Contents

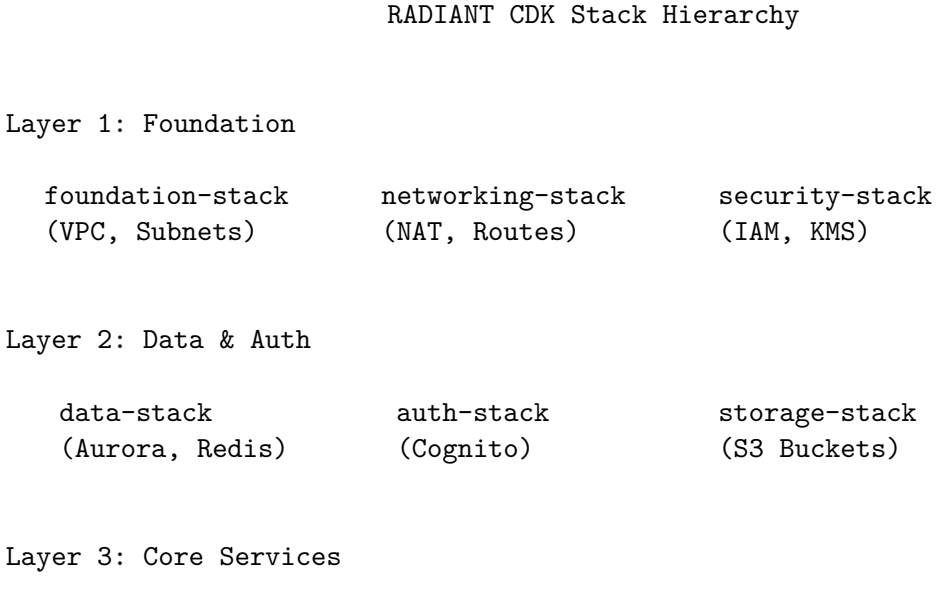
RADIANT v4.18.0 - CDK Infrastructure Stacks Export

Component: AWS CDK Infrastructure **Language**: TypeScript **Files**: 30 CDK stacks **Framework**: AWS CDK v2

Architecture Narrative

RADIANT's infrastructure is defined using **AWS CDK (Cloud Development Kit)**, enabling Infrastructure as Code with TypeScript. The platform deploys 30 interconnected stacks that provision all AWS resources needed for the multi-tenant AI platform.

Stack Dependency Graph



```
    api-stack              admin-stack          brain-stack
    (API Gateway)          (Admin APIs)         (AGI Brain)


Layer 4: AI Services

    ai-stack               grimoire-stack       consciousness-
    (LiteLLM)              (Proc Memory)        stack (Ego)


Layer 5: Advanced Features

  cato-redis-stack       formal-reasoning       cognition-stack
  (Safety)               -stack (Logic)         (Metacognition)


Layer 6: Operations

  monitoring-stack       scheduled-tasks        multi-region
  (CloudWatch)           -stack (Events)        -stack (DR)
```

Stack Inventory

Stack	Lines	Purpose
`admin-stack.ts`	26K	Admin dashboard APIs and Lambda handlers
`ai-stack.ts`	6K	AI model integration with LiteLLM
`api-stack.ts`	34K	Public REST API Gateway and Lambda
`auth-stack.ts`	6K	Cognito user pools and authentication
`batch-stack.ts`	8K	AWS Batch for heavy compute jobs
`cato-genesis-stack.ts`	12K	Cato consciousness genesis
`cato-tier-transition-stack.ts`	17K	Tenant tier upgrade automation
`brain-stack.ts`	10K	AGI Brain Router infrastructure
`cato-redis-stack.ts`	4K	Cato Safety with Redis state
`cognition-stack.ts`	7K	Metacognition and introspection
`collaboration-stack.ts`	8K	Real-time collaboration (WebSocket)
`consciousness-stack.ts`	25K	Consciousness Engine (MCP, Sleep)

