEY'),
    MISTRAL_API_KEY: ecs.Secret.fromSecretsManager(providerSecrets, 'MISTRAL_API_KEY'),
    GROQ_API_KEY: ecs.Secret.fromSecretsManager(providerSecrets, 'GROQ_API_KEY'),
    TOGETHER_API_KEY: ecs.Secret.fromSecretsManager(providerSecrets, 'TOGETHER_API_KEY'),
    FIREWORKS_API_KEY: ecs.Secret.fromSecretsManager(providerSecrets, 'FIREWORKS_API_KEY')
  },
  healthCheck: {
    command: ['CMD-SHELL', 'curl -f http://localhost:4000/health || exit 1'],
    interval: cdk.Duration.seconds(30),
    timeout: cdk.Duration.seconds(5),
    retries: 3,
    startPeriod: cdk.Duration.seconds(60),
  },
});

litellmContainer.addPortMappings({
  containerPort: 4000,
  protocol: ecs.Protocol.TCP,
});

// =====
// LITELLM FARGATE SERVICE WITH ALB
// =====

this.litellmService = new ecsPatterns.ApplicationLoadBalancedFargateService(this, 'Litellm', {
  cluster: this.cluster,
  serviceName: `${resourcePrefix}-litellm`,
  taskDefinition: litellmTaskDef,

  // Capacity
  desiredCount: tierConfig.litellm.taskCount,

  // Networking
  assignPublicIp: false,
  securityGroups: [props.ecsSecurityGroup],
  taskSubnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },

  // Load Balancer
  publicLoadBalancer: false,
  loadBalancerName: `${resourcePrefix}-litellm-alb`,

  // Service discovery
  cloudMapOptions: {

```

```

    name: 'litellm',
    cloudMapNamespace: this.namespace,
    dnsRecordType: servicediscovery.DnsRecordType.A,
  },

  // Health check
  healthCheckGracePeriod: cdk.Duration.seconds(120),

  // Capacity providers
  capacityProviderStrategies: [
    {
      capacityProvider: 'FARGATE',
      weight: 1,
      base: 1,
    },
    {
      capacityProvider: 'FARGATE_SPOT',
      weight: tierConfig.tier >= 3 ? 2 : 0,
    },
  ],
});

// Configure health check
this.litellmService.targetGroup.configureHealthCheck({
  path: '/health',
  healthyHttpCodes: '200',
  healthyThresholdCount: 2,
  unhealthyThresholdCount: 3,
  interval: cdk.Duration.seconds(30),
  timeout: cdk.Duration.seconds(10),
});

// Internal URL
this.litellmUrl = `http://${this.litellmService.loadBalancer.loadBalancerDnsName}`;

// =====
// AUTO SCALING (Tier 2+)
// =====

if (tierConfig.litellm.enableAutoScaling) {
  const scaling = this.litellmService.service.autoScaleTaskCount({
    minCapacity: tierConfig.litellm.minTasks || 1,
    maxCapacity: tierConfig.litellm.maxTasks || tierConfig.litellm.taskCount * 2,
  });

  // Scale on CPU
  scaling.scaleOnCpuUtilization('CpuScaling', {
    targetUtilizationPercent: 70,
  });
}

```

```

        scaleInCooldown: cdk.Duration.minutes(5),
        scaleOutCooldown: cdk.Duration.minutes(2),
    });

    // Scale on Memory
    scaling.scaleOnMemoryUtilization('MemoryScaling', {
        targetUtilizationPercent: 80,
        scaleInCooldown: cdk.Duration.minutes(5),
        scaleOutCooldown: cdk.Duration.minutes(2),
    });

    // Scale on request count (Tier 3+)
    if (tierConfig.tier >= 3) {
        scaling.scaleOnRequestCount('RequestScaling', {
            requestsPerTarget: 1000,
            targetGroup: this.litellmService.targetGroup,
            scaleInCooldown: cdk.Duration.minutes(5),
            scaleOutCooldown: cdk.Duration.minutes(1),
        });
    }
}

// =====
// SAGEMAKER ENDPOINTS (Tier 3+)
// =====

if (tierConfig.sagemaker.enabled) {
    this.sagemakerRole = new iam.Role(this, 'SageMakerRole', {
        roleName: `${resourcePrefix}-sagemaker-execution`,
        assumedBy: new iam.ServicePrincipal('sagemaker.amazonaws.com'),
        managedPolicies: [
            iam.ManagedPolicy.fromAwsManagedPolicyName('AmazonSageMakerFullAccess'),
        ],
    });

    // Grant access to model artifacts in S3
    this.sagemakerRole.addToPolicy(new iam.PolicyStatement({
        actions: [
            's3:GetObject',
            's3:ListBucket',
        ],
        resources: [
            'arn:aws:s3:::jumpstart-cache-prod-*',
            'arn:aws:s3:::jumpstart-cache-prod-*/*',
            'arn:aws:s3:::huggingface-sagemaker-*',
            'arn:aws:s3:::huggingface-sagemaker-*/*',
        ],
    }));
}

```

```

// Grant CloudWatch Logs access
this.sagemakerRole.addToPolicy(new iam.PolicyStatement({
    actions: [
        'logs:CreateLogGroup',
        'logs:CreateLogStream',
        'logs:PutLogEvents',
    ],
    resources: [`arn:aws:logs:${this.region}:${this.account}:log-group:/aws/sagemaker/*`],
}));

// Create self-hosted model endpoints based on tier
this.createSageMakerEndpoints(resourcePrefix, tierConfig);
}

// =====
// SSM PARAMETERS
// =====

new ssm.StringParameter(this, 'LitellmUrlParam', {
    parameterName: `/radiant/${props.appId}/${props.environment}/ai/litellm-url`,
    stringValue: this.litellmUrl,
});

new ssm.StringParameter(this, 'LitellmAlbDnsParam', {
    parameterName: `/radiant/${props.appId}/${props.environment}/ai/litellm-alb-dns`,
    stringValue: this.litellmService.loadBalancer.loadBalancerDnsName,
});

new ssm.StringParameter(this, 'ProviderSecretsArnParam', {
    parameterName: `/radiant/${props.appId}/${props.environment}/ai/provider-secrets-arn`,
    stringValue: this.providerSecretsArn,
});

new ssm.StringParameter(this, 'ClusterArnParam', {
    parameterName: `/radiant/${props.appId}/${props.environment}/ai/ecs-cluster-arn`,
    stringValue: this.cluster.clusterArn,
});

// =====
// TAGS
// =====

applyTags(this, {
    appId: props.appId,
    environment: props.environment,
    tier: props.tier,
});

```

