Contents

SECTION 2: CDK INFRASTRUCTURE STACKS (v2.0.0)

â•

> **Dependencies:** Sections 0, 1 **Creates:** VPC, Aurora, DynamoDB, S3, KMS, WAF stacks

IMPORTANT: Type Imports

All CDK stacks should import types and configs from @radiant/shared:

```
// At the top of each stack file:
import {
  TierConfig,
  TierLevel,
  getTierConfig,
  TIER_CONFIGS,
  REGIONS,
  getMultiRegionDeployment,
} from '@radiant/shared';
```

DO NOT recreate the following files (they come from Section 0): - `lib/config/tiers.ts` - Import from @radiant/shared instead - `lib/config/regions.ts` - Import from @radiant/shared instead

DO create these CDK-specific files: - `lib/config/tags.ts` - CDK tagging utilities (shown in Section 1)

RADIANT v2.2.0 - Prompt 2: CDK Infrastructure Stacks

> **Prompt 2 of 9** | Target Size: ~35KB | Version: 3.7.0 | December 2024

OVERVIEW

This prompt creates the core AWS CDK infrastructure stacks:

1. **Foundation Stack** - SSM parameters, tagging, base configuration
2. **Networking Stack** - VPC, subnets, NAT, VPC endpoints
3. **Security Stack** - KMS keys, IAM roles, security groups, WAF
4. **Data Stack** - Aurora PostgreSQL, DynamoDB, ElastiCache
5. **Storage Stack** - S3 buckets, CloudFront distributions

All stacks are tier-aware and scale automatically based on the selected infrastructure tier (1-5).

INFRASTRUCTURE PACKAGE STRUCTURE

```
packages/infrastructure/
Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ package.json
Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ tsconfig.json
Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ cdk.json
Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ bin/
Ã¢â€ â€š   Ã¢â€ â€ Ã¢â€ â¬Ã¢â€ â¬ radiant.ts                    # CDK app entry point
Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ lib/
Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ index.ts
Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ config/
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ index.ts
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ tiers.ts                 # Tier configurations
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ regions.ts               # Region configurations
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ â€ Ã¢â€ â¬Ã¢â€ â¬ tags.ts                 # Tagging utilities
Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ stacks/
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ index.ts
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ foundation.stack.ts
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ networking.stack.ts
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ security.stack.ts
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ data.stack.ts
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ storage.stack.ts
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ ai.stack.ts              # Prompt 3
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ api.stack.ts             # Prompt 3
Ã¢â€ â€š   Ã¢â€ â€š   Ã¢â€ â€ Ã¢â€ â¬Ã¢â€ â¬ admin.stack.ts           # Prompt 3
Ã¢â€ â€š   Ã¢â€ â€ Ã¢â€ â¬Ã¢â€ â¬ constructs/
Ã¢â€ â€š       Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ index.ts
Ã¢â€ â€š       Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ aurora-global.construct.ts
Ã¢â€ â€š       Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ vpc-endpoints.construct.ts
Ã¢â€ â€š       Ã¢â€ â€ Ã¢â€ â¬Ã¢â€ â¬ waf.construct.ts
Ã¢â€ Å"Ã¢â€ â¬Ã¢â€ â¬ lambda/                          # Prompts 4-5
Ã¢â€ â€š   Ã¢â€ â€ Ã¢â€ â¬Ã¢â€ â¬ [See Prompts 4-5]
Ã¢â€ â€ Ã¢â€ â¬Ã¢â€ â¬ migrations/
    Ã¢â€ â€ Ã¢â€ â¬Ã¢â€ â¬ [See Prompt 7]
```

PART 1: PACKAGE CONFIGURATION

packages/infrastructure/package.json

```json
{
  "name": "@radiant/infrastructure",
  "version": "2.2.0",
  "private": true,
  "scripts": {
    "build": "tsc",
    "watch": "tsc -w",
    "clean": "rm -rf dist cdk.out",
```

```
    "cdk": "cdk",
    "synth": "cdk synth",
    "deploy": "cdk deploy --all",
    "deploy:dev": "cdk deploy --all --context environment=dev --context tier=1",
    "deploy:staging": "cdk deploy --all --context environment=staging --context tier=2",
    "deploy:prod": "cdk deploy --all --context environment=prod --context tier=3",
    "diff": "cdk diff",
    "destroy": "cdk destroy --all",
    "test": "jest"
  },
  "dependencies": {
    "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",
    "jest": "^29.7.0",
    "@types/jest": "^29.5.11",
    "ts-jest": "^29.1.1"
  }
}
```

packages/infrastructure/tsconfig.json

2.2 CDK ENTRY POINT

NOTE: The bin/radiant.ts file imports from @radiant/shared, not local config files.

```
  natGateways: 1,
  enableFlowLogs: true,
  flowLogsRetentionDays: 14,
},

aurora: {
  instanceClass: 'db.r6g.large',
  minCapacity: 1,
  maxCapacity: 4,
  enableGlobal: false,
  readerCount: 1,
  backupRetentionDays: 14,
  enablePerformanceInsights: true,
  performanceInsightsRetentionDays: 7,
  deletionProtection: true,
  enableIAMAuth: true,
},
```

```
dynamodb: {
  billingMode: 'PAY_PER_REQUEST',
  enablePointInTimeRecovery: true,
  enableContributorInsights: false,
},

elasticache: {
  enabled: true,
  nodeType: 'cache.t4g.micro',
  numCacheNodes: 1,
  enableMultiAz: false,
  enableAutoFailover: false,
  snapshotRetentionDays: 1,
},

lambda: {
  memoryMB: 1024,
  timeoutSeconds: 60,
},

litellm: {
  taskCount: 2,
  cpu: 512,
  memory: 1024,
  enableAutoScaling: true,
  minTasks: 1,
  maxTasks: 4,
},

sagemaker: {
  enabled: false,
},

s3: {
  intelligentTiering: true,
  lifecycleRules: true,
  replicationEnabled: false,
  versioning: true,
},

security: {
  enableWaf: true,
  enableShield: false,
  enableGuardDuty: true,
  enableInspector: false,
  enableSecurityHub: false,
  enableMacie: false,
  kmsKeyRotation: true,
```

```
},

features: {
  multiRegion: false,
  enhancedMonitoring: true,
  xrayTracing: true,
  detailedCloudWatchMetrics: false,
},

estimatedCost: {
  min: 300,
  max: 800,
  notes: 'Production-ready, single region',
},
```

},

// ==========

// TIER 3: GROWTH - Growing Production // ==========

3: { tier: 3, name: 'GROWTH', description: 'Growing production workloads',

```
vpc: {
  maxAzs: 3,
  natGateways: 2,
  enableFlowLogs: true,
  flowLogsRetentionDays: 30,
},

aurora: {
  instanceClass: 'db.r6g.xlarge',
  minCapacity: 2,
  maxCapacity: 16,
  enableGlobal: false,
  readerCount: 2,
  backupRetentionDays: 30,
  enablePerformanceInsights: true,
  performanceInsightsRetentionDays: 31,
  deletionProtection: true,
  enableIAMAuth: true,
},

dynamodb: {
  billingMode: 'PAY_PER_REQUEST',
  enablePointInTimeRecovery: true,
  enableContributorInsights: true,
},

elasticache: {
  enabled: true,
```

```
  nodeType: 'cache.r6g.large',
  numCacheNodes: 2,
  enableMultiAz: true,
  enableAutoFailover: true,
  snapshotRetentionDays: 7,
},

lambda: {
  memoryMB: 2048,
  timeoutSeconds: 120,
  reservedConcurrency: 100,
},

litellm: {
  taskCount: 4,
  cpu: 1024,
  memory: 2048,
  enableAutoScaling: true,
  minTasks: 2,
  maxTasks: 8,
},

sagemaker: {
  enabled: true,
  defaultInstanceType: 'ml.g4dn.xlarge',
  enableMultiModel: false,
},

s3: {
  intelligentTiering: true,
  lifecycleRules: true,
  replicationEnabled: false,
  versioning: true,
},

security: {
  enableWaf: true,
  enableShield: false,
  enableGuardDuty: true,
  enableInspector: true,
  enableSecurityHub: true,
  enableMacie: false,
  kmsKeyRotation: true,
},

features: {
  multiRegion: false,
  enhancedMonitoring: true,
```

```
  xrayTracing: true,
  detailedCloudWatchMetrics: true,
},

estimatedCost: {
  min: 1500,
  max: 4000,
  notes: 'High availability, full monitoring',
},
```

},

// ==
// TIER 4: SCALE - Large Production + Multi-Region // ================================
4: { tier: 4, name: 'SCALE', description: 'High-scale production with multi-region',

```
vpc: {
  maxAzs: 3,
  natGateways: 3,
  enableFlowLogs: true,
  flowLogsRetentionDays: 90,
},

aurora: {
  instanceClass: 'db.r6g.2xlarge',
  minCapacity: 4,
  maxCapacity: 64,
  enableGlobal: true,
  readerCount: 3,
  backupRetentionDays: 35,
  enablePerformanceInsights: true,
  performanceInsightsRetentionDays: 93,
  deletionProtection: true,
  enableIAMAuth: true,
},

dynamodb: {
  billingMode: 'PAY_PER_REQUEST',
  enablePointInTimeRecovery: true,
  enableContributorInsights: true,
},

elasticache: {
  enabled: true,
  nodeType: 'cache.r6g.xlarge',
  numCacheNodes: 3,
  enableMultiAz: true,
  enableAutoFailover: true,
  snapshotRetentionDays: 14,
```

```
},

lambda: {
  memoryMB: 3072,
  timeoutSeconds: 300,
  reservedConcurrency: 500,
  provisionedConcurrency: 10,
},

litellm: {
  taskCount: 8,
  cpu: 2048,
  memory: 4096,
  enableAutoScaling: true,
  minTasks: 4,
  maxTasks: 16,
},

sagemaker: {
  enabled: true,
  defaultInstanceType: 'ml.g5.xlarge',
  enableMultiModel: true,
},

s3: {
  intelligentTiering: true,
  lifecycleRules: true,
  replicationEnabled: true,
  versioning: true,
},

security: {
  enableWaf: true,
  enableShield: true,
  enableGuardDuty: true,
  enableInspector: true,
  enableSecurityHub: true,
  enableMacie: true,
  kmsKeyRotation: true,
},

features: {
  multiRegion: true,
  enhancedMonitoring: true,
  xrayTracing: true,
  detailedCloudWatchMetrics: true,
},
```

```
estimatedCost: {
  min: 5000,
  max: 15000,
  notes: 'Multi-region, advanced security',
},
```