Stack	Lines	Purpose
data-stack.ts	5K	Aurora PostgreSQL and DynamoDB
formal-reasoning-stack.ts	16K	Formal logic and verification
foundation-stack.ts	2K	Base VPC and networking
grimoire-stack.ts	7K	Procedural memory (Grimoire)
library-execution-stack.ts	11K	Library execution runtime
library-registry-stack.ts	7K	AI library registry
mission-control-stack.ts	13K	Deployment orchestration
model-sync-scheduler-stack.ts	4K	Model sync scheduling
monitoring-stack.ts	13K	CloudWatch and alarms
multi-region-stack.ts	8K	Multi-region DR
networking-stack.ts	3K	NAT Gateways and routing
scheduled-tasks-stack.ts	13K	EventBridge scheduled tasks
security-monitoring-stack.ts	13K	Security monitoring (GuardDuty)
security-stack.ts	6K	IAM roles and KMS keys
storage-stack.ts	5K	S3 buckets and lifecycle
tms-stack.ts	17K	Tenant Management System
user-registry-stack.ts	14K	User registry and profiles
webhooks-stack.ts	9K	Webhook delivery system

Key Stack Implementations

1. api-stack.ts

Purpose: Defines the main REST API Gateway with Lambda integrations for all public endpoints.

```typescript
import * as cdk from 'aws-cdk-lib';
import * as apigateway from 'aws-cdk-lib/aws-apigateway';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as cognito from 'aws-cdk-lib/aws-cognito';
import * as logs from 'aws-cdk-lib/aws-logs';
import * as iam from 'aws-cdk-lib/aws-iam';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
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 { Construct } from 'constructs';
import type { TierConfig, Environment } from '@radiant/shared';
import { RADIANT_VERSION } from '@radiant/shared';

export interface ApiStackProps extends cdk.StackProps {
  appId: string;
  environment: Environment;
  tier: number;
  tierConfig: TierConfig;
```

```
  vpc: ec2.Vpc;
  userPool: cognito.UserPool;
  adminUserPool: cognito.UserPool;
  auroraCluster: rds.DatabaseCluster;
  usageTable: dynamodb.Table;
  sessionsTable: dynamodb.Table;
  cacheTable: dynamodb.Table;
  mediaBucket: s3.Bucket;
  litellmUrl: string;
  apiSecurityGroup: ec2.SecurityGroup;
  sagemakerRoleArn?: string;
  catoRedisEndpoint?: string;
  catoRedisPort?: number;
}

export class ApiStack extends cdk.Stack {
  public readonly api: apigateway.RestApi;
  public readonly routerFunction: lambda.Function;

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

    const {
      appId, environment, tier, vpc, userPool, adminUserPool,
      auroraCluster, usageTable, sessionsTable, cacheTable, mediaBucket,
      litellmUrl, apiSecurityGroup,
    } = props;

    // Lambda execution role
    const lambdaRole = new iam.Role(this, 'LambdaRole', {
      assumedBy: new iam.ServicePrincipal('lambda.amazonaws.com'),
      managedPolicies: [
        iam.ManagedPolicy.fromAwsManagedPolicyName('service-role/AWSLambdaVPCAccessExecutionRo
      ],
    });

    // Common Lambda environment
    const commonEnv: Record<string, string> = {
      APP_ID: appId,
      ENVIRONMENT: environment,
      TIER: tier.toString(),
      LITELLM_URL: litellmUrl,
      AURORA_SECRET_ARN: auroraCluster.secret?.secretArn || '',
      AURORA_CLUSTER_ARN: auroraCluster.clusterArn,
      USAGE_TABLE: usageTable.tableName,
      SESSIONS_TABLE: sessionsTable.tableName,
      CACHE_TABLE: cacheTable.tableName,
      MEDIA_BUCKET: mediaBucket.bucketName,
```

```
    USER_POOL_ID: userPool.userPoolId,
    ADMIN_USER_POOL_ID: adminUserPool.userPoolId,
    LOG_LEVEL: environment === 'prod' ? 'info' : 'debug',
    RADIANT_VERSION: RADIANT_VERSION,
    ...(props.catoRedisEndpoint ? {
      CATO_REDIS_ENDPOINT: props.catoRedisEndpoint,