```

// =====
// OUTPUTS
// =====

new cdk.CfnOutput(this, 'ClusterName', {
    value: this.cluster.clusterName,
    description: 'ECS Cluster Name',
    exportName: `${resourcePrefix}-ai-cluster-name`,
});

new cdk.CfnOutput(this, 'LitellmUrl', {
    value: this.litellmUrl,
    description: 'LiteLLM Internal URL',
    exportName: `${resourcePrefix}-litellm-url`,
});

new cdk.CfnOutput(this, 'LitellmAlbDns', {
    value: this.litellmService.loadBalancer.loadBalancerDnsName,
    description: 'LiteLLM ALB DNS Name',
    exportName: `${resourcePrefix}-litellm-alb-dns`,
});

new cdk.CfnOutput(this, 'ProviderSecretsArn', {
    value: this.providerSecretsArn,
    description: 'Provider Secrets ARN',
    exportName: `${resourcePrefix}-provider-secrets-arn`,
});

new cdk.CfnOutput(this, 'ServiceDiscoveryNamespace', {
    value: this.namespace.namespaceName,
    description: 'Service Discovery Namespace',
    exportName: `${resourcePrefix}-ai-namespace`,
});
}

/**
 * Create SageMaker endpoints for self-hosted models
 */
private createSageMakerEndpoints(resourcePrefix: string, tierConfig: TierConfig): void {
    // Define model configurations based on tier
    const modelConfigs = this.getModelConfigs(tierConfig);

    for (const config of modelConfigs) {
        // Model
        const model = new sagemaker.CfnModel(this, `Model${config.id}`, {
            modelName: `${resourcePrefix}-${config.id}`,
            executionRoleArn: this.sagemakerRole!.roleArn,

```

```

    primaryContainer: {
      image: config.image,
      environment: config.environment,
      modelDataUrl: config.modelDataUrl,
    },
  });

  // Endpoint Config
  const endpointConfig = new sagemaker.CfnEndpointConfig(this, `EndpointConfig${config.id}`, {
    endpointConfigName: `${resourcePrefix}-${config.id}-config`,
    productionVariants: [
      {
        variantName: 'AllTraffic',
        modelName: model.modelName!,
        initialInstanceCount: 0, // Start scaled to zero
        instanceType: config.instanceType,
        initialVariantWeight: 1,
      },
    ],
  });
  endpointConfig.addDependency(model);

  // Endpoint
  const endpoint = new sagemaker.CfnEndpoint(this, `Endpoint${config.id}`, {
    endpointName: `${resourcePrefix}-${config.id}`,
    endpointConfigName: endpointConfig.endpointConfigName!,
  });
  endpoint.addDependency(endpointConfig);

  this.sagemakerEndpoints.set(config.id, endpoint);

  // Output
  new cdk.CfnOutput(this, `Endpoint${config.id}Name`, {
    value: endpoint.endpointName!,
    description: `SageMaker Endpoint for ${config.name}`,
    exportName: `${resourcePrefix}-sagemaker-${config.id}`,
  });
}
}

/**
 * Get model configurations based on tier
 */
private getModelConfigs(tierConfig: TierConfig): ModelConfig[] {
  const configs: ModelConfig[] = [];

  // Tier 3: Basic models
  if (tierConfig.tier >= 3) {

```

```

configs.push(
  {
    id: 'mistral-7b',
    name: 'Mistral 7B Instruct',
    image: `763104351884.dkr.ecr.${this.region}.amazonaws.com/huggingface-pytorch-tgi-infer`,
    instanceType: 'ml.g4dn.xlarge',
    environment: {
      HF_MODEL_ID: 'mistralai/Mistral-7B-Instruct-v0.2',
      MAX_INPUT_LENGTH: '4096',
      MAX_TOTAL_TOKENS: '8192',
    },
  },
);
{
  id: 'whisper-large',
  name: 'Whisper Large V3',
  image: `763104351884.dkr.ecr.${this.region}.amazonaws.com/huggingface-pytorch-infer`,
  instanceType: 'ml.g4dn.xlarge',
  environment: {
    HF_MODEL_ID: 'openai/whisper-large-v3',
    HF_TASK: 'automatic-speech-recognition',
  },
},
);
}

// Tier 4+: Advanced models
if (tierConfig.tier >= 4) {
  configs.push(
    {
      id: 'llama-70b',
      name: 'Llama 2 70B Chat',
      image: `763104351884.dkr.ecr.${this.region}.amazonaws.com/huggingface-pytorch-tgi-infer`,
      instanceType: 'ml.g5.12xlarge',
      environment: {
        HF_MODEL_ID: 'meta-llama/Llama-2-70b-chat-hf',
        MAX_INPUT_LENGTH: '4096',
        MAX_TOTAL_TOKENS: '8192',
        SM_NUM_GPUS: '4',
      },
    },
  ),
  {
    id: 'sdxl',
    name: 'Stable Diffusion XL',
    image: `763104351884.dkr.ecr.${this.region}.amazonaws.com/stabilityai-pytorch-infer`,
    instanceType: 'ml.g5.2xlarge',
    environment: {
      MODEL_ID: 'stabilityai/stable-diffusion-xl-base-1.0',
    },
  },
}

```

```

    },
    {
        id: 'yolov8',
        name: 'YOLOv8 Object Detection',
        image: `763104351884.dkr.ecr.${this.region}.amazonaws.com/pytorch-inference:2.1-gpu-1`,
        instanceType: 'ml.g4dn.xlarge',
        environment: {
            MODEL_NAME: 'yolov8x',
        },
    },
);
}

// Tier 5: Enterprise models
if (tierConfig.tier >= 5) {
    configs.push(
        {
            id: 'qwen-72b',
            name: 'Qwen 72B Chat',
            image: `763104351884.dkr.ecr.${this.region}.amazonaws.com/huggingface-pytorch-tgi-infer`,
            instanceType: 'ml.p4d.24xlarge',
            environment: {
                HF_MODEL_ID: 'Qwen/Qwen-72B-Chat',
                MAX_INPUT_LENGTH: '8192',
                MAX_TOTAL_TOKENS: '16384',
                SM_NUM_GPUS: '8',
            },
        },
        {
            id: 'esm2',
            name: 'ESM-2 Protein',
            image: `763104351884.dkr.ecr.${this.region}.amazonaws.com/huggingface-pytorch-inferen`,
            instanceType: 'ml.g5.4xlarge',
            environment: {
                HF_MODEL_ID: 'facebook/esm2_t36_3B_UR50D',
                HF_TASK: 'feature-extraction',
            },
        },
    );
}

return configs;
}

interface ModelConfig {
    id: string;
    name: string;
}

```

```

    image: string;
    instanceType: string;
    environment: Record<string, string>;
    modelDataUrl?: string;
}

```

PART 4: API STACK (REST + GRAPHQL)

packages/infrastructure/lib/stacks/api.stack.ts