},

// ==========

// TIER 5: ENTERPRISE - Full Compliance // ==========

5: { tier: 5, name: 'ENTERPRISE', description: 'Enterprise-scale with full compliance',

```
vpc: {
  maxAzs: 3,
  natGateways: 3,
  enableFlowLogs: true,
  flowLogsRetentionDays: 365,
},

aurora: {
  instanceClass: 'db.r6g.4xlarge',
  minCapacity: 8,
  maxCapacity: 128,
  enableGlobal: true,
  readerCount: 5,
  backupRetentionDays: 35,
  enablePerformanceInsights: true,
  performanceInsightsRetentionDays: 731,
  deletionProtection: true,
  enableIAMAuth: true,
},

dynamodb: {
  billingMode: 'PAY_PER_REQUEST',
  enablePointInTimeRecovery: true,
  enableContributorInsights: true,
},

elasticache: {
  enabled: true,
  nodeType: 'cache.r6g.2xlarge',
  numCacheNodes: 6,
  enableMultiAz: true,
  enableAutoFailover: true,
  snapshotRetentionDays: 35,
},

lambda: {
  memoryMB: 4096,
```

```
  timeoutSeconds: 900,
  reservedConcurrency: 1000,
  provisionedConcurrency: 50,
},

litellm: {
  taskCount: 16,
  cpu: 4096,
  memory: 8192,
  enableAutoScaling: true,
  minTasks: 8,
  maxTasks: 32,
},

sagemaker: {
  enabled: true,
  defaultInstanceType: 'ml.g5.2xlarge',
  enableMultiModel: true,
},

s3: {
  intelligentTiering: true,
  lifecycleRules: true,
  replicationEnabled: true,
  versioning: true,
},

security: {
  enableWaf: true,
  enableShield: true,
  enableGuardDuty: true,
  enableInspector: true,
  enableSecurityHub: true,
  enableMacie: true,
  kmsKeyRotation: true,
},

features: {
  multiRegion: true,
  enhancedMonitoring: true,
  xrayTracing: true,
  detailedCloudWatchMetrics: true,
},

estimatedCost: {
  min: 15000,
  max: 50000,
  notes: 'Full enterprise, HIPAA/SOC 2',
```

},

}, };

/** * Get tier configuration */ export function getTierConfig(tier: TierLevel): TierConfig { const config = TIER_CONFIGS[tier]; if (!config) { throw new Error(`Invalid tier: ${tier}. Must be 1-5.`); } return config; }

/** * Validate tier for environment */ export function validateTierForEnvironment(tier: TierLevel, environment: string): void { if (environment === 'prod' && tier < 2) { console.warn('Ã¢Å¡Â Ã¯Â¸Â Warning: Tier 1 (SEED) is not recommended for production'); } if (environment === 'dev' && tier > 2) { console.warn('Ã¢Å¡Â Ã¯Â¸Â Warning: Tier 3+ may be expensive for development'); } }

packages/infrastructure/lib/config/regions.ts

```typescript
/**
 * AWS Region Configuration
 */

export interface RegionConfig {
  code: string;
  name: string;
  geography: 'us' | 'eu' | 'apac';
  isPrimary: boolean;
  supportsAllServices: boolean;
  latencyZone: number;  // 1 = lowest latency from US
}

export const REGIONS: Record<string, RegionConfig> = {
  'us-east-1': {
    code: 'us-east-1',
    name: 'US East (N. Virginia)',
    geography: 'us',
    isPrimary: true,
    supportsAllServices: true,
    latencyZone: 1,
  },
  'us-west-2': {
    code: 'us-west-2',
    name: 'US West (Oregon)',
    geography: 'us',
    isPrimary: false,
    supportsAllServices: true,
    latencyZone: 2,
  },
  'eu-west-1': {
    code: 'eu-west-1',
```

```
    name: 'Europe (Ireland)',
    geography: 'eu',
    isPrimary: false,
    supportsAllServices: true,
    latencyZone: 3,
  },
  'eu-central-1': {
    code: 'eu-central-1',
    name: 'Europe (Frankfurt)',
    geography: 'eu',
    isPrimary: false,
    supportsAllServices: true,
    latencyZone: 3,
  },
  'ap-northeast-1': {
    code: 'ap-northeast-1',
    name: 'Asia Pacific (Tokyo)',
    geography: 'apac',
    isPrimary: false,
    supportsAllServices: true,
    latencyZone: 4,
  },
  'ap-southeast-1': {
    code: 'ap-southeast-1',
    name: 'Asia Pacific (Singapore)',
    geography: 'apac',
    isPrimary: false,
    supportsAllServices: true,
    latencyZone: 4,
  },
};

/**
 * Default multi-region configuration
 * Maps geography to preferred region
 */
export const MULTI_REGION_CONFIG: Record<string, string> = {
  us: 'us-east-1',
  eu: 'eu-west-1',
  apac: 'ap-northeast-1',
};

/**
 * Get regions for multi-region deployment
 */
export function getMultiRegionDeployment(primaryRegion: string): string[] {
  const regions = new Set<string>([primaryRegion]);
```

```
  for (const region of Object.values(MULTI_REGION_CONFIG)) {
    regions.add(region);
  }

  return Array.from(regions);
}

/**
 * Get region configuration
 */
export function getRegionConfig(region: string): RegionConfig {
  const config = REGIONS[region];
  if (!config) {
    throw new Error(`Unsupported region: ${region}`);
  }
  return config;
}
```

packages/infrastructure/lib/config/tags.ts

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

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

/**
 * Apply standard RADIANT tags to a construct
 */
export function applyTags(scope: Construct, tags: RadiantTags): void {
  cdk.Tags.of(scope).add('Project', tags.project || 'RADIANT');
  cdk.Tags.of(scope).add('AppId', tags.appId);
  cdk.Tags.of(scope).add('Environment', tags.environment);
  cdk.Tags.of(scope).add('Tier', String(tags.tier));
  cdk.Tags.of(scope).add('Version', tags.version || '2.2.0');
  cdk.Tags.of(scope).add('ManagedBy', 'CDK');

  if (tags.owner) {
    cdk.Tags.of(scope).add('Owner', tags.owner);
  }
```

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

/**
 * Get standard resource naming
 */
export function getResourceName(
  appId: string,
  environment: string,
  resourceType: string,
  suffix?: string
): string {
  const base = `${appId}-${environment}-${resourceType}`;
  return suffix ? `${base}-${suffix}` : base;
}
```

PART 4: FOUNDATION STACK

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

```
import * as cdk from 'aws-cdk-lib';
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 FoundationStackProps extends cdk.StackProps {
  appId: string;
  appName: string;
  environment: string;
  tier: TierLevel;
  tierConfig: TierConfig;
  domain: string;
}

/**
 * Foundation Stack
 *
 * Creates base configuration and SSM parameters used by all other stacks.
 * This stack has no dependencies and is deployed first.
 */
export class FoundationStack extends cdk.Stack {
```

```
public readonly tierParameter: ssm.StringParameter;
public readonly configParameter: ssm.StringParameter;

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

  const prefix = `/radiant/${props.appId}/${props.environment}`;

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

  // Tier level
  this.tierParameter = new ssm.StringParameter(this, 'TierParameter', {
    parameterName: `${prefix}/tier`,
    stringValue: String(props.tier),
    description: `RADIANT infrastructure tier for ${props.appName} ${props.environment}`,
    tier: ssm.ParameterTier.STANDARD,
  });

  // Full tier configuration (JSON)
  this.configParameter = new ssm.StringParameter(this, 'TierConfigParameter', {
    parameterName: `${prefix}/tier-config`,
    stringValue: JSON.stringify(props.tierConfig),
    description: `RADIANT tier configuration for ${props.appName} ${props.environment}`,
    tier: ssm.ParameterTier.ADVANCED,  // Allows larger values
  });

  // App metadata
  new ssm.StringParameter(this, 'AppIdParameter', {
    parameterName: `${prefix}/app-id`,
    stringValue: props.appId,
    description: 'Application identifier',
  });

  new ssm.StringParameter(this, 'AppNameParameter', {
    parameterName: `${prefix}/app-name`,
    stringValue: props.appName,
    description: 'Application display name',
  });

  new ssm.StringParameter(this, 'DomainParameter', {
    parameterName: `${prefix}/domain`,
    stringValue: props.domain,
    description: 'Base domain for the application',
  });

  new ssm.StringParameter(this, 'VersionParameter', {
```

```
  parameterName: `${prefix}/version`,
  stringValue: '2.2.0',
  description: 'RADIANT platform version',
});

new ssm.StringParameter(this, 'DeployedAtParameter', {
  parameterName: `${prefix}/deployed-at`,
  stringValue: new Date().toISOString(),
  description: 'Last deployment timestamp',
});

// Feature flags based on tier
new ssm.StringParameter(this, 'FeaturesParameter', {
  parameterName: `${prefix}/features`,
  stringValue: JSON.stringify({
    multiRegion: props.tierConfig.features.multiRegion,
    waf: props.tierConfig.security.enableWaf,
    guardDuty: props.tierConfig.security.enableGuardDuty,
    sagemaker: props.tierConfig.sagemaker.enabled,
    elasticache: props.tierConfig.elasticache.enabled,
    xray: props.tierConfig.features.xrayTracing,
  }),
  description: 'Enabled features for this deployment',
});

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

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

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

new cdk.CfnOutput(this, 'TierName', {
  value: props.tierConfig.name,
  description: 'Infrastructure tier name',
  exportName: `${props.appId}-${props.environment}-tier-name`,
});

new cdk.CfnOutput(this, 'ParameterPrefix', {
  value: prefix,
  description: 'SSM parameter prefix',
```

```
      exportName: `${props.appId}-${props.environment}-param-prefix`,
    });
  }
}
```