      CATO_REDIS_PORT: String(props.catoRedisPort || 6379),
    } : {}),
  };

  // Router Lambda
  this.routerFunction = new lambda.Function(this, 'RouterFunction', {
    functionName: `${appId}-${environment}-router`,
    runtime: lambda.Runtime.NODEJS_20_X,
    handler: 'api/router.handler',
    code: lambda.Code.fromAsset('lambda/dist'),
    memorySize: 512,
    timeout: cdk.Duration.seconds(30),
    environment: commonEnv,
    vpc,
    vpcSubnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
    securityGroups: [apiSecurityGroup],
    role: lambdaRole,
    tracing: tier >= 2 ? lambda.Tracing.ACTIVE : lambda.Tracing.DISABLED,
    logRetention: logs.RetentionDays.ONE_MONTH,
  });

  // Grant permissions
  auroraCluster.secret?.grantRead(lambdaRole);
  usageTable.grantReadWriteData(lambdaRole);
  sessionsTable.grantReadWriteData(lambdaRole);
  cacheTable.grantReadWriteData(lambdaRole);
  mediaBucket.grantReadWrite(lambdaRole);

  // RDS Data API access
  lambdaRole.addToPrincipalPolicy(new iam.PolicyStatement({
    effect: iam.Effect.ALLOW,
    actions: [
      'rds-data:ExecuteStatement',
      'rds-data:BatchExecuteStatement',
      'rds-data:BeginTransaction',
      'rds-data:CommitTransaction',
      'rds-data:RollbackTransaction',
    ],
    resources: [auroraCluster.clusterArn],
  }));

  // REST API
```

```
this.api = new apigateway.RestApi(this, 'Api', {
  restApiName: `${appId}-${environment}-api`,
  description: `RADIANT API for ${appId} ${environment}`,
  deployOptions: {
    stageName: 'v2',
    throttlingRateLimit: tier >= 3 ? 10000 : 1000,
    throttlingBurstLimit: tier >= 3 ? 5000 : 500,
    loggingLevel: apigateway.MethodLoggingLevel.INFO,
    dataTraceEnabled: environment !== 'prod',
    metricsEnabled: true,
    tracingEnabled: tier >= 2,
  },
  defaultCorsPreflightOptions: {
    allowOrigins: apigateway.Cors.ALL_ORIGINS,
    allowMethods: apigateway.Cors.ALL_METHODS,
    allowHeaders: ['Content-Type', 'Authorization', 'X-Api-Key', 'X-Tenant-Id'],
  },
});

// Cognito Authorizers
const cognitoAuthorizer = new apigateway.CognitoUserPoolsAuthorizer(this, 'CognitoAuthoriz
  cognitoUserPools: [userPool],
  authorizerName: `${appId}-${environment}-authorizer`,
  identitySource: 'method.request.header.Authorization',
});

const adminAuthorizer = new apigateway.CognitoUserPoolsAuthorizer(this, 'AdminAuthorizer',
  cognitoUserPools: [adminUserPool],
  authorizerName: `${appId}-${environment}-admin-authorizer`,
  identitySource: 'method.request.header.Authorization',
});

// Lambda integration
const routerIntegration = new apigateway.LambdaIntegration(this.routerFunction, {
  proxy: true,
});

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

// Health check (no auth)
const health = v2.addResource('health');
health.addMethod('GET', routerIntegration);

// Chat endpoints
const chat = v2.addResource('chat');
chat.addResource('completions').addMethod('POST', routerIntegration, {
  authorizer: cognitoAuthorizer,
```

```
      authorizationType: apigateway.AuthorizationType.COGNITO,
    });

    // Models endpoints
    const models = v2.addResource('models');
    models.addMethod('GET', routerIntegration, {
      authorizer: cognitoAuthorizer,
      authorizationType: apigateway.AuthorizationType.COGNITO,
    });

    // Admin endpoints
    const admin = v2.addResource('admin');
    admin.addProxy({
      defaultIntegration: routerIntegration,
      defaultMethodOptions: {
        authorizer: adminAuthorizer,
        authorizationType: apigateway.AuthorizationType.COGNITO,
      },
    });
  }
}
```