```

import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as apigateway from 'aws-cdk-lib/aws-apigateway';
import * as appsync from 'aws-cdk-lib/aws-appsync';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as lambdaNodejs from 'aws-cdk-lib/aws-lambda-nodejs';
import * as cognito from 'aws-cdk-lib/aws-cognito';
import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';
import * as rds from 'aws-cdk-lib/aws-rds';
import * as s3 from 'aws-cdk-lib/aws-s3';
import * as iam from 'aws-cdk-lib/aws-iam';
import * as ssm from 'aws-cdk-lib/aws-ssm';
import * as logs from 'aws-cdk-lib/aws-logs';
import * as wafv2 from 'aws-cdk-lib/aws-wafv2';
import * as secretsmanager from 'aws-cdk-lib/aws-secretsmanager';
import { Construct } from 'constructs';
import { TierConfig, TierLevel } from '../../config/tiers';
import { applyTags } from '../../config/tags';

export interface APIStackProps extends cdk.StackProps {
    appId: string;
    appName: string;
    environment: string;
    tier: TierLevel;
    tierConfig: TierConfig;
    domain: string;
    vpc: ec2.Vpc;
    lambdaSecurityGroup: ec2.SecurityGroup;
    userPool: cognito.UserPool;
    userPoolClient: cognito.UserPoolClient;
    litellmUrl: string;
    auroraCluster: rds.DatabaseCluster;
    auroraSecret: secretsmanager.ISecret;
    usageTable: dynamodb.Table;
    sessionsTable: dynamodb.Table;
    cacheTable: dynamodb.Table;
    mediaBucket: s3.Bucket;
}

```

```

    lambdaRole: iam.Role;
    webAcl?: wafv2.CfnWebACL;
}

/**
 * API Stack
 *
 * Creates REST API Gateway for external integrations and
 * AppSync GraphQL API for client applications.
 */
export class APIStack extends cdk.Stack {
    // REST API
    public readonly api: apigateway.RestApi;
    public readonly apiUrl: string;

    // GraphQL API
    public readonly graphqlApi: appsync.GraphqlApi;
    public readonly graphqlUrl: string;

    // Lambda Functions
    public readonly routerFunction: lambda.Function;
    public readonly chatFunction: lambda.Function;
    public readonly modelsFunction: lambda.Function;
    public readonly providersFunction: lambda.Function;

    constructor(scope: Construct, id: string, props: APIStackProps) {
        super(scope, id, props);

        const { tierConfig, vpc } = props;
        const resourcePrefix = `${props.appId}-${props.environment}`;

        // =====
        // SHARED LAMBDA CONFIGURATION
        // =====

        const lambdaEnvironment = {
            APP_ID: props.appId,
            ENVIRONMENT: props.environment,
            TIER: String(props.tier),
            LITELLM_URL: props.litellmUrl,
            AURORA_SECRET_ARN: props.auroraSecret.secretArn,
            AURORA_CLUSTER_ARN: props.auroraCluster.clusterArn,
            USAGE_TABLE: props.usageTable.tableName,
            SESSIONS_TABLE: props.sessionsTable.tableName,
            CACHE_TABLE: props.cacheTable.tableName,
            MEDIA_BUCKET: props.mediaBucket.bucketName,
            USER_POOL_ID: props.userPool.userPoolId,
            LOG_LEVEL: props.environment === 'prod' ? 'info' : 'debug',

```

```

};

const lambdaLogGroup = new logs.LogGroup(this, 'LambdaLogs', {
  logGroupName: `/radiant/${props.appId}/${props.environment}/lambda`,
  retention: tierConfig.tier >= 3
    ? logs.RetentionDays.THREE_MONTHS
    : logs.RetentionDays.TWO_WEEKS,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
});

// =====
// LAMBDA FUNCTIONS
// =====

// Router Lambda (main entry point)
this.routerFunction = new lambdaNodejs.NodejsFunction(this, 'RouterFunction', {
  functionName: `${resourcePrefix}-router`,
  runtime: lambda.Runtime.NODEJS_20_X,
  handler: 'handler',
  entry: 'lambda/api/router.ts',
  timeout: cdk.Duration.seconds(tierConfig.lambda.timeoutSeconds),
  memorySize: tierConfig.lambda.memoryMB,
  vpc,
  vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
  securityGroups: [props.lambdaSecurityGroup],
  environment: lambdaEnvironment,
  role: props.lambdaRole,
  tracing: tierConfig.features.xrayTracing
    ? lambda.Tracing.ACTIVE
    : lambda.Tracing.DISABLED,
  logGroup: lambdaLogGroup,
  bundling: {
    minify: true,
    sourceMap: true,
    target: 'node20',
    externalModules: ['@aws-sdk/*'],
  },
});

// Chat Lambda
this.chatFunction = new lambdaNodejs.NodejsFunction(this, 'ChatFunction', {
  functionName: `${resourcePrefix}-chat`,
  runtime: lambda.Runtime.NODEJS_20_X,
  handler: 'handler',
  entry: 'lambda/api/chat.ts',
  timeout: cdk.Duration.seconds(300), // Longer timeout for AI requests
  memorySize: tierConfig.lambda.memoryMB,
  vpc,

```

```

vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
securityGroups: [props.lambdaSecurityGroup],
environment: lambdaEnvironment,
role: props.lambdaRole,
tracing: tierConfig.features.xrayTracing
    ? lambda.Tracing.ACTIVE
    : lambda.Tracing.DISABLED,
logGroup: lambdaLogGroup,
bundling: {
    minify: true,
    sourceMap: true,
    target: 'node20',
    externalModules: ['@aws-sdk/*'],
},
});

// Models Lambda
this.modelsFunction = new lambdaNodejs.NodejsFunction(this, 'ModelsFunction', {
    functionName: `${resourcePrefix}-models`,
    runtime: lambda.Runtime.NODEJS_20_X,
    handler: 'handler',
    entry: 'lambda/api/models.ts',
    timeout: cdk.Duration.seconds(30),
    memorySize: 512,
    vpc,
    vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
    securityGroups: [props.lambdaSecurityGroup],
    environment: lambdaEnvironment,
    role: props.lambdaRole,
    logGroup: lambdaLogGroup,
    bundling: {
        minify: true,
        sourceMap: true,
        target: 'node20',
        externalModules: ['@aws-sdk/*'],
    },
},
});

// Providers Lambda
this.providersFunction = new lambdaNodejs.NodejsFunction(this, 'ProvidersFunction', {
    functionName: `${resourcePrefix}-providers`,
    runtime: lambda.Runtime.NODEJS_20_X,
    handler: 'handler',
    entry: 'lambda/api/providers.ts',
    timeout: cdk.Duration.seconds(30),
    memorySize: 512,
    vpc,
    vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },

```

```

securityGroups: [props.lambdaSecurityGroup],
environment: lambdaEnvironment,
role: props.lambdaRole,
logGroup: lambdaLogGroup,
bundling: {
  minify: true,
  sourceMap: true,
  target: 'node20',
  externalModules: ['@aws-sdk/*'],
},
});