PART 5: NETWORKING STACK

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

```
import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as logs from 'aws-cdk-lib/aws-logs';
import * as iam from 'aws-cdk-lib/aws-iam';
import { Construct } from 'constructs';
import { TierConfig, TierLevel } from '../config/tiers';
import { applyTags } from '../config/tags';

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

/**
 * Networking Stack
 *
 * Creates VPC with public, private, and isolated subnets.
 * Configures NAT gateways, VPC endpoints, and flow logs.
 */
export class NetworkingStack extends cdk.Stack {
  public readonly vpc: ec2.Vpc;
  public readonly flowLogsGroup: logs.LogGroup;

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

    const { tierConfig } = props;

    // ============================================================================
    // VPC FLOW LOGS
    // ============================================================================

    this.flowLogsGroup = new logs.LogGroup(this, 'VpcFlowLogs', {
      logGroupName: `/radiant/${props.appId}/${props.environment}/vpc-flow-logs`,
```

```
    retention: this.getLogRetention(tierConfig.vpc.flowLogsRetentionDays),
    removalPolicy: props.environment === 'prod'
      ? cdk.RemovalPolicy.RETAIN
      : cdk.RemovalPolicy.DESTROY,
  });

  const flowLogsRole = new iam.Role(this, 'VpcFlowLogsRole', {
    assumedBy: new iam.ServicePrincipal('vpc-flow-logs.amazonaws.com'),
    description: 'Role for VPC Flow Logs',
  });

  this.flowLogsGroup.grantWrite(flowLogsRole);

  // ============================================================================
  // VPC
  // ============================================================================

  this.vpc = new ec2.Vpc(this, 'Vpc', {
    vpcName: `${props.appId}-${props.environment}-vpc`,

    // IP addressing
    ipAddresses: ec2.IpAddresses.cidr('10.0.0.0/16'),

    // Availability zones
    maxAzs: tierConfig.vpc.maxAzs,

    // NAT configuration
    natGateways: tierConfig.vpc.natGateways,
    natGatewayProvider: tierConfig.tier >= 3
      ? ec2.NatProvider.gateway()
      : ec2.NatProvider.gateway(),  // Could use NAT instances for tier 1

    // Subnet configuration
    subnetConfiguration: [
      {
        name: 'Public',
        subnetType: ec2.SubnetType.PUBLIC,
        cidrMask: 24,
        mapPublicIpOnLaunch: false,
      },
      {
        name: 'Private',
        subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS,
        cidrMask: 24,
      },
      {
        name: 'Isolated',
        subnetType: ec2.SubnetType.PRIVATE_ISOLATED,
```

```
      cidrMask: 24,
    },
  ],

  // Enable DNS
  enableDnsHostnames: true,
  enableDnsSupport: true,

  // Flow logs
  flowLogs: tierConfig.vpc.enableFlowLogs ? {
    flowLog: {
      destination: ec2.FlowLogDestination.toCloudWatchLogs(
        this.flowLogsGroup,
        flowLogsRole
      ),
      trafficType: ec2.FlowLogTrafficType.ALL,
    },
  } : undefined,
});

// ============================================================================
// VPC ENDPOINTS - GATEWAY (Free)
// ============================================================================

// S3 Gateway Endpoint (free)
this.vpc.addGatewayEndpoint('S3Endpoint', {
  service: ec2.GatewayVpcEndpointAwsService.S3,
  subnets: [
    { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
    { subnetType: ec2.SubnetType.PRIVATE_ISOLATED },
  ],
});

// DynamoDB Gateway Endpoint (free)
this.vpc.addGatewayEndpoint('DynamoDbEndpoint', {
  service: ec2.GatewayVpcEndpointAwsService.DYNAMODB,
  subnets: [
    { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
    { subnetType: ec2.SubnetType.PRIVATE_ISOLATED },
  ],
});

// ============================================================================
// VPC ENDPOINTS - INTERFACE (Tier 2+)
// ============================================================================

if (tierConfig.tier >= 2) {
  // Security group for VPC endpoints
```

```
    const endpointSecurityGroup = new ec2.SecurityGroup(this, 'EndpointSecurityGroup', {
      vpc: this.vpc,
      description: 'Security group for VPC Interface Endpoints',
      allowAllOutbound: false,
    });

    // Allow HTTPS from within VPC
    endpointSecurityGroup.addIngressRule(
      ec2.Peer.ipv4(this.vpc.vpcCidrBlock),
      ec2.Port.tcp(443),
      'Allow HTTPS from VPC'
    );

    // Secrets Manager
    this.vpc.addInterfaceEndpoint('SecretsManagerEndpoint', {
      service: ec2.InterfaceVpcEndpointAwsService.SECRETS_MANAGER,
      securityGroups: [endpointSecurityGroup],
      subnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
      privateDnsEnabled: true,
    });

    // SSM
    this.vpc.addInterfaceEndpoint('SsmEndpoint', {
      service: ec2.InterfaceVpcEndpointAwsService.SSM,
      securityGroups: [endpointSecurityGroup],
      subnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
      privateDnsEnabled: true,
    });

    // CloudWatch Logs
    this.vpc.addInterfaceEndpoint('CloudWatchLogsEndpoint', {
      service: ec2.InterfaceVpcEndpointAwsService.CLOUDWATCH_LOGS,
      securityGroups: [endpointSecurityGroup],
      subnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
      privateDnsEnabled: true,
    });
  }

  // ============================================================================
  // VPC ENDPOINTS - ADDITIONAL (Tier 3+)
  // ============================================================================

  if (tierConfig.tier >= 3) {
    const endpointSecurityGroup = ec2.SecurityGroup.fromSecurityGroupId(
      this,
      'ExistingEndpointSg',
      this.vpc.vpcEndpoints[0]?.connections?.securityGroups[0]?.securityGroupId || '',
    );
```

```
    // ECR for container images
    this.vpc.addInterfaceEndpoint('EcrEndpoint', {
      service: ec2.InterfaceVpcEndpointAwsService.ECR,
      subnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
      privateDnsEnabled: true,
    });

    this.vpc.addInterfaceEndpoint('EcrDockerEndpoint', {
      service: ec2.InterfaceVpcEndpointAwsService.ECR_DOCKER,
      subnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
      privateDnsEnabled: true,
    });

    // SageMaker Runtime (if enabled)
    if (tierConfig.sagemaker.enabled) {
      this.vpc.addInterfaceEndpoint('SageMakerRuntimeEndpoint', {
        service: ec2.InterfaceVpcEndpointAwsService.SAGEMAKER_RUNTIME,
        subnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
        privateDnsEnabled: true,
      });
    }

    // Lambda
    this.vpc.addInterfaceEndpoint('LambdaEndpoint', {
      service: ec2.InterfaceVpcEndpointAwsService.LAMBDA,
      subnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
      privateDnsEnabled: true,
    });

    // STS
    this.vpc.addInterfaceEndpoint('StsEndpoint', {
      service: ec2.InterfaceVpcEndpointAwsService.STS,
      subnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
      privateDnsEnabled: true,
    });

    // KMS
    this.vpc.addInterfaceEndpoint('KmsEndpoint', {
      service: ec2.InterfaceVpcEndpointAwsService.KMS,
      subnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
      privateDnsEnabled: true,
    });
  }

  // ============================================================================
  // TAGS
  // ============================================================================
```

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

// Tag subnets for easier identification
for (const subnet of this.vpc.publicSubnets) {
  cdk.Tags.of(subnet).add('Name', `${props.appId}-${props.environment}-public-${subnet.ava
  cdk.Tags.of(subnet).add('SubnetType', 'Public');
}
for (const subnet of this.vpc.privateSubnets) {
  cdk.Tags.of(subnet).add('Name', `${props.appId}-${props.environment}-private-${subnet.av
  cdk.Tags.of(subnet).add('SubnetType', 'Private');
}
for (const subnet of this.vpc.isolatedSubnets) {
  cdk.Tags.of(subnet).add('Name', `${props.appId}-${props.environment}-isolated-${subnet.a
  cdk.Tags.of(subnet).add('SubnetType', 'Isolated');
}

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

new cdk.CfnOutput(this, 'VpcId', {
  value: this.vpc.vpcId,
  description: 'VPC ID',
  exportName: `${props.appId}-${props.environment}-vpc-id`,
});

new cdk.CfnOutput(this, 'VpcCidr', {
  value: this.vpc.vpcCidrBlock,
  description: 'VPC CIDR block',
  exportName: `${props.appId}-${props.environment}-vpc-cidr`,
});

new cdk.CfnOutput(this, 'PrivateSubnets', {
  value: this.vpc.privateSubnets.map(s => s.subnetId).join(','),
  description: 'Private subnet IDs',
  exportName: `${props.appId}-${props.environment}-private-subnets`,
});

new cdk.CfnOutput(this, 'IsolatedSubnets', {
  value: this.vpc.isolatedSubnets.map(s => s.subnetId).join(','),
  description: 'Isolated subnet IDs',
  exportName: `${props.appId}-${props.environment}-isolated-subnets`,
});
```

```
  }

  /**
   * Convert days to CloudWatch log retention enum
   */
  private getLogRetention(days: number): logs.RetentionDays {
    const retentionMap: Record<number, logs.RetentionDays> = {
      1: logs.RetentionDays.ONE_DAY,
      3: logs.RetentionDays.THREE_DAYS,
      5: logs.RetentionDays.FIVE_DAYS,
      7: logs.RetentionDays.ONE_WEEK,
      14: logs.RetentionDays.TWO_WEEKS,
      30: logs.RetentionDays.ONE_MONTH,
      60: logs.RetentionDays.TWO_MONTHS,
      90: logs.RetentionDays.THREE_MONTHS,
      120: logs.RetentionDays.FOUR_MONTHS,
      150: logs.RetentionDays.FIVE_MONTHS,
      180: logs.RetentionDays.SIX_MONTHS,
      365: logs.RetentionDays.ONE_YEAR,
      400: logs.RetentionDays.THIRTEEN_MONTHS,
      545: logs.RetentionDays.EIGHTEEN_MONTHS,
      731: logs.RetentionDays.TWO_YEARS,
      1827: logs.RetentionDays.FIVE_YEARS,
      3653: logs.RetentionDays.TEN_YEARS,
    };

    // Find closest match
    const sortedDays = Object.keys(retentionMap)
      .map(Number)
      .sort((a, b) => a - b);

    for (const d of sortedDays) {
      if (d >= days) {
        return retentionMap[d];
      }
    }

    return logs.RetentionDays.TEN_YEARS;
  }
}
```

PART 6: SECURITY STACK

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