---

2. brain-stack.ts

Purpose: Provisions the AGI Brain Router infrastructure including inference Lambda, reconciliation scheduler, Redis cache, and SQS queues.

```
/**
 * RADIANT v6.0.4 - Brain Stack
 * CDK Stack for AGI Brain infrastructure
 *
 * Provisions:
 * - Brain inference Lambda
 * - Reconciliation Lambda (scheduled)
 * - API Gateway routes
 * - ElastiCache (Redis) for flash facts and ghost cache
 * - SQS queues for async processing
 */

import * as cdk from 'aws-cdk-lib';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as apigateway from 'aws-cdk-lib/aws-apigateway';
import * as events from 'aws-cdk-lib/aws-events';
import * as targets from 'aws-cdk-lib/aws-events-targets';
import * as sqs from 'aws-cdk-lib/aws-sqs';
import * as elasticache from 'aws-cdk-lib/aws-elasticache';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
```

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

export interface BrainStackProps extends cdk.StackProps {
  vpc: ec2.IVpc;
  dbSecurityGroup: ec2.ISecurityGroup;
  dbClusterArn: string;
  dbSecretArn: string;
  environment: string;
  litellmUrl?: string;
}

export class BrainStack extends cdk.Stack {
  public readonly brainApi: apigateway.RestApi;
  public readonly brainInferenceLambda: lambda.Function;
  public readonly reconciliationLambda: lambda.Function;
  public readonly redisCluster: elasticache.CfnCacheCluster;

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

    const { vpc, dbSecurityGroup, dbClusterArn, dbSecretArn, environment } = props;

    // Security Groups
    const brainSecurityGroup = new ec2.SecurityGroup(this, 'BrainSecurityGroup', {
      vpc,
      description: 'Security group for Brain Lambda functions',
      allowAllOutbound: true,
    });

    const redisSecurityGroup = new ec2.SecurityGroup(this, 'RedisSecurityGroup', {
      vpc,
      description: 'Security group for Redis cluster',
      allowAllOutbound: false,
    });

    // Allow Brain Lambda to access Redis
    redisSecurityGroup.addIngressRule(
      brainSecurityGroup,
      ec2.Port.tcp(6379),
      'Allow Brain Lambda to access Redis'
    );

    // ElastiCache Redis
    const redisSubnetGroup = new elasticache.CfnSubnetGroup(this, 'RedisSubnetGroup', {
      description: 'Subnet group for Brain Redis cluster',
      subnetIds: vpc.privateSubnets.map(subnet => subnet.subnetId),
      cacheSubnetGroupName: `radiant-brain-redis-${environment}`,
```

```
});

this.redisCluster = new elasticache.CfnCacheCluster(this, 'BrainRedisCluster', {
  cacheNodeType: environment === 'prod' ? 'cache.r6g.large' : 'cache.t3.micro',
  engine: 'redis',
  numCacheNodes: 1,
  clusterName: `radiant-brain-${environment}`,
  vpcSecurityGroupIds: [redisSecurityGroup.securityGroupId],
  cacheSubnetGroupName: redisSubnetGroup.cacheSubnetGroupName,
  port: 6379,
});