// Grant permissions
props.auroraSecret.grantRead(this.routerFunction);
props.auroraSecret.grantRead(this.chatFunction);
props.auroraSecret.grantRead(this.modelsFunction);
props.auroraSecret.grantRead(this.providersFunction);

props.usageTable.grantReadWriteData(this.chatFunction);
props.sessionsTable.grantReadWriteData(this.chatFunction);
props.cacheTable.grantReadWriteData(this.routerFunction);

props.mediaBucket.grantReadWrite(this.chatFunction);

// =====
// REST API GATEWAY
// =====

this.api = new apigateway.RestApi(this, 'RestApi', {
  restApiName: `${resourcePrefix}-api`,
  description: `RADIANT REST API for ${props.appName}`,

  deployOptions: {
    stageName: props.environment,
    tracingEnabled: tierConfig.features.xrayTracing,
    loggingLevel: apigateway.MethodLoggingLevel.INFO,
    dataTraceEnabled: props.environment !== 'prod',
    metricsEnabled: true,
    throttlingBurstLimit: tierConfig.tier >= 4 ? 5000 : 1000,
    throttlingRateLimit: tierConfig.tier >= 4 ? 10000 : 2000,
  },

  // CORS
  defaultCorsPreflightOptions: {
    allowOrigins: [
      `https://${props.domain}`,
      `https://*.${props.domain}`,
      'http://localhost:*',
    ],
  },
});

```

```

    ],
    allowMethods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
    allowHeaders: [
        'Content-Type',
        'Authorization',
        'X-API-Key',
        'X-Tenant-Id',
    ],
    allowCredentials: true,
    maxAge: cdk.Duration.hours(1),
},

// Binary media types
binaryMediaTypes: [
    'image/*',
    'audio/*',
    'video/*',
    'application/octet-stream',
],

// Minimum compression
minimumCompressionSize: 1024,

// Endpoint configuration
endpointConfiguration: {
    types: [apigateway.EndpointType.REGIONAL],
},
});

this.apiUrl = this.api.url;

// Cognito Authorizer
const cognitoAuthorizer = new apigateway.CognitoUserPoolsAuthorizer(this, 'CognitoAuthorizer', {
    cognitoUserPools: [props.userPool],
    authorizerName: `${resourcePrefix}-cognito-auth`,
    identitySource: 'method.request.header.Authorization',
});

// API Routes
const v2 = this.api.root.addResource('api').addResource('v2');

// /api/v2/chat
const chat = v2.addResource('chat');
chat.addResource('completions').addMethod('POST',
    new apigateway.LambdaIntegration(this.chatFunction),
    {
        authorizer: cognitoAuthorizer,
        authorizationType: apigateway.AuthorizationType.COGNITO,
    }
);

```

```

    }
  );

  // /api/v2/models
  const models = v2.addResource('models');
  models.addMethod('GET',
    new apigateway.LambdaIntegration(this.modelsFunction),
    {
      authorizer: cognitoAuthorizer,
      authorizationType: apigateway.AuthorizationType.COGNITO,
    }
  );
  models.addResource('{modelId}').addMethod('GET',
    new apigateway.LambdaIntegration(this.modelsFunction),
    {
      authorizer: cognitoAuthorizer,
      authorizationType: apigateway.AuthorizationType.COGNITO,
    }
  );

  // /api/v2/providers
  const providers = v2.addResource('providers');
  providers.addMethod('GET',
    new apigateway.LambdaIntegration(this.providersFunction),
    {
      authorizer: cognitoAuthorizer,
      authorizationType: apigateway.AuthorizationType.COGNITO,
    }
  );

  // /api/v2/health (no auth)
  v2.addResource('health').addMethod('GET',
    new apigateway.LambdaIntegration(this.routerFunction),
    {
      authorizationType: apigateway.AuthorizationType.NONE,
    }
  );

  // Associate WAF (Tier 2+)
  if (props.webAcl) {
    new wafv2.CfnWebACLAssociation(this, 'ApiWafAssociation', {
      resourceArn: this.api.deploymentStage.stageArn,
      webAclArn: props.webAcl.attrArn,
    });
  }

  // =====
  // APPSYNC GRAPHQL API

```

```

// =====

this.graphqlApi = new appsync.GraphqlApi(this, 'GraphqlApi', {
  name: `${resourcePrefix}-graphql`,

  // Schema
  definition: appsync.Definition.fromFile('graphql/schema.graphql'),

  // Authorization
  authorizationConfig: {
    defaultAuthorization: {
      authorizationType: appsync.AuthorizationType.USER_POOL,
      userPoolConfig: {
        userPool: props.userPool,
        defaultAction: appsync.UserPoolDefaultAction.ALLOW,
      },
    },
    additionalAuthorizationModes: [
      {
        authorizationType: appsync.AuthorizationType.API_KEY,
        apiKeyConfig: {
          name: 'default',
          expires: cdk.Expiration.after(cdk.Duration.days(365)),
        },
      },
      {
        authorizationType: appsync.AuthorizationType.IAM,
      },
    ],
  },

  // Logging
  logConfig: {
    fieldLogLevel: props.environment === 'prod'
      ? appsync.FieldLogLevel.ERROR
      : appsync.FieldLogLevel.ALL,
    excludeVerboseContent: props.environment === 'prod',
  },

  // X-Ray
  xrayEnabled: tierConfig.features.xrayTracing,
});

this.graphqlUrl = this.graphqlApi.graphqlUrl;

// Lambda Data Source
const lambdaDs = this.graphqlApi.addLambdaDataSource(
  'LambdaDataSource',

```

```

    this.routerFunction,
    { name: 'LambdaResolver' }
  );

  // DynamoDB Data Source
  const dynamoDs = this.graphqlApi.addDynamoDbDataSource(
    'DynamoDataSource',
    props.sessionsTable,
    { name: 'DynamoResolver' }
  );

  // Resolvers
  lambdaDs.createResolver('QueryModels', {
    typeName: 'Query',
    fieldName: 'models',
  });

  lambdaDs.createResolver('QueryModel', {
    typeName: 'Query',
    fieldName: 'model',
  });

  lambdaDs.createResolver('QueryProviders', {
    typeName: 'Query',
    fieldName: 'providers',
  });

  lambdaDs.createResolver('MutationChat', {
    typeName: 'Mutation',
    fieldName: 'chat',
  });

  dynamoDs.createResolver('QuerySessions', {
    typeName: 'Query',
    fieldName: 'sessions',
    requestMappingTemplate: appsync.MappingTemplate.dynamoDbQuery(
      appsync.KeyCondition.eq('userId', 'userId'),
      'gsi1'
    ),
    responseMappingTemplate: appsync.MappingTemplate.dynamoDbResultList(),
  });

  // =====
  // SSM PARAMETERS
  // =====

  new ssm.StringParameter(this, 'ApiUrlParam', {
    parameterName: `~/radiant/${props.appId}/${props.environment}/api/rest-url`,
  });

```