```
import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
```

```typescript
import * as kms from 'aws-cdk-lib/aws-kms';
import * as iam from 'aws-cdk-lib/aws-iam';
import * as wafv2 from 'aws-cdk-lib/aws-wafv2';
import * as guardduty from 'aws-cdk-lib/aws-guardduty';
import { Construct } from 'constructs';
import { TierConfig, TierLevel } from '../config/tiers';
import { applyTags } from '../config/tags';

export interface SecurityStackProps extends cdk.StackProps {
  appId: string;
  appName: string;
  environment: string;
  tier: TierLevel;
  tierConfig: TierConfig;
  domain: string;
  vpc: ec2.Vpc;
}

/**
 * Security Stack
 *
 * Creates KMS keys, IAM roles, security groups, and WAF rules.
 * Configures GuardDuty, Inspector, and other security services based on tier.
 */
export class SecurityStack extends cdk.Stack {
  // KMS Keys
  public readonly dataKey: kms.Key;
  public readonly mediaKey: kms.Key;
  public readonly logsKey: kms.Key;
  public readonly secretsKey: kms.Key;

  // Security Groups
  public readonly lambdaSecurityGroup: ec2.SecurityGroup;
  public readonly databaseSecurityGroup: ec2.SecurityGroup;
  public readonly cacheSecurityGroup: ec2.SecurityGroup;
  public readonly ecsSecurityGroup: ec2.SecurityGroup;
  public readonly albSecurityGroup: ec2.SecurityGroup;

  // WAF
  public readonly webAcl?: wafv2.CfnWebACL;

  // IAM Roles
  public readonly lambdaExecutionRole: iam.Role;

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

    const { tierConfig, vpc } = props;
```

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

// ============================================================================
// KMS KEYS
// ============================================================================

// Data encryption key (Aurora, DynamoDB)
this.dataKey = new kms.Key(this, 'DataKey', {
  alias: `alias/${resourcePrefix}-data`,
  description: `RADIANT data encryption key for ${props.appName} ${props.environment}`,
  enableKeyRotation: tierConfig.security.kmsKeyRotation,
  removalPolicy: props.environment === 'prod'
    ? cdk.RemovalPolicy.RETAIN
    : cdk.RemovalPolicy.DESTROY,
  pendingWindow: cdk.Duration.days(7),
});

// Media encryption key (S3)
this.mediaKey = new kms.Key(this, 'MediaKey', {
  alias: `alias/${resourcePrefix}-media`,
  description: `RADIANT media encryption key for ${props.appName} ${props.environment}`,
  enableKeyRotation: tierConfig.security.kmsKeyRotation,
  removalPolicy: props.environment === 'prod'
    ? cdk.RemovalPolicy.RETAIN
    : cdk.RemovalPolicy.DESTROY,
  pendingWindow: cdk.Duration.days(7),
});

// Logs encryption key
this.logsKey = new kms.Key(this, 'LogsKey', {
  alias: `alias/${resourcePrefix}-logs`,
  description: `RADIANT logs encryption key for ${props.appName} ${props.environment}`,
  enableKeyRotation: tierConfig.security.kmsKeyRotation,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
  pendingWindow: cdk.Duration.days(7),
});

// Allow CloudWatch Logs to use the key
this.logsKey.addToResourcePolicy(new iam.PolicyStatement({
  principals: [new iam.ServicePrincipal(`logs.${this.region}.amazonaws.com`)],
  actions: [
    'kms:Encrypt*',
    'kms:Decrypt*',
    'kms:ReEncrypt*',
    'kms:GenerateDataKey*',
    'kms:Describe*',
  ],
  resources: ['*'],
```

```
  conditions: {
    ArnLike: {
      'kms:EncryptionContext:aws:logs:arn': `arn:aws:logs:${this.region}:${this.account}:*
    },
  },
}));

// Secrets encryption key
this.secretsKey = new kms.Key(this, 'SecretsKey', {
  alias: `alias/${resourcePrefix}-secrets`,
  description: `RADIANT secrets encryption key for ${props.appName} ${props.environment}`,
  enableKeyRotation: tierConfig.security.kmsKeyRotation,
  removalPolicy: props.environment === 'prod'
    ? cdk.RemovalPolicy.RETAIN
    : cdk.RemovalPolicy.DESTROY,
  pendingWindow: cdk.Duration.days(7),
});

// ============================================================================
// SECURITY GROUPS
// ============================================================================

// ALB Security Group
this.albSecurityGroup = new ec2.SecurityGroup(this, 'AlbSecurityGroup', {
  vpc,
  securityGroupName: `${resourcePrefix}-alb-sg`,
  description: 'Security group for Application Load Balancer',
  allowAllOutbound: true,
});

this.albSecurityGroup.addIngressRule(
  ec2.Peer.anyIpv4(),
  ec2.Port.tcp(443),
  'Allow HTTPS from anywhere'
);

this.albSecurityGroup.addIngressRule(
  ec2.Peer.anyIpv4(),
  ec2.Port.tcp(80),
  'Allow HTTP for redirect'
);

// Lambda Security Group
this.lambdaSecurityGroup = new ec2.SecurityGroup(this, 'LambdaSecurityGroup', {
  vpc,
  securityGroupName: `${resourcePrefix}-lambda-sg`,
  description: 'Security group for Lambda functions',
  allowAllOutbound: true,
```

```
});

// ECS Security Group (for LiteLLM)
this.ecsSecurityGroup = new ec2.SecurityGroup(this, 'EcsSecurityGroup', {
  vpc,
  securityGroupName: `${resourcePrefix}-ecs-sg`,
  description: 'Security group for ECS Fargate tasks',
  allowAllOutbound: true,
});

this.ecsSecurityGroup.addIngressRule(
  this.albSecurityGroup,
  ec2.Port.tcp(4000),
  'Allow traffic from ALB to LiteLLM'
);

this.ecsSecurityGroup.addIngressRule(
  this.lambdaSecurityGroup,
  ec2.Port.tcp(4000),
  'Allow Lambda to call LiteLLM'
);

// Database Security Group
this.databaseSecurityGroup = new ec2.SecurityGroup(this, 'DatabaseSecurityGroup', {
  vpc,
  securityGroupName: `${resourcePrefix}-db-sg`,
  description: 'Security group for Aurora PostgreSQL',
  allowAllOutbound: false,
});

this.databaseSecurityGroup.addIngressRule(
  this.lambdaSecurityGroup,
  ec2.Port.tcp(5432),
  'Allow Lambda to connect to Aurora'
);

this.databaseSecurityGroup.addIngressRule(
  this.ecsSecurityGroup,
  ec2.Port.tcp(5432),
  'Allow ECS to connect to Aurora'
);

// Cache Security Group
this.cacheSecurityGroup = new ec2.SecurityGroup(this, 'CacheSecurityGroup', {
  vpc,
  securityGroupName: `${resourcePrefix}-cache-sg`,
  description: 'Security group for ElastiCache Redis',
  allowAllOutbound: false,
```

```
});

if (tierConfig.elasticache.enabled) {
  this.cacheSecurityGroup.addIngressRule(
    this.lambdaSecurityGroup,
    ec2.Port.tcp(6379),
    'Allow Lambda to connect to Redis'
  );

  this.cacheSecurityGroup.addIngressRule(
    this.ecsSecurityGroup,
    ec2.Port.tcp(6379),
    'Allow ECS to connect to Redis'
  );
}

// ============================================================================
// IAM ROLES
// ============================================================================

// Lambda execution role
this.lambdaExecutionRole = new iam.Role(this, 'LambdaExecutionRole', {
  roleName: `${resourcePrefix}-lambda-execution`,
  assumedBy: new iam.ServicePrincipal('lambda.amazonaws.com'),
  description: 'Execution role for RADIANT Lambda functions',
  managedPolicies: [
    iam.ManagedPolicy.fromAwsManagedPolicyName('service-role/AWSLambdaVPCAccessExecutionRol
  ],
});

// Basic Lambda permissions
this.lambdaExecutionRole.addToPolicy(new iam.PolicyStatement({
  sid: 'CloudWatchLogs',
  actions: [
    'logs:CreateLogGroup',
    'logs:CreateLogStream',
    'logs:PutLogEvents',
  ],
  resources: [`arn:aws:logs:${this.region}:${this.account}:log-group:/aws/lambda/${resource
}));

// SSM Parameter access
this.lambdaExecutionRole.addToPolicy(new iam.PolicyStatement({
  sid: 'SSMParameters',
  actions: [
    'ssm:GetParameter',
    'ssm:GetParameters',
    'ssm:GetParametersByPath',
```

```
    ],
    resources: [`arn:aws:ssm:${this.region}:${this.account}:parameter/radiant/${props.appId}/
  }));

  // KMS access
  this.lambdaExecutionRole.addToPolicy(new iam.PolicyStatement({
    sid: 'KMSAccess',
    actions: [
      'kms:Decrypt',
      'kms:GenerateDataKey',
    ],
    resources: [
      this.dataKey.keyArn,
      this.secretsKey.keyArn,
    ],
  }));

  // Secrets Manager access
  this.lambdaExecutionRole.addToPolicy(new iam.PolicyStatement({
    sid: 'SecretsManager',
    actions: [
      'secretsmanager:GetSecretValue',
    ],
    resources: [`arn:aws:secretsmanager:${this.region}:${this.account}:secret:${resourcePrefi
  }));

  // X-Ray tracing (if enabled)
  if (tierConfig.features.xrayTracing) {
    this.lambdaExecutionRole.addManagedPolicy(
      iam.ManagedPolicy.fromAwsManagedPolicyName('AWSXRayDaemonWriteAccess')
    );
  }

  // ============================================================================
  // WAF (Tier 2+)
  // ============================================================================

  if (tierConfig.security.enableWaf) {
    this.webAcl = new wafv2.CfnWebACL(this, 'WebAcl', {
      name: `${resourcePrefix}-waf`,
      scope: 'REGIONAL',
      defaultAction: { allow: {} },
      visibilityConfig: {
        cloudWatchMetricsEnabled: true,
        metricName: `${resourcePrefix}-waf`,
        sampledRequestsEnabled: true,
      },
      rules: [
```

```
// AWS Managed Rules - Common Rule Set
{
  name: 'AWSManagedRulesCommonRuleSet',
  priority: 1,
  overrideAction: { none: {} },
  statement: {
    managedRuleGroupStatement: {
      vendorName: 'AWS',
      name: 'AWSManagedRulesCommonRuleSet',
    },
  },
  visibilityConfig: {
    cloudWatchMetricsEnabled: true,
    metricName: 'AWSManagedRulesCommonRuleSet',
    sampledRequestsEnabled: true,
  },
},
// AWS Managed Rules - Known Bad Inputs
{
  name: 'AWSManagedRulesKnownBadInputsRuleSet',
  priority: 2,
  overrideAction: { none: {} },
  statement: {
    managedRuleGroupStatement: {
      vendorName: 'AWS',
      name: 'AWSManagedRulesKnownBadInputsRuleSet',
    },
  },
  visibilityConfig: {
    cloudWatchMetricsEnabled: true,
    metricName: 'AWSManagedRulesKnownBadInputsRuleSet',
    sampledRequestsEnabled: true,
  },
},
// AWS Managed Rules - SQL Injection
{
  name: 'AWSManagedRulesSQLiRuleSet',
  priority: 3,
  overrideAction: { none: {} },
  statement: {
    managedRuleGroupStatement: {
      vendorName: 'AWS',
      name: 'AWSManagedRulesSQLiRuleSet',
    },
  },
  visibilityConfig: {
    cloudWatchMetricsEnabled: true,
    metricName: 'AWSManagedRulesSQLiRuleSet',
```

```
          sampledRequestsEnabled: true,
        },
      },
      // Rate limiting
      {
        name: 'RateLimitRule',
        priority: 10,
        action: { block: {} },
        statement: {
          rateBasedStatement: {
            limit: tierConfig.tier >= 4 ? 5000 : 2000,
            aggregateKeyType: 'IP',
          },
        },
        visibilityConfig: {
          cloudWatchMetricsEnabled: true,
          metricName: 'RateLimitRule',
          sampledRequestsEnabled: true,
        },
      },
    ],
  });
}

// ============================================================================
// GUARDDUTY (Tier 2+)
// ============================================================================

if (tierConfig.security.enableGuardDuty) {
  // Note: GuardDuty detector is typically enabled once per account/region
  // This creates it if it doesn't exist
  new guardduty.CfnDetector(this, 'GuardDutyDetector', {
    enable: true,
    findingPublishingFrequency: 'FIFTEEN_MINUTES',
    dataSources: {
      s3Logs: { enable: true },
      kubernetes: {
        auditLogs: { enable: false },
      },
      malwareProtection: {
        scanEc2InstanceWithFindings: {
          ebsVolumes: true,
        },
      },
    },
  });
}
```

```
    // ============================================================================
    // TAGS
    // ============================================================================

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

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

    new cdk.CfnOutput(this, 'DataKeyArn', {
      value: this.dataKey.keyArn,
      description: 'Data encryption key ARN',
      exportName: `${resourcePrefix}-data-key-arn`,
    });

    new cdk.CfnOutput(this, 'MediaKeyArn', {
      value: this.mediaKey.keyArn,
      description: 'Media encryption key ARN',
      exportName: `${resourcePrefix}-media-key-arn`,
    });

    new cdk.CfnOutput(this, 'LambdaSecurityGroupId', {
      value: this.lambdaSecurityGroup.securityGroupId,
      description: 'Lambda security group ID',
      exportName: `${resourcePrefix}-lambda-sg-id`,
    });

    new cdk.CfnOutput(this, 'DatabaseSecurityGroupId', {
      value: this.databaseSecurityGroup.securityGroupId,
      description: 'Database security group ID',
      exportName: `${resourcePrefix}-db-sg-id`,
    });

    if (this.webAcl) {
      new cdk.CfnOutput(this, 'WebAclArn', {
        value: this.webAcl.attrArn,
        description: 'WAF Web ACL ARN',
        exportName: `${resourcePrefix}-waf-arn`,
      });
    }
  }
}
```