// SQS Queues
const dreamQueue = new sqs.Queue(this, 'DreamQueue', {
  queueName: `radiant-dream-queue-${environment}`,
  visibilityTimeout: cdk.Duration.minutes(15),
  retentionPeriod: cdk.Duration.days(7),
  deadLetterQueue: {
    queue: new sqs.Queue(this, 'DreamDLQ', {
      queueName: `radiant-dream-dlq-${environment}`,
      retentionPeriod: cdk.Duration.days(14),
    }),
    maxReceiveCount: 3,
  },
});

// Lambda Environment Variables
const lambdaEnvironment: Record<string, string> = {
  ENVIRONMENT: environment,
  DB_CLUSTER_ARN: dbClusterArn,
  DB_SECRET_ARN: dbSecretArn,
  REDIS_ENDPOINT: this.redisCluster.attrRedisEndpointAddress,
  REDIS_PORT: this.redisCluster.attrRedisEndpointPort,
  REDIS_URL: `redis://${this.redisCluster.attrRedisEndpointAddress}:${this.redisCluster.at
  DREAM_QUEUE_URL: dreamQueue.queueUrl,
  LOG_LEVEL: environment === 'prod' ? 'info' : 'debug',
  LITELLM_ENDPOINT: props.litellmUrl || 'http://localhost:4000',
  SYSTEM1_MODEL: 'llama3-8b-instruct',
  SYSTEM15_MODEL: 'llama3-8b-instruct',
  SYSTEM2_MODEL: 'llama3-70b-instruct',
};

// Brain Inference Lambda
this.brainInferenceLambda = new lambda.Function(this, 'BrainInferenceLambda', {
  functionName: `radiant-brain-inference-${environment}`,
  runtime: lambda.Runtime.NODEJS_20_X,
  handler: 'brain/inference.handler',
  code: lambda.Code.fromAsset('lambda/dist'),
```

```
      memorySize: environment === 'prod' ? 2048 : 1024,
      timeout: cdk.Duration.seconds(30),
      vpc,
      securityGroups: [brainSecurityGroup],
      environment: lambdaEnvironment,
      tracing: lambda.Tracing.ACTIVE,
    });

    // Reconciliation Lambda
    this.reconciliationLambda = new lambda.Function(this, 'ReconciliationLambda', {
      functionName: `radiant-brain-reconciliation-${environment}`,
      runtime: lambda.Runtime.NODEJS_20_X,
      handler: 'brain/reconciliation.handler',
      code: lambda.Code.fromAsset('lambda/dist'),
      memorySize: 1024,
      timeout: cdk.Duration.minutes(5),
      vpc,
      securityGroups: [brainSecurityGroup],
      environment: lambdaEnvironment,
      tracing: lambda.Tracing.ACTIVE,
    });

    // Run reconciliation every 15 minutes
    const reconciliationRule = new events.Rule(this, 'ReconciliationSchedule', {
      schedule: events.Schedule.rate(cdk.Duration.minutes(15)),
      description: 'Trigger Brain reconciliation job every 15 minutes',
    });
    reconciliationRule.addTarget(new targets.LambdaFunction(this.reconciliationLambda));

    // API Gateway
    this.brainApi = new apigateway.RestApi(this, 'BrainApi', {
      restApiName: `radiant-brain-api-${environment}`,
      description: 'RADIANT AGI Brain API',
      deployOptions: {
        stageName: environment,
        tracingEnabled: true,
        loggingLevel: apigateway.MethodLoggingLevel.INFO,
      },
    });

    // /brain/inference endpoint
    const brainResource = this.brainApi.root.addResource('brain');
    const inferenceResource = brainResource.addResource('inference');
    inferenceResource.addMethod('POST', new apigateway.LambdaIntegration(this.brainInferenceLa
  }
}
```

3. consciousness-stack.ts

Purpose: Deploys the Consciousness Engine infrastructure including MCP Server, Sleep Cycle, Deep Research, and admin APIs.