```

        stringValue: this.apiUrl,
    });

    new ssm.StringParameter(this, 'GraphQLUrlParam', {
        parameterName: `/radiant/${props.appId}/${props.environment}/api/graphql-url`,
        stringValue: this.graphqlUrl,
    });

    new ssm.StringParameter(this, 'ApiIdParam', {
        parameterName: `/radiant/${props.appId}/${props.environment}/api/rest-api-id`,
        stringValue: this.api.restApiId,
    });

    // =====
    // TAGS
    // =====

    applyTags(this, {
        appId: props.appId,
        environment: props.environment,
        tier: props.tier,
    });

    // =====
    // OUTPUTS
    // =====

    new cdk.CfnOutput(this, 'RestApiUrl', {
        value: this.apiUrl,
        description: 'REST API URL',
        exportName: `${resourcePrefix}-api-url`,
    });

    new cdk.CfnOutput(this, 'RestApiId', {
        value: this.api.restApiId,
        description: 'REST API ID',
        exportName: `${resourcePrefix}-api-id`,
    });

    new cdk.CfnOutput(this, 'GraphQLApiUrl', {
        value: this.graphqlUrl,
        description: 'GraphQL API URL',
        exportName: `${resourcePrefix}-graphql-url`,
    });

    new cdk.CfnOutput(this, 'GraphQLApiId', {
        value: this.graphqlApi.apiId,
        description: 'GraphQL API ID',
    });

```

```

        exportName: `${resourcePrefix}-graphql-api-id`,
    });

    new cdk.CfnOutput(this, 'GraphqlApiKey', {
        value: this.graphqlApi.apiKey || '',
        description: 'GraphQL API Key (for development)',
        exportName: `${resourcePrefix}-graphql-api-key`,
    });
}
}

```

PART 5: ADMIN STACK

packages/infrastructure/lib/stacks/admin.stack.ts

```

import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as s3 from 'aws-cdk-lib/aws-s3';
import * as cloudfront from 'aws-cdk-lib/aws-cloudfront';
import * as origins from 'aws-cdk-lib/aws-cloudfront-origins';
import * as cognito from 'aws-cdk-lib/aws-cognito';
import * as apigateway from 'aws-cdk-lib/aws-apigateway';
import * as appsync from 'aws-cdk-lib/aws-appsync';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as lambdaNodejs from 'aws-cdk-lib/aws-lambda-nodejs';
import * as iam from 'aws-cdk-lib/aws-iam';
import * as ssm from 'aws-cdk-lib/aws-ssm';
import * as logs from 'aws-cdk-lib/aws-logs';
import * as route53 from 'aws-cdk-lib/aws-route53';
import * as route53targets from 'aws-cdk-lib/aws-route53-targets';
import * as acm from 'aws-cdk-lib/aws-certificatemanager';
import { Construct } from 'constructs';
import { TierConfig, TierLevel } from '../../config/tiers';
import { applyTags } from '../../config/tags';

export interface AdminStackProps extends cdk.StackProps {
    appId: string;
    appName: string;
    environment: string;
    tier: TierLevel;
    tierConfig: TierConfig;
    domain: string;
    vpc: ec2.Vpc;
    adminUserPool: cognito.UserPool;
    adminUserPoolClient: cognito.UserPoolClient;
    restApi: apigateway.RestApi;
    graphqlApi: appsync.GraphqlApi;
}

```

```

    assetsBucket: s3.Bucket;
    assetsDistribution: cloudfront.Distribution;
    hostedZoneId?: string;
}

/**
 * Admin Stack
 *
 * Creates admin dashboard hosting and admin-specific APIs.
 * Includes invitation system, two-person approval, and audit logging.
 */
export class AdminStack extends cdk.Stack {
    // Admin Dashboard
    public readonly adminBucket: s3.Bucket;
    public readonly adminDistribution: cloudfront.Distribution;
    public readonly adminUrl: string;

    // Admin Lambda Functions
    public readonly adminFunction: lambda.Function;
    public readonly invitationFunction: lambda.Function;
    public readonly approvalFunction: lambda.Function;

    constructor(scope: Construct, id: string, props: AdminStackProps) {
        super(scope, id, props);

        const { tierConfig, vpc } = props;
        const resourcePrefix = `${props.appId}-${props.environment}`;
        const adminDomain = `admin.${props.domain}`;

        // =====
        // ADMIN DASHBOARD S3 BUCKET
        // =====

        this.adminBucket = new s3.Bucket(this, 'AdminBucket', {
            bucketName: `${resourcePrefix}-admin-${this.account}-${this.region}`,

            encryption: s3.BucketEncryption.S3_MANAGED,
            blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
            enforceSSL: true,

            versioned: true,

            removalPolicy: cdk.RemovalPolicy.DESTROY,
            autoDeleteObjects: true,
        });

        // =====
        // CLOUDFRONT DISTRIBUTION FOR ADMIN DASHBOARD

```

```

// =====

const adminOai = new cloudfront.OriginAccessIdentity(this, 'AdminOai', {
  comment: `OAI for ${props.appName} admin dashboard`,
});

this.adminBucket.grantRead(adminOai);

// Response headers policy for security
const securityHeadersPolicy = new cloudfront.ResponseHeadersPolicy(this, 'AdminSecurityHead
responseHeadersPolicyName: `${resourcePrefix}-admin-security`,
securityHeadersBehavior: {
  contentSecurityPolicy: {
    contentSecurityPolicy: `
      default-src 'self';
      script-src 'self' 'unsafe-inline' 'unsafe-eval';
      style-src 'self' 'unsafe-inline';
      img-src 'self' data: https:;
      font-src 'self' data:;
      connect-src 'self' https://*.amazonaws.com https://*.amazoncognito.com;
      frame-ancestors 'none';
      `.replace(/\s+/g, ' ').trim(),
    override: true,
  },
  contentTypeOptions: { override: true },
  frameOptions: {
    frameOption: cloudfront.HeadersFrameOption.DENY,
    override: true,
  },
  referrerPolicy: {
    referrerPolicy: cloudfront.HeadersReferrerPolicy.STRICT_ORIGIN_WHEN_CROSS_ORIGIN,
    override: true,
  },
  strictTransportSecurity: {
    accessControlMaxAge: cdk.Duration.days(365),
    includeSubdomains: true,
    preload: true,
    override: true,
  },
  xssProtection: {
    protection: true,
    modeBlock: true,
    override: true,
  },
},
});