PART 7: DATA STACK

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

```typescript
import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as rds from 'aws-cdk-lib/aws-rds';
import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';
import * as elasticache from 'aws-cdk-lib/aws-elasticache';
import * as kms from 'aws-cdk-lib/aws-kms';
import * as secretsmanager from 'aws-cdk-lib/aws-secretsmanager';
import * as logs from 'aws-cdk-lib/aws-logs';
import { Construct } from 'constructs';
import { TierConfig, TierLevel } from '../config/tiers';
import { applyTags } from '../config/tags';

export interface DataStackProps extends cdk.StackProps {
  appId: string;
  appName: string;
  environment: string;
  tier: TierLevel;
  tierConfig: TierConfig;
  domain: string;
  vpc: ec2.Vpc;
  databaseSecurityGroup: ec2.SecurityGroup;
  lambdaSecurityGroup: ec2.SecurityGroup;
  dataKey: kms.Key;
  isPrimary: boolean;
  enableGlobal: boolean;
  globalClusterIdentifier?: string;
}

/**
 * Data Stack
 *
 * Creates Aurora PostgreSQL Serverless v2, DynamoDB tables, and ElastiCache.
 * Supports Aurora Global Database for multi-region deployments.
 */
export class DataStack extends cdk.Stack {
  public readonly cluster: rds.DatabaseCluster;
  public readonly dbSecret: secretsmanager.ISecret;
  public readonly globalClusterIdentifier?: string;

  // DynamoDB Tables
  public readonly usageTable: dynamodb.Table;
  public readonly sessionsTable: dynamodb.Table;
```

```
public readonly cacheTable: dynamodb.Table;

// ElastiCache
public readonly redisCluster?: elasticache.CfnCacheCluster | elasticache.CfnReplicationGroup
public readonly redisEndpoint?: string;

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

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

  // ============================================================================
  // AURORA POSTGRESQL SERVERLESS V2
  // ============================================================================

  // Database credentials
  const dbCredentials = rds.Credentials.fromGeneratedSecret('radiant_admin', {
    secretName: `${resourcePrefix}/aurora/credentials`,
    encryptionKey: props.dataKey,
  });

  this.dbSecret = dbCredentials.secret!;

  // Subnet group for Aurora
  const subnetGroup = new rds.SubnetGroup(this, 'AuroraSubnetGroup', {
    vpc,
    description: `Subnet group for ${props.appName} Aurora cluster`,
    vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE_ISOLATED },
    subnetGroupName: `${resourcePrefix}-aurora-subnet-group`,
  });

  // Parameter group with optimized settings
  const parameterGroup = new rds.ParameterGroup(this, 'AuroraParameterGroup', {
    engine: rds.DatabaseClusterEngine.auroraPostgres({
      version: rds.AuroraPostgresEngineVersion.VER_15_4,
    }),
    description: `RADIANT parameter group for ${props.appName}`,
    parameters: {
      // Security
      'rds.force_ssl': '1',

      // Logging
      'log_statement': 'ddl',
      'log_min_duration_statement': '1000',
      'log_connections': '1',
      'log_disconnections': '1',
```

```
    // Performance
    'shared_preload_libraries': 'pg_stat_statements,auto_explain',
    'pg_stat_statements.track': 'all',
    'auto_explain.log_min_duration': '3000',

    // Memory (will be adjusted by Serverless v2)
    'work_mem': '65536',
    'maintenance_work_mem': '524288',

    // WAL
    'wal_level': 'logical',
    'max_wal_senders': '10',

    // Row-Level Security
    'row_security': 'on',
  },
});

// CloudWatch Log exports
const cloudwatchLogsExports = tierConfig.tier >= 2
  ? ['postgresql', 'upgrade']
  : ['postgresql'];

// Determine if we're creating primary or secondary cluster
if (props.isPrimary) {
  // Primary cluster
  this.cluster = new rds.DatabaseCluster(this, 'AuroraCluster', {
    engine: rds.DatabaseClusterEngine.auroraPostgres({
      version: rds.AuroraPostgresEngineVersion.VER_15_4,
    }),

    clusterIdentifier: `${resourcePrefix}-aurora`,
    defaultDatabaseName: 'radiant',

    credentials: dbCredentials,

    // Serverless v2 configuration
    serverlessV2MinCapacity: tierConfig.aurora.minCapacity,
    serverlessV2MaxCapacity: tierConfig.aurora.maxCapacity,

    // Writer instance
    writer: rds.ClusterInstance.serverlessV2('Writer', {
      publiclyAccessible: false,
      enablePerformanceInsights: tierConfig.aurora.enablePerformanceInsights,
      performanceInsightRetention: tierConfig.aurora.enablePerformanceInsights
        ? this.getPerformanceInsightRetention(tierConfig.aurora.performanceInsightsRetentio
        : undefined,
      performanceInsightEncryptionKey: tierConfig.aurora.enablePerformanceInsights
```

```
      ? props.dataKey
      : undefined,
  }),

  // Reader instances
  readers: this.createReaderInstances(tierConfig),

  vpc,
  vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE_ISOLATED },
  subnetGroup,
  securityGroups: [props.databaseSecurityGroup],

  parameterGroup,

  // Encryption
  storageEncrypted: true,
  storageEncryptionKey: props.dataKey,

  // Backup
  backup: {
    retention: cdk.Duration.days(tierConfig.aurora.backupRetentionDays),
    preferredWindow: '03:00-04:00',
  },

  // Maintenance
  preferredMaintenanceWindow: 'sun:04:00-sun:05:00',

  // High availability
  deletionProtection: tierConfig.aurora.deletionProtection,

  // Logging
  cloudwatchLogsExports,
  cloudwatchLogsRetention: logs.RetentionDays.ONE_MONTH,

  // IAM authentication
  iamAuthentication: tierConfig.aurora.enableIAMAuth,

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

// Store global cluster identifier for secondary regions
if (props.enableGlobal) {
  this.globalClusterIdentifier = `${resourcePrefix}-global`;
  // Note: Aurora Global Database requires additional configuration
  // that would be handled in a separate construct
```

```
  }
} else {
  // Secondary cluster (Aurora Global Database reader)
  // This requires the global cluster to exist
  if (!props.globalClusterIdentifier) {
    throw new Error('globalClusterIdentifier required for secondary clusters');
  }

  this.cluster = new rds.DatabaseCluster(this, 'AuroraCluster', {
    engine: rds.DatabaseClusterEngine.auroraPostgres({
      version: rds.AuroraPostgresEngineVersion.VER_15_4,
    }),

    clusterIdentifier: `${resourcePrefix}-aurora-${this.region}`,

    // Serverless v2 for readers
    serverlessV2MinCapacity: tierConfig.aurora.minCapacity,
    serverlessV2MaxCapacity: tierConfig.aurora.maxCapacity,

    writer: rds.ClusterInstance.serverlessV2('Reader', {
      publiclyAccessible: false,
    }),

    vpc,
    vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE_ISOLATED },
    subnetGroup,
    securityGroups: [props.databaseSecurityGroup],

    storageEncrypted: true,
    storageEncryptionKey: props.dataKey,

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

// ============================================================================
// DYNAMODB TABLES
// ============================================================================

const dynamodbConfig = {