```typescript
import * as cdk from 'aws-cdk-lib';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as events from 'aws-cdk-lib/aws-events';
import * as targets from 'aws-cdk-lib/aws-events-targets';
import * as apigateway from 'aws-cdk-lib/aws-apigateway';
import * as iam from 'aws-cdk-lib/aws-iam';
import * as secretsmanager from 'aws-cdk-lib/aws-secretsmanager';
import * as sqs from 'aws-cdk-lib/aws-sqs';
import { Construct } from 'constructs';

interface ConsciousnessStackProps extends cdk.StackProps {
  appId: string;
  environment: string;
  apiGateway: apigateway.RestApi;
  lambdaLayer: lambda.LayerVersion;
  dbClusterArn: string;
  dbSecretArn: string;
  databaseName: string;
}

/**
 * Consciousness Engine Stack
 *
 * Deploys the consciousness engine infrastructure:
 * - MCP Server Lambda (Model Context Protocol)
 * - Sleep Cycle Lambda (weekly evolution)
 * - Deep Research Lambda (browser automation)
 * - Thinking Session Lambda (async processing)
 * - Budget Monitor Lambda (cost control)
 * - Admin API endpoints
 */
export class ConsciousnessStack extends cdk.Stack {
  public readonly mcpServerLambda: lambda.Function;
  public readonly sleepCycleLambda: lambda.Function;
  public readonly deepResearchLambda: lambda.Function;
  public readonly thinkingSessionLambda: lambda.Function;
  public readonly adminApiLambda: lambda.Function;
  public readonly consciousnessExecutorLambda: lambda.Function;

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

    const { appId, environment, apiGateway, lambdaLayer, dbClusterArn, dbSecretArn, databaseNa
```

```
// SQS Queues for Async Processing
const thinkingSessionQueue = new sqs.Queue(this, 'ThinkingSessionQueue', {
  queueName: `${appId}-${environment}-thinking-sessions`,
  visibilityTimeout: cdk.Duration.minutes(15),
  retentionPeriod: cdk.Duration.days(7),
  deadLetterQueue: {
    queue: new sqs.Queue(this, 'ThinkingSessionDLQ', {
      queueName: `${appId}-${environment}-thinking-sessions-dlq`,
    }),
    maxReceiveCount: 3,
  },
});

const deepResearchQueue = new sqs.Queue(this, 'DeepResearchQueue', {
  queueName: `${appId}-${environment}-deep-research`,
  visibilityTimeout: cdk.Duration.minutes(30),
  retentionPeriod: cdk.Duration.days(7),
});

// Common Lambda Environment
const commonEnv = {
  NODE_OPTIONS: '--enable-source-maps',
  ENVIRONMENT: environment,
  DB_CLUSTER_ARN: dbClusterArn,
  DB_SECRET_ARN: dbSecretArn,
  DATABASE_NAME: databaseName,
  THINKING_SESSION_QUEUE_URL: thinkingSessionQueue.queueUrl,
  DEEP_RESEARCH_QUEUE_URL: deepResearchQueue.queueUrl,
};

// Consciousness Executor Lambda (Python - 16 Libraries)
this.consciousnessExecutorLambda = new lambda.Function(this, 'ConsciousnessExecutor', {
  functionName: `${appId}-${environment}-consciousness-executor`,
  runtime: lambda.Runtime.PYTHON_3_11,
  handler: 'handler.handler',
  code: lambda.Code.fromAsset('lambda/consciousness-executor'),
  timeout: cdk.Duration.minutes(5),
  memorySize: 4096,
  environment: { ENVIRONMENT: environment },
});

// MCP Server Lambda (Model Context Protocol)
this.mcpServerLambda = new lambda.Function(this, 'MCPServer', {
  functionName: `${appId}-${environment}-consciousness-mcp`,
  runtime: lambda.Runtime.NODEJS_20_X,
  handler: 'consciousness/mcp-server.handler',
  code: lambda.Code.fromAsset('lambda'),
```