// Cache policy for static assets

```

```

const adminCachePolicy = new cloudfront.CachePolicy(this, 'AdminCachePolicy', {
  cachePolicyName: `${resourcePrefix}-admin-cache`,
  defaultTtl: cdk.Duration.days(1),
  minTtl: cdk.Duration.seconds(0),
  maxTtl: cdk.Duration.days(365),
  enableAcceptEncodingGzip: true,
  enableAcceptEncodingBrotli: true,
  queryStringBehavior: cloudfront.CacheQueryStringBehavior.none(),
});

this.adminDistribution = new cloudfront.Distribution(this, 'AdminDistribution', {
  comment: `${props.appName} Admin Dashboard`,

  defaultBehavior: {
    origin: new origins.S3Origin(this.adminBucket, {
      originAccessIdentity: adminOai,
    }),
    viewerProtocolPolicy: cloudfront.ViewerProtocolPolicy.REDIRECT_TO_HTTPS,
    allowedMethods: cloudfront.AllowedMethods.ALLOW_GET_HEAD,
    cachedMethods: cloudfront.CachedMethods.CACHE_GET_HEAD,
    cachePolicy: adminCachePolicy,
    responseHeadersPolicy: securityHeadersPolicy,
    compress: true,
  },

  // SPA routing
  defaultRootObject: 'index.html',
  errorResponses: [
    {
      httpStatus: 404,
      responseHttpStatus: 200,
      responsePagePath: '/index.html',
      ttl: cdk.Duration.seconds(0),
    },
    {
      httpStatus: 403,
      responseHttpStatus: 200,
      responsePagePath: '/index.html',
      ttl: cdk.Duration.seconds(0),
    },
  ],

  priceClass: cloudfront.PriceClass.PRICE_CLASS_100,
  httpVersion: cloudfront.HttpVersion.HTTP2_AND_3,
  minimumProtocolVersion: cloudfront.SecurityPolicyProtocol.TLS_V1_2_2021,
});

this.adminUrl = `https://${this.adminDistribution.distributionDomainName}`;

```

```

// =====
// ADMIN LAMBDA FUNCTIONS
// =====

const adminLogGroup = new logs.LogGroup(this, 'AdminLambdaLogs', {
  logGroupName: `/radiant/${props.appId}/${props.environment}/admin-lambda`,
  retention: tierConfig.tier >= 3
    ? logs.RetentionDays.ONE_YEAR
    : logs.RetentionDays.THREE_MONTHS,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
});

const adminLambdaRole = new iam.Role(this, 'AdminLambdaRole', {
  roleName: `${resourcePrefix}-admin-lambda`,
  assumedBy: new iam.ServicePrincipal('lambda.amazonaws.com'),
  managedPolicies: [
    iam.ManagedPolicy.fromAwsManagedPolicyName('service-role/AWSLambdaVPCAccessExecutionRole'),
  ],
});

// Cognito admin permissions
adminLambdaRole.addToPolicy(new iam.PolicyStatement({
  actions: [
    'cognito-idp:AdminCreateUser',
    'cognito-idp:AdminDeleteUser',
    'cognito-idp:AdminGetUser',
    'cognito-idp:AdminUpdateUserAttributes',
    'cognito-idp:AdminListGroupsForUser',
    'cognito-idp:AdminAddUserToGroup',
    'cognito-idp:AdminRemoveUserFromGroup',
    'cognito-idp:ListUsers',
    'cognito-idp:ListUsersInGroup',
  ],
  resources: [props.adminUserPool.userPoolArn],
})));

// SSM permissions
adminLambdaRole.addToPolicy(new iam.PolicyStatement({
  actions: ['ssm:GetParameter', 'ssm:GetParameters'],
  resources: [`arn:aws:ssm:${this.region}:${this.account}:parameter/radiant/${props.appId}/${props.environment}`],
})));

const adminEnvironment = {
  APP_ID: props.appId,
  ENVIRONMENT: props.environment,
  ADMIN_USER_POOL_ID: props.adminUserPool.userPoolId,
  ADMIN_CLIENT_ID: props.adminUserPoolClient.userPoolClientId,
};

```

```

    ADMIN_URL: this.adminUrl,
    LOG_LEVEL: props.environment === 'prod' ? 'info' : 'debug',
  };

  // Admin CRUD Function
  this.adminFunction = new lambdaNodejs.NodejsFunction(this, 'AdminFunction', {
    functionName: `${resourcePrefix}-admin-api`,
    runtime: lambda.Runtime.NODEJS_20_X,
    handler: 'handler',
    entry: 'lambda/admin/admin.ts',
    timeout: cdk.Duration.seconds(30),
    memorySize: 512,
    vpc,
    vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
    environment: adminEnvironment,
    role: adminLambdaRole,
    logGroup: adminLogGroup,
    bundling: {
      minify: true,
      sourceMap: true,
      target: 'node20',
      externalModules: ['@aws-sdk/*'],
    },
  });

  // Invitation Function
  this.invitationFunction = new lambdaNodejs.NodejsFunction(this, 'InvitationFunction', {
    functionName: `${resourcePrefix}-invitation`,
    runtime: lambda.Runtime.NODEJS_20_X,
    handler: 'handler',
    entry: 'lambda/admin/invitation.ts',
    timeout: cdk.Duration.seconds(30),
    memorySize: 512,
    vpc,
    vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
    environment: {
      ...adminEnvironment,
      SES_SENDER: `noreply@${props.domain}`,
    },
    role: adminLambdaRole,
    logGroup: adminLogGroup,
    bundling: {
      minify: true,
      sourceMap: true,
      target: 'node20',
      externalModules: ['@aws-sdk/*'],
    },
  });

```

```

// Add SES permissions for invitations
this.invitationFunction.addToRolePolicy(new iam.PolicyStatement({
  actions: ['ses:SendEmail', 'ses:SendRawEmail'],
  resources: ['*'],
}));

// Two-Person Approval Function
this.approvalFunction = new lambdaNodejs.NodejsFunction(this, 'ApprovalFunction', {
  functionName: `${resourcePrefix}-approval`,
  runtime: lambda.Runtime.NODEJS_20_X,
  handler: 'handler',
  entry: 'lambda/admin/approval.ts',
  timeout: cdk.Duration.seconds(60),
  memorySize: 512,
  vpc,
  vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
  environment: adminEnvironment,
  role: adminLambdaRole,
  logGroup: adminLogGroup,
  bundling: {
    minify: true,
    sourceMap: true,
    target: 'node20',
    externalModules: ['@aws-sdk/*'],
  },
});

// =====
// ADMIN API ROUTES
// =====

// Add admin routes to existing REST API
const adminAuthorizer = new apigateway.CognitoUserPoolsAuthorizer(this, 'AdminAuthorizer', {
  cognitoUserPools: [props.adminUserPool],
  authorizerName: `${resourcePrefix}-admin-auth`,
});

const adminApi = props.restApi.root.addResource('admin');

// /admin/users
const users = adminApi.addResource('users');
users.addMethod('GET',
  new apigateway.LambdaIntegration(this.adminFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }

```

```

);
users.addMethod('POST',
  new apigateway.LambdaIntegration(this.adminFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }
);

const userById = users.addResource('{userId}');
userById.addMethod('GET',
  new apigateway.LambdaIntegration(this.adminFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }
);
userById.addMethod('PUT',
  new apigateway.LambdaIntegration(this.adminFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }
);
userById.addMethod('DELETE',
  new apigateway.LambdaIntegration(this.adminFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }
);