  billingMode: tierConfig.dynamodb.billingMode === 'PAY_PER_REQUEST'
    ? dynamodb.BillingMode.PAY_PER_REQUEST
    : dynamodb.BillingMode.PROVISIONED,
  pointInTimeRecovery: tierConfig.dynamodb.enablePointInTimeRecovery,
  contributorInsightsEnabled: tierConfig.dynamodb.enableContributorInsights,
  encryption: dynamodb.TableEncryption.CUSTOMER_MANAGED,
  encryptionKey: props.dataKey,
};
```

```
// Usage/Metering Table
this.usageTable = new dynamodb.Table(this, 'UsageTable', {
  tableName: `${resourcePrefix}-usage`,
  partitionKey: { name: 'pk', type: dynamodb.AttributeType.STRING }, // tenantId
  sortKey: { name: 'sk', type: dynamodb.AttributeType.STRING },      // timestamp
  ...dynamodbConfig,
  timeToLiveAttribute: 'ttl',
  stream: dynamodb.StreamViewType.NEW_AND_OLD_IMAGES,
  removalPolicy: props.environment === 'prod'
    ? cdk.RemovalPolicy.RETAIN
    : cdk.RemovalPolicy.DESTROY,
});

// GSI for querying by model/provider
this.usageTable.addGlobalSecondaryIndex({
  indexName: 'gsi1',
  partitionKey: { name: 'gsi1pk', type: dynamodb.AttributeType.STRING }, // modelId
  sortKey: { name: 'gsi1sk', type: dynamodb.AttributeType.STRING },      // timestamp
  projectionType: dynamodb.ProjectionType.ALL,
});

// GSI for querying by period
this.usageTable.addGlobalSecondaryIndex({
  indexName: 'gsi2',
  partitionKey: { name: 'gsi2pk', type: dynamodb.AttributeType.STRING }, // period (YYYY-M
  sortKey: { name: 'gsi2sk', type: dynamodb.AttributeType.STRING },      // tenantId#times
  projectionType: dynamodb.ProjectionType.ALL,
});

// Sessions Table
this.sessionsTable = new dynamodb.Table(this, 'SessionsTable', {
  tableName: `${resourcePrefix}-sessions`,
  partitionKey: { name: 'pk', type: dynamodb.AttributeType.STRING }, // sessionId
  ...dynamodbConfig,
  timeToLiveAttribute: 'ttl',
  removalPolicy: cdk.RemovalPolicy.DESTROY,
});

// GSI for user sessions
this.sessionsTable.addGlobalSecondaryIndex({
  indexName: 'gsi1',
  partitionKey: { name: 'userId', type: dynamodb.AttributeType.STRING },
  sortKey: { name: 'createdAt', type: dynamodb.AttributeType.STRING },
  projectionType: dynamodb.ProjectionType.ALL,
});

// Cache Table (for API response caching)
```

```
this.cacheTable = new dynamodb.Table(this, 'CacheTable', {
  tableName: `${resourcePrefix}-cache`,
  partitionKey: { name: 'pk', type: dynamodb.AttributeType.STRING }, // cacheKey
  ...dynamodbConfig,
  timeToLiveAttribute: 'ttl',
  removalPolicy: cdk.RemovalPolicy.DESTROY,
});

// ============================================================================
// ELASTICACHE REDIS (Tier 2+)
// ============================================================================

if (tierConfig.elasticache.enabled) {
  // Subnet group for ElastiCache
  const cacheSubnetGroup = new elasticache.CfnSubnetGroup(this, 'CacheSubnetGroup', {
    subnetIds: vpc.privateSubnets.map(subnet => subnet.subnetId),
    description: `Subnet group for ${props.appName} ElastiCache`,
    cacheSubnetGroupName: `${resourcePrefix}-cache-subnet-group`,
  });

  // Get the cache security group from security stack
  // Note: This is passed in from the security stack
  const cacheSecurityGroupId = cdk.Fn.importValue(
    `${props.appId}-${props.environment}-cache-sg-id`
  );

  if (tierConfig.elasticache.enableMultiAz && tierConfig.elasticache.numCacheNodes! >= 2) {
    // Replication Group (Multi-AZ)
    const replicationGroup = new elasticache.CfnReplicationGroup(this, 'RedisReplicationGr
      replicationGroupDescription: `RADIANT Redis for ${props.appName}`,
      replicationGroupId: `${resourcePrefix}-redis`,

      engine: 'redis',
      engineVersion: '7.0',

      cacheNodeType: tierConfig.elasticache.nodeType,
      numCacheClusters: tierConfig.elasticache.numCacheNodes,

      automaticFailoverEnabled: tierConfig.elasticache.enableAutoFailover,
      multiAzEnabled: tierConfig.elasticache.enableMultiAz,

      cacheSubnetGroupName: cacheSubnetGroup.cacheSubnetGroupName,
      securityGroupIds: [cacheSecurityGroupId],

      atRestEncryptionEnabled: true,
      transitEncryptionEnabled: true,

      snapshotRetentionLimit: tierConfig.elasticache.snapshotRetentionDays,
```

```
      snapshotWindow: '05:00-06:00',
      preferredMaintenanceWindow: 'sun:06:00-sun:07:00',

      autoMinorVersionUpgrade: true,
    });

    replicationGroup.addDependency(cacheSubnetGroup);

    this.redisCluster = replicationGroup;
    this.redisEndpoint = replicationGroup.attrPrimaryEndPointAddress;
  } else {
    // Single node (Tier 2)
    const cacheCluster = new elasticache.CfnCacheCluster(this, 'RedisCluster', {
      clusterName: `${resourcePrefix}-redis`,

      engine: 'redis',
      engineVersion: '7.0',

      cacheNodeType: tierConfig.elasticache.nodeType,
      numCacheNodes: 1,

      cacheSubnetGroupName: cacheSubnetGroup.cacheSubnetGroupName,
      vpcSecurityGroupIds: [cacheSecurityGroupId],

      snapshotRetentionLimit: tierConfig.elasticache.snapshotRetentionDays,
      snapshotWindow: '05:00-06:00',
      preferredMaintenanceWindow: 'sun:06:00-sun:07:00',

      autoMinorVersionUpgrade: true,
    });

    cacheCluster.addDependency(cacheSubnetGroup);

    this.redisCluster = cacheCluster;
    this.redisEndpoint = cacheCluster.attrRedisEndpointAddress;
  }
}

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

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

```
    // ============================================================================
    // OUTPUTS
    // ============================================================================

    new cdk.CfnOutput(this, 'ClusterEndpoint', {
      value: this.cluster.clusterEndpoint.hostname,
      description: 'Aurora cluster endpoint',
      exportName: `${resourcePrefix}-aurora-endpoint`,
    });

    new cdk.CfnOutput(this, 'ClusterReaderEndpoint', {
      value: this.cluster.clusterReadEndpoint.hostname,
      description: 'Aurora cluster reader endpoint',
      exportName: `${resourcePrefix}-aurora-reader-endpoint`,
    });

    new cdk.CfnOutput(this, 'DbSecretArn', {
      value: this.dbSecret.secretArn,
      description: 'Database credentials secret ARN',
      exportName: `${resourcePrefix}-db-secret-arn`,
    });

    new cdk.CfnOutput(this, 'UsageTableName', {
      value: this.usageTable.tableName,
      description: 'Usage DynamoDB table name',
      exportName: `${resourcePrefix}-usage-table`,
    });

    new cdk.CfnOutput(this, 'SessionsTableName', {
      value: this.sessionsTable.tableName,
      description: 'Sessions DynamoDB table name',
      exportName: `${resourcePrefix}-sessions-table`,
    });

    if (this.redisEndpoint) {
      new cdk.CfnOutput(this, 'RedisEndpoint', {
        value: this.redisEndpoint,
        description: 'ElastiCache Redis endpoint',
        exportName: `${resourcePrefix}-redis-endpoint`,
      });
    }
  }

  /**
   * Create reader instances based on tier configuration
   */
  private createReaderInstances(tierConfig: TierConfig): rds.IClusterInstance[] {
    const readers: rds.IClusterInstance[] = [];
```