```
      layers: [lambdaLayer],
      timeout: cdk.Duration.seconds(60),
      memorySize: 1024,
      environment: {
        ...commonEnv,
        CONSCIOUSNESS_EXECUTOR_ARN: this.consciousnessExecutorLambda.functionArn,
      },
    });

    // Sleep Cycle Lambda (Weekly Evolution)
    this.sleepCycleLambda = new lambda.Function(this, 'SleepCycle', {
      functionName: `${appId}-${environment}-consciousness-sleep`,
      runtime: lambda.Runtime.NODEJS_20_X,
      handler: 'consciousness/sleep-cycle.handler',
      code: lambda.Code.fromAsset('lambda'),
      layers: [lambdaLayer],
      timeout: cdk.Duration.minutes(15),
      memorySize: 2048,
      environment: commonEnv,
    });

    // Schedule: Nightly at 3 AM UTC
    new events.Rule(this, 'SleepCycleSchedule', {
      ruleName: `${appId}-${environment}-consciousness-sleep`,
      schedule: events.Schedule.cron({ minute: '0', hour: '3' }),
      targets: [new targets.LambdaFunction(this.sleepCycleLambda)],
    });
  }
}
```

---

4. grimoire-stack.ts

Purpose: Provisions The Grimoire (procedural memory) and Economic Governor infrastructure.

```
// See GRIMOIRE-GOVERNOR-SOURCE-PART5.md for full implementation
// Key components:
// - Grimoire API Lambda
// - Governor API Lambda
// - Daily cleanup Lambda (EventBridge)
// - Shared Python layer for Flyte dependencies
```

Additional Stacks (Summary)

Stack	Key Resources
admin-stack.ts	Admin API Gateway, 20+ Lambda handlers, IAM roles
auth-stack.ts	Cognito User Pools (user + admin), Identity Pools
batch-stack.ts	AWS Batch compute environment, job definitions
cato-redis-stack.ts	ElastiCache Redis for Cato state
cognition-stack.ts	Metacognition Lambda, introspection APIs
collaboration-stack.ts	WebSocket API, connection management
data-stack.ts	Aurora Serverless v2, DynamoDB tables
formal-reasoning-stack.ts	Z3 solver Lambda, verification APIs
foundation-stack.ts	VPC, subnets, security groups
library-execution-stack.ts	Code execution sandbox Lambda
library-registry-stack.ts	AI library registry Lambda
mission-control-stack.ts	Deployment orchestration Lambda
model-sync-scheduler-stack.ts	Model sync EventBridge rules
monitoring-stack.ts	CloudWatch dashboards, alarms, SNS
multi-region-stack.ts	Global Aurora, Route 53 health checks
networking-stack.ts	NAT Gateways, VPC endpoints
scheduled-tasks-stack.ts	EventBridge scheduled tasks
security-monitoring-stack.ts	GuardDuty, Security Hub
security-stack.ts	KMS keys, IAM policies
storage-stack.ts	S3 buckets, lifecycle policies
tms-stack.ts	Tenant Management System Lambda
user-registry-stack.ts	User profile management
webhooks-stack.ts	Webhook delivery system

Stack Configuration Patterns

Environment Variables

All stacks use consistent environment variable naming: - `ENVIRONMENT` - dev/staging/prod - `DB_CLUSTER_ARN` - Aurora cluster ARN - `DB_SECRET_ARN` - Secrets Manager ARN - `LITELLM_URL` - LiteLLM proxy endpoint - `LOG_LEVEL` - info/debug based on environment

Security Patterns

- VPC placement for all Lambdas
- Security groups with least privilege
- IAM roles with scoped permissions
- Secrets Manager for credentials
- KMS encryption for data at rest

Tier-Based Configuration

- Tier 1-2: Minimal resources, t3.micro instances
- Tier 3-4: Production resources, r6g instances
- Tier 5+: High availability, multi-AZ

This concludes the CDK stacks export. See other export files for Lambda handlers and services.