// /admin/invitations
const invitations = adminApi.addResource('invitations');
invitations.addMethod('GET',
  new apigateway.LambdaIntegration(this.invitationFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }
);
invitations.addMethod('POST',
  new apigateway.LambdaIntegration(this.invitationFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }
);

```

```

const invitationById = invitations.addResource('{invitationId}');
invitationById.addMethod('DELETE',
  new apigateway.LambdaIntegration(this.invitationFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }
);
invitationById.addResource('resend').addMethod('POST',
  new apigateway.LambdaIntegration(this.invitationFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }
);

// /admin/promotions
const promotions = adminApi.addResource('promotions');
promotions.addMethod('GET',
  new apigateway.LambdaIntegration(this.approvalFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }
);
promotions.addMethod('POST',
  new apigateway.LambdaIntegration(this.approvalFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }
);

const promotionById = promotions.addResource('{promotionId}');
promotionById.addResource('approve').addMethod('POST',
  new apigateway.LambdaIntegration(this.approvalFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }
);
promotionById.addResource('reject').addMethod('POST',
  new apigateway.LambdaIntegration(this.approvalFunction),
  {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  }
);

```

```

);

// =====
// SSM PARAMETERS
// =====

new ssm.StringParameter(this, 'AdminUrlParam', {
    parameterName: `/radiant/${props.appId}/${props.environment}/admin/url`,
    stringValue: this.adminUrl,
});

new ssm.StringParameter(this, 'AdminBucketParam', {
    parameterName: `/radiant/${props.appId}/${props.environment}/admin/bucket`,
    stringValue: this.adminBucket.bucketName,
});

new ssm.StringParameter(this, 'AdminDistributionIdParam', {
    parameterName: `/radiant/${props.appId}/${props.environment}/admin/distribution-id`,
    stringValue: this.adminDistribution.distributionId,
});

// =====
// TAGS
// =====

applyTags(this, {
    appId: props.appId,
    environment: props.environment,
    tier: props.tier,
});

// =====
// OUTPUTS
// =====

new cdk.CfnOutput(this, 'AdminUrl', {
    value: this.adminUrl,
    description: 'Admin Dashboard URL',
    exportName: `${resourcePrefix}-admin-url`,
});

new cdk.CfnOutput(this, 'AdminBucketName', {
    value: this.adminBucket.bucketName,
    description: 'Admin S3 Bucket',
    exportName: `${resourcePrefix}-admin-bucket`,
});

new cdk.CfnOutput(this, 'AdminDistributionId', {

```

```

        value: this.adminDistribution.distributionId,
        description: 'Admin CloudFront Distribution ID',
        exportName: `${resourcePrefix}-admin-distribution-id`,
    });
}
}

```

PART 6: GRAPHQL SCHEMA

graphql/schema.graphql

RADIANT GraphQL Schema v2.2.0

```

# =====
# DIRECTIVES
# =====

```

```

directive @auth(
  rules: [AuthRule!]!
) on OBJECT | FIELD_DEFINITION

```

```

input AuthRule {
  allow: AuthStrategy!
  groups: [String]
  operations: [ModelOperation]
}

```

```

enum AuthStrategy {
  owner
  groups
  private
  public
}

```

```

enum ModelOperation {
  create
  read
  update
  delete
}

```

```

# =====
# TYPES
# =====

```

```

type Query {
  # Models

```

```

models(
  specialty: ModelSpecialty
  provider: String
  status: ModelStatus
  limit: Int
  nextToken: String
): ModelConnection!

model(id: ID!): Model

# Providers
providers(
  type: ProviderType
  status: ProviderStatus
  limit: Int
  nextToken: String
): ProviderConnection!

provider(id: ID!): Provider

# Sessions
sessions(
  userId: ID!
  limit: Int
  nextToken: String
): SessionConnection!

session(id: ID!): Session

# Usage
usage(
  tenantId: ID!
  startDate: AWSDateTime!
  endDate: AWSDateTime!
): UsageReport!
}

type Mutation {
  # Chat
  chat(input: ChatInput!): ChatResponse!

  # Sessions
  createSession(input: CreateSessionInput!): Session!
  updateSession(input: UpdateSessionInput!): Session!
  deleteSession(id: ID!): Boolean!

  # Admin (requires admin group)
  updateProvider(input: UpdateProviderInput!): Provider! @auth(rules: [{ allow: groups, groups

```

```

    updateModel(input: UpdateModelInput!): Model! @auth(rules: [{ allow: groups, groups: ["admin"]
}]

type Subscription {
  onChatResponse(sessionId: ID!): ChatStreamResponse
    @aws_subscribe(mutations: ["chat"])
}

# =====
# MODEL TYPES
# =====

type Model {
  id: ID!
  providerId: ID!
  provider: Provider
  name: String!
  displayName: String!
  description: String
  type: ModelType!
  specialty: ModelSpecialty!
  capabilities: [String!]!
  contextWindow: Int
  maxOutputTokens: Int
  supportsFunctions: Boolean
  supportsVision: Boolean
  supportsStreaming: Boolean
  hasThinkingMode: Boolean
  thinkingBudgetTokens: Int
  pricing: ModelPricing!
  thermalState: ThermalState
  status: ModelStatus!
  releaseDate: AWSDateTime
  deprecationDate: AWSDateTime
  createdAt: AWSDateTime!
  updatedAt: AWSDateTime!
}

type ModelConnection {
  items: [Model!]!
  nextToken: String
  totalCount: Int
}

type ModelPricing {
  inputTokens: Float
  outputTokens: Float
  perImage: Float

```

```

    perMinuteAudio: Float
    perMinuteVideo: Float
    per3DModel: Float
    billedMarkup: Float!
}

```

```

enum ModelType {
    EXTERNAL
    SELF_HOSTED
}

```

```

enum ModelSpecialty {
    TEXT_GENERATION
    TEXT_REASONING
    IMAGE_UNDERSTANDING
    IMAGE_GENERATION
    VIDEO_GENERATION
    AUDIO_TRANSCRIPTION
    AUDIO_GENERATION
    TEXT_TO_SPEECH
    CODE_GENERATION
    EMBEDDINGS
    THREE_D_GENERATION
    COMPUTER_VISION
    SCIENTIFIC
    MEDICAL
    GEOSPATIAL
}

```