```
    for (let i = 0; i < tierConfig.aurora.readerCount; i++) {
      readers.push(
        rds.ClusterInstance.serverlessV2(`Reader${i + 1}`, {
          publiclyAccessible: false,
          enablePerformanceInsights: tierConfig.aurora.enablePerformanceInsights,
          performanceInsightRetention: tierConfig.aurora.enablePerformanceInsights
            ? this.getPerformanceInsightRetention(tierConfig.aurora.performanceInsightsRetentio
            : undefined,
        })
      );
    }

    return readers;
  }

  /**
   * Convert days to Performance Insights retention enum
   */
  private getPerformanceInsightRetention(days: number): rds.PerformanceInsightRetention {
    if (days <= 7) return rds.PerformanceInsightRetention.DEFAULT;
    if (days <= 31) return rds.PerformanceInsightRetention.MONTHS_1;
    if (days <= 93) return rds.PerformanceInsightRetention.MONTHS_3;
    if (days <= 186) return rds.PerformanceInsightRetention.MONTHS_6;
    if (days <= 372) return rds.PerformanceInsightRetention.MONTHS_12;
    return rds.PerformanceInsightRetention.MONTHS_24;
  }
}
```

PART 8: STORAGE STACK

packages/infrastructure/lib/stacks/storage.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 kms from 'aws-cdk-lib/aws-kms';
import * as iam from 'aws-cdk-lib/aws-iam';
import * as route53 from 'aws-cdk-lib/aws-route53';
import * as acm from 'aws-cdk-lib/aws-certificatemanager';
import * as route53targets from 'aws-cdk-lib/aws-route53-targets';
import { Construct } from 'constructs';
import { TierConfig, TierLevel } from '../config/tiers';
import { applyTags } from '../config/tags';
```

```typescript
export interface StorageStackProps extends cdk.StackProps {
  appId: string;
  appName: string;
  environment: string;
  tier: TierLevel;
  tierConfig: TierConfig;
  domain: string;
  vpc: ec2.Vpc;
  mediaKey: kms.Key;
  hostedZoneId?: string;
  enableReplication?: boolean;
}

/**
 * Storage Stack
 *
 * Creates S3 buckets for media, assets, and logs.
 * Configures CloudFront distributions for global content delivery.
 */
export class StorageStack extends cdk.Stack {
  // S3 Buckets
  public readonly mediaBucket: s3.Bucket;
  public readonly assetsBucket: s3.Bucket;
  public readonly logsBucket: s3.Bucket;

  // CloudFront
  public readonly mediaDistribution: cloudfront.Distribution;
  public readonly assetsDistribution: cloudfront.Distribution;

  // URLs
  public readonly mediaUrl: string;
  public readonly assetsUrl: string;

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

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

    // ========================================================================
    // LOGS BUCKET (for access logs)
    // ========================================================================

    this.logsBucket = new s3.Bucket(this, 'LogsBucket', {
      bucketName: `${resourcePrefix}-logs-${this.account}-${this.region}`,

      encryption: s3.BucketEncryption.S3_MANAGED,
```

```
  blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,

  enforceSSL: true,

  lifecycleRules: [
    {
      id: 'TransitionToIA',
      transitions: [
        {
          storageClass: s3.StorageClass.INFREQUENT_ACCESS,
          transitionAfter: cdk.Duration.days(30),
        },
        {
          storageClass: s3.StorageClass.GLACIER,
          transitionAfter: cdk.Duration.days(90),
        },
      ],
      expiration: cdk.Duration.days(365),
    },
  ],

  removalPolicy: props.environment === 'prod'
    ? cdk.RemovalPolicy.RETAIN
    : cdk.RemovalPolicy.DESTROY,

  autoDeleteObjects: props.environment !== 'prod',
});

// ============================================================================
// MEDIA BUCKET (user uploads, AI-generated content)
// ============================================================================

this.mediaBucket = new s3.Bucket(this, 'MediaBucket', {
  bucketName: `${resourcePrefix}-media-${this.account}-${this.region}`,

  // Encryption with KMS
  encryption: s3.BucketEncryption.KMS,
  encryptionKey: props.mediaKey,
  bucketKeyEnabled: true,

  // Security
  blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
  enforceSSL: true,

  // Versioning
  versioned: tierConfig.s3.versioning,

  // CORS for direct uploads
```

```
cors: [
  {
    allowedMethods: [
      s3.HttpMethods.GET,
      s3.HttpMethods.PUT,
      s3.HttpMethods.POST,
    ],
    allowedOrigins: [
      `https://${props.domain}`,
      `https://*.${props.domain}`,
      'http://localhost:*',
    ],
    allowedHeaders: ['*'],
    exposedHeaders: ['ETag'],
    maxAge: 3600,
  },
],

// Lifecycle rules
lifecycleRules: tierConfig.s3.lifecycleRules ? [
  {
    id: 'IntelligentTiering',
    enabled: tierConfig.s3.intelligentTiering,
    transitions: tierConfig.s3.intelligentTiering ? [
      {
        storageClass: s3.StorageClass.INTELLIGENT_TIERING,
        transitionAfter: cdk.Duration.days(0),
      },
    ] : [],
  },
  {
    id: 'CleanupIncompleteUploads',
    abortIncompleteMultipartUploadAfter: cdk.Duration.days(7),
  },
  {
    id: 'ExpireOldVersions',
    noncurrentVersionExpiration: cdk.Duration.days(90),
    enabled: tierConfig.s3.versioning,
  },
] : undefined,

// Server access logging
serverAccessLogsBucket: this.logsBucket,
serverAccessLogsPrefix: 'media/',

// Removal policy
removalPolicy: props.environment === 'prod'
  ? cdk.RemovalPolicy.RETAIN
```

```
    : cdk.RemovalPolicy.DESTROY,

  autoDeleteObjects: props.environment !== 'prod',
});

// ============================================================================
// ASSETS BUCKET (static assets, compiled frontend)
// ============================================================================

this.assetsBucket = new s3.Bucket(this, 'AssetsBucket', {
  bucketName: `${resourcePrefix}-assets-${this.account}-${this.region}`,

  // S3-managed encryption for static assets
  encryption: s3.BucketEncryption.S3_MANAGED,

  blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL,
  enforceSSL: true,

  lifecycleRules: [
    {
      id: 'CleanupOldAssets',
      prefix: 'old/',
      expiration: cdk.Duration.days(30),
    },
  ],

  serverAccessLogsBucket: this.logsBucket,
  serverAccessLogsPrefix: 'assets/',

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

// ============================================================================
// CLOUDFRONT - MEDIA DISTRIBUTION
// ============================================================================

// Origin Access Identity for S3
const mediaOai = new cloudfront.OriginAccessIdentity(this, 'MediaOai', {
  comment: `OAI for ${props.appName} media bucket`,
});

this.mediaBucket.grantRead(mediaOai);

// Response headers policy
const responseHeadersPolicy = new cloudfront.ResponseHeadersPolicy(this, 'MediaResponseHea
  responseHeadersPolicyName: `${resourcePrefix}-media-headers`,
  securityHeadersBehavior: {
```

```
      contentSecurityPolicy: {
        contentSecurityPolicy: "default-src 'self'",
        override: false,
      },
      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,
      },
    },
    corsBehavior: {
      accessControlAllowCredentials: false,
      accessControlAllowHeaders: ['*'],
      accessControlAllowMethods: ['GET', 'HEAD'],
      accessControlAllowOrigins: [
        `https://${props.domain}`,
        `https://*.${props.domain}`,
      ],
      accessControlMaxAge: cdk.Duration.days(1),
      originOverride: true,
    },
  });

  // Cache policy for media
  const mediaCachePolicy = new cloudfront.CachePolicy(this, 'MediaCachePolicy', {
    cachePolicyName: `${resourcePrefix}-media-cache`,
    defaultTtl: cdk.Duration.days(1),
    minTtl: cdk.Duration.seconds(0),
    maxTtl: cdk.Duration.days(365),
    enableAcceptEncodingGzip: true,
    enableAcceptEncodingBrotli: true,
```

```
    queryStringBehavior: cloudfront.CacheQueryStringBehavior.allowList('v', 'w', 'h', 'q'),
  });

  // Media CloudFront distribution
  this.mediaDistribution = new cloudfront.Distribution(this, 'MediaDistribution', {
    comment: `${props.appName} Media Distribution`,

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

    // Additional behaviors for specific paths
    additionalBehaviors: {
      '/uploads/*': {
        origin: new origins.S3Origin(this.mediaBucket, {
          originAccessIdentity: mediaOai,
        }),
        viewerProtocolPolicy: cloudfront.ViewerProtocolPolicy.REDIRECT_TO_HTTPS,
        allowedMethods: cloudfront.AllowedMethods.ALLOW_GET_HEAD,
        cachePolicy: cloudfront.CachePolicy.CACHING_OPTIMIZED,
        responseHeadersPolicy,
      },
    },

    // Error pages
    errorResponses: [
      {
        httpStatus: 403,
        responseHttpStatus: 404,
        responsePagePath: '/404.html',
        ttl: cdk.Duration.minutes(5),
      },
      {
        httpStatus: 404,
        responseHttpStatus: 404,
        responsePagePath: '/404.html',
        ttl: cdk.Duration.minutes(5),
      },
    ],
```

```
  // Geo restrictions (if needed for compliance)
  geoRestriction: cloudfront.GeoRestriction.allowlist('US', 'CA', 'GB', 'DE', 'FR', 'JP',

  // Price class based on tier
  priceClass: tierConfig.tier >= 4
    ? cloudfront.PriceClass.PRICE_CLASS_ALL
    : tierConfig.tier >= 2
      ? cloudfront.PriceClass.PRICE_CLASS_100
      : cloudfront.PriceClass.PRICE_CLASS_100,

  // HTTP/2 and HTTP/3
  httpVersion: cloudfront.HttpVersion.HTTP2_AND_3,

  // Logging
  enableLogging: true,
  logBucket: this.logsBucket,
  logFilePrefix: 'cloudfront/media/',

  // Minimum protocol version
  minimumProtocolVersion: cloudfront.SecurityPolicyProtocol.TLS_V1_2_2021,
});

this.mediaUrl = `https://${this.mediaDistribution.distributionDomainName}`;

// ============================================================================
// CLOUDFRONT - ASSETS DISTRIBUTION
// ============================================================================

const assetsOai = new cloudfront.OriginAccessIdentity(this, 'AssetsOai', {
  comment: `OAI for ${props.appName} assets bucket`,
});

this.assetsBucket.grantRead(assetsOai);

// Cache policy for static assets (long cache)
const assetsCachePolicy = new cloudfront.CachePolicy(this, 'AssetsCachePolicy', {
  cachePolicyName: `${resourcePrefix}-assets-cache`,
  defaultTtl: cdk.Duration.days(30),
  minTtl: cdk.Duration.days(1),
  maxTtl: cdk.Duration.days(365),
  enableAcceptEncodingGzip: true,
  enableAcceptEncodingBrotli: true,
  queryStringBehavior: cloudfront.CacheQueryStringBehavior.allowList('v'),
});

this.assetsDistribution = new cloudfront.Distribution(this, 'AssetsDistribution', {
  comment: `${props.appName} Assets Distribution`,
```

```
  defaultBehavior: {
    origin: new origins.S3Origin(this.assetsBucket, {
      originAccessIdentity: assetsOai,
    }),
    viewerProtocolPolicy: cloudfront.ViewerProtocolPolicy.REDIRECT_TO_HTTPS,
    allowedMethods: cloudfront.AllowedMethods.ALLOW_GET_HEAD,
    cachedMethods: cloudfront.CachedMethods.CACHE_GET_HEAD,
    cachePolicy: assetsCachePolicy,
    responseHeadersPolicy,
    compress: true,
  },

  // SPA support
  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: tierConfig.tier >= 4
    ? cloudfront.PriceClass.PRICE_CLASS_ALL
    : cloudfront.PriceClass.PRICE_CLASS_100,

  httpVersion: cloudfront.HttpVersion.HTTP2_AND_3,

  enableLogging: true,
  logBucket: this.logsBucket,
  logFilePrefix: 'cloudfront/assets/',

  minimumProtocolVersion: cloudfront.SecurityPolicyProtocol.TLS_V1_2_2021,
});

this.assetsUrl = `https://${this.assetsDistribution.distributionDomainName}`;

// ============================================================================
// S3 CROSS-REGION REPLICATION (Tier 4+)
// ============================================================================
```

```
if (props.enableReplication && tierConfig.s3.replicationEnabled) {
  // Note: Cross-region replication requires destination buckets in other regions
  // This would be handled by the multi-region deployment setup

  // Create replication role
  const replicationRole = new iam.Role(this, 'ReplicationRole', {
    assumedBy: new iam.ServicePrincipal('s3.amazonaws.com'),
    description: 'Role for S3 cross-region replication',
  });

  replicationRole.addToPolicy(new iam.PolicyStatement({
    actions: [
      's3:GetReplicationConfiguration',
      's3:ListBucket',
    ],
    resources: [this.mediaBucket.bucketArn],
  }));

  replicationRole.addToPolicy(new iam.PolicyStatement({
    actions: [
      's3:GetObjectVersionForReplication',
      's3:GetObjectVersionAcl',
      's3:GetObjectVersionTagging',
    ],
    resources: [`${this.mediaBucket.bucketArn}/*`],
  }));

  // Note: Destination bucket permissions would be added when those buckets are created
}

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

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

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

new cdk.CfnOutput(this, 'MediaBucketName', {
  value: this.mediaBucket.bucketName,
  description: 'Media S3 bucket name',
  exportName: `${resourcePrefix}-media-bucket`,
```

```
    });

    new cdk.CfnOutput(this, 'MediaBucketArn', {
      value: this.mediaBucket.bucketArn,
      description: 'Media S3 bucket ARN',
      exportName: `${resourcePrefix}-media-bucket-arn`,
    });

    new cdk.CfnOutput(this, 'AssetsBucketName', {
      value: this.assetsBucket.bucketName,
      description: 'Assets S3 bucket name',
      exportName: `${resourcePrefix}-assets-bucket`,
    });

    new cdk.CfnOutput(this, 'MediaDistributionId', {
      value: this.mediaDistribution.distributionId,
      description: 'Media CloudFront distribution ID',
      exportName: `${resourcePrefix}-media-cf-id`,
    });

    new cdk.CfnOutput(this, 'MediaDistributionUrl', {
      value: this.mediaUrl,
      description: 'Media CloudFront URL',
      exportName: `${resourcePrefix}-media-url`,
    });

    new cdk.CfnOutput(this, 'AssetsDistributionId', {
      value: this.assetsDistribution.distributionId,
      description: 'Assets CloudFront distribution ID',
      exportName: `${resourcePrefix}-assets-cf-id`,
    });

    new cdk.CfnOutput(this, 'AssetsDistributionUrl', {
      value: this.assetsUrl,
      description: 'Assets CloudFront URL',
      exportName: `${resourcePrefix}-assets-url`,
    });
  }
}
```