```

enum ModelStatus {
    ACTIVE
    INACTIVE
    DEPRECATED
    COMING_SOON
}

```

```

enum ThermalState {
    OFF
    COLD
    WARM
    HOT
    AUTOMATIC
}

```

```

# =====
# PROVIDER TYPES
# =====

```

```

type Provider {
  id: ID!
  name: String!
  type: ProviderType!
  status: ProviderStatus!
  hipaaCompliant: Boolean!
  baaAvailable: Boolean!
  baseUrl: String
  authType: AuthType!
  capabilities: [String!]!
  models: [Model!]
  createdAt: AWSDateTime!
  updatedAt: AWSDateTime!
}

type ProviderConnection {
  items: [Provider!]!
  nextToken: String
  totalCount: Int
}

enum ProviderType {
  EXTERNAL
  SELF_HOSTED
  MID_TIER
}

enum ProviderStatus {
  ACTIVE
  INACTIVE
  DEPRECATED
}

enum AuthType {
  API_KEY
  OAUTH
  IAM
  NONE
}

# =====
# CHAT TYPES
# =====

input ChatInput {
  sessionId: ID
  model: String!
  messages: [MessageInput!]!
}

```

```

    maxTokens: Int
    temperature: Float
    stream: Boolean
    enablePHI: Boolean
    phiCategories: [String!]
}

input MessageInput {
  role: MessageRole!
  content: String!
  name: String
  images: [ImageInput!]
}

input ImageInput {
  url: String
  base64: String
  mediaType: String
}

enum MessageRole {
  system
  user
  assistant
  function
}

type ChatResponse {
  id: ID!
  sessionId: ID!
  model: String!
  message: Message!
  usage: TokenUsage!
  finishReason: String
  createdAt: AWSDateTime!
}

type ChatStreamResponse {
  id: ID!
  sessionId: ID!
  delta: String
  finishReason: String
}

type Message {
  role: MessageRole!
  content: String!
  name: String

```

```

}

type TokenUsage {
  promptTokens: Int!
  completionTokens: Int!
  totalTokens: Int!
}

# =====
# SESSION TYPES
# =====

type Session {
  id: ID!
  userId: ID!
  tenantId: ID!
  title: String
  model: String!
  messages: [Message!]!
  metadata: AWSJSON
  createdAt: AWSDateTime!
  updatedAt: AWSDateTime!
}

type SessionConnection {
  items: [Session!]!
  nextToken: String
  totalCount: Int
}

input CreateSessionInput {
  title: String
  model: String!
  metadata: AWSJSON
}

input UpdateSessionInput {
  id: ID!
  title: String
  metadata: AWSJSON
}

# =====
# USAGE TYPES
# =====

type UsageReport {
  tenantId: ID!

```

```

    period: String!
    totalCost: Float!
    totalBilled: Float!
    breakdown: [UsageBreakdown!]!
}

```

```

type UsageBreakdown {
  providerId: ID!
  modelId: ID!
  requests: Int!
  inputTokens: Int!
  outputTokens: Int!
  cost: Float!
  billed: Float!
}

```

```

# =====
# ADMIN TYPES
# =====

```

```

input UpdateProviderInput {
  id: ID!
  status: ProviderStatus
  hipaaCompliant: Boolean
}

```

```

input UpdateModelInput {
  id: ID!
  status: ModelStatus
  thermalState: ThermalState
  displayName: String
  description: String
}

```

PART 7: UPDATE STACK EXPORTS

packages/infrastructure/lib/stacks/index.ts (Updated)

```

// Foundation Stacks (Prompt 2)
export * from './foundation.stack';
export * from './networking.stack';
export * from './security.stack';
export * from './data.stack';
export * from './storage.stack';

// AI & API Stacks (Prompt 3)
export * from './auth.stack';

```

```
export * from './ai.stack';
export * from './api.stack';
export * from './admin.stack';
```

DEPLOYMENT COMMANDS

Deploy All Stacks

```
cd packages/infrastructure
```

```
# Development (Tier 1)
```

```
npx cdk deploy --all \
  --context appId=thinktank \
  --context appName="Think Tank" \
  --context environment=dev \
  --context tier=1 \
  --context domain=thinktank.YOUR_DOMAIN.com
```

```
# Staging (Tier 2)
```

```
npx cdk deploy --all \
  --context appId=thinktank \
  --context appName="Think Tank" \
  --context environment=staging \
  --context tier=2 \
  --context domain=thinktank.YOUR_DOMAIN.com
```

```
# Production (Tier 3+)
```

```
npx cdk deploy --all \
  --context appId=thinktank \
  --context appName="Think Tank" \
  --context environment=prod \
  --context tier=3 \
  --context domain=thinktank.YOUR_DOMAIN.com
```

Deploy Individual Stacks

```
# Auth Stack only
```

```
npx cdk deploy thinktank-dev-auth \
  --context appId=thinktank \
  --context environment=dev \
  --context tier=1
```

```
# AI Stack only
```

```
npx cdk deploy thinktank-dev-ai \
  --context appId=thinktank \
  --context environment=dev \
  --context tier=3
```

```
# API Stack only
npx cdk deploy thinktank-dev-api \
  --context appId=thinktank \
  --context environment=dev \
  --context tier=2

# Admin Stack only
npx cdk deploy thinktank-dev-admin \
  --context appId=thinktank \
  --context environment=dev \
  --context tier=2
```

Verify Deployment

```
# Check LiteLLM health
curl http://$(aws ssm get-parameter \
  --name /radiant/thinktank/dev/ai/litellm-alb-dns \
  --query 'Parameter.Value' --output text)/health

# Check API health
curl $(aws ssm get-parameter \
  --name /radiant/thinktank/dev/api/rest-url \
  --query 'Parameter.Value' --output text)api/v2/health

# Get Admin URL
aws ssm get-parameter \
  --name /radiant/thinktank/dev/admin/url \
  --query 'Parameter.Value' --output text
```

ESTIMATED COSTS BY TIER

Component	Tier 1	Tier 2	Tier 3	Tier 4	Tier 5
Cognito	\$0	\$5	\$25	\$100	\$500
LiteLLM (ECS)	\$25	\$75	\$200	\$500	\$1,500
API Gateway	\$5	\$25	\$100	\$300	\$1,000
AppSync	\$5	\$15	\$50	\$150	\$500
Lambda	\$5	\$20	\$75	\$250	\$750
CloudFront	\$5	\$15	\$50	\$200	\$500
SageMaker	\$0	\$0	\$200	\$1,000	\$5,000
Prompt 3 Total	~\$45	~\$155	~\$700	~\$2,500	~\$9,750

Note: Add to Prompt 2 infrastructure costs for total monthly estimate.

NEXT PROMPTS

Continue with: - **Prompt 4:** Lambda Functions - Core (Router, Chat, Models, Providers, PHI) - **Prompt 5:** Lambda Functions - Admin & Billing (Invitations, Approvals, Metering) - **Prompt 6:** Self-Hosted Models & Mid-Level Services Configuration - **Prompt 7:** External Providers & Database Schema/Migrations - **Prompt 8:** Admin Web Dashboard (Next.js) - **Prompt 9:** Assembly & Deployment Guide

End of Prompt 3: CDK AI & API Stacks RADIANT v2.2.0 - December 2024

[illegible]

END OF SECTION 3

[illegible][illegible]