PART 9: STACK EXPORTS

packages/infrastructure/lib/stacks/index.ts

```
export * from './foundation.stack';
export * from './networking.stack';
export * from './security.stack';
```

```typescript
export * from './data.stack';
export * from './storage.stack';
// Prompt 3 exports
// export * from './ai.stack';
// export * from './api.stack';
// export * from './admin.stack';
```

packages/infrastructure/lib/index.ts

```typescript
export * from './config';
export * from './stacks';
// export * from './constructs';  // When constructs are added
```

DEPLOYMENT COMMANDS

Development Environment (Tier 1)

```
cd packages/infrastructure

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

# Deploy specific stack
npx cdk deploy thinktank-dev-networking \
  --context appId=thinktank \
  --context environment=dev \
  --context tier=1
```

Staging Environment (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
```

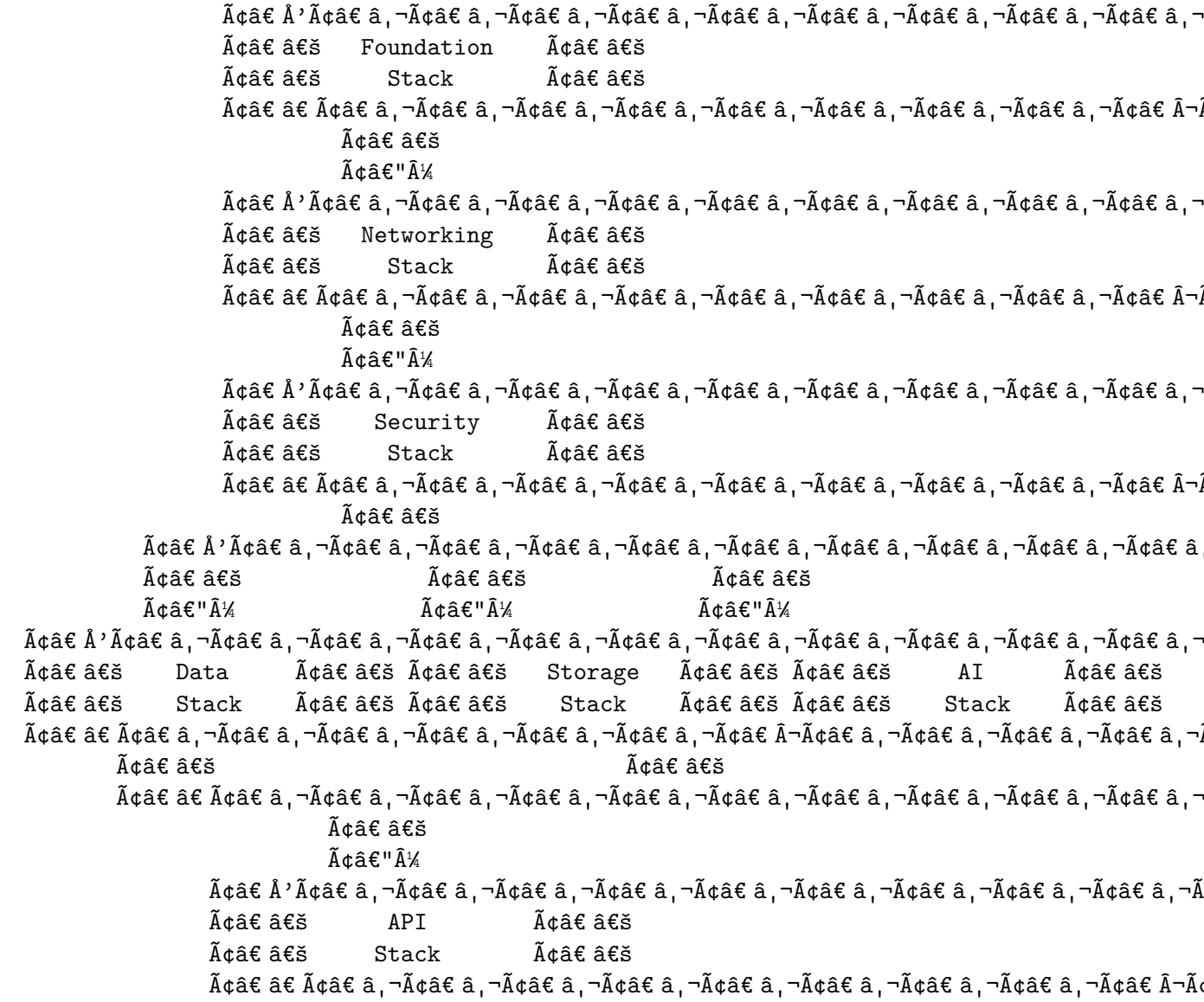
Production Environment (Tier 3-5)

```
# Tier 3 - Single Region
npx cdk deploy --all \
  --context appId=thinktank \
  --context appName="Think Tank" \
```

```
  --context environment=prod \
  --context tier=3 \
  --context domain=thinktank.YOUR_DOMAIN.com

# Tier 4 - Multi-Region
npx cdk deploy --all \
  --context appId=thinktank \
  --context appName="Think Tank" \
  --context environment=prod \
  --context tier=4 \
  --context domain=thinktank.YOUR_DOMAIN.com
```

STACK DEPENDENCIES



```
          Ã¢â€ â€š
          Ã¢â€"Â¼
Ã¢â€ Å'Ã¢â€ â,¬Ã¢â€ â,¬Ã¢â€ â,¬Ã¢â€ â,¬Ã¢â€ â,¬Ã¢â€ â,¬Ã¢â€ â,¬Ã¢â€ â,¬Ã
Ã¢â€ â€š     Admin       Ã¢â€ â€š
Ã¢â€ â€š     Stack       Ã¢â€ â€š
Ã¢â€ â€ Ã¢â€ â,¬Ã¢â€ â,¬Ã¢â€ â,¬Ã¢â€ â,¬Ã¢â€ â,¬Ã¢â€ â,¬Ã¢â€ â,¬Ã¢â€ â,¬Ã
```

NEXT PROMPTS

Continue with: - **Prompt 3**: CDK AI & API Stacks (LiteLLM, SageMaker, API Gateway, Cognito) - **Prompt 4**: Lambda Functions - Core (Router, Providers, Models, Chat) - **Prompt 5**: Lambda Functions - Admin & Billing (Invitations, Approvals, Metering) - **Prompt 6**: Self-Hosted Models & Mid-Level Services - **Prompt 7**: External Providers & Database Schema/Migrations - **Prompt 8**: Admin Web Dashboard (Next.js) - **Prompt 9**: Assembly & Deployment Guide

---

End of Prompt 2: CDK Infrastructure Stacks RADIANT v2.2.0 - December 2024

â•

END OF SECTION 2

â•

â•