## Contents

# ADR-005: Circadian Budget Management

## Status

Accepted

## Context

Cato's curiosity is designed to be autonomous and continuous. Without constraints, this creates a runaway cost problem:

### Uncontrolled Curiosity Cost Model

```
Curiosity loop = 1 question + 1 answer + grounding
Average cost per loop = $0.01 (Haiku) to $0.10 (Sonnet with tools)

Continuous operation (24/7):
- 1 loop/second = 86,400 loops/day
- At $0.05 average = $4,320/day = $129,600/month

This exceeds the $500/month target by 260x!
```

Additionally, running curiosity during peak user hours: 1. Competes for model capacity 2. Increases latency for user queries 3. Wastes money on real-time inference (vs. batch pricing)

## Decision

Implement a **circadian rhythm** for Cato with distinct day/night operational modes and hard budget caps.

### Operating Modes

| Mode | Hours (UTC) | Behavior | Budget |
|---|---|---|---|
| **DAY** | 6 AM - 2 AM | Queue curiosity, serve users | $0 exploration |
| **NIGHT** | 2 AM - 6 AM | Batch process exploration | Up to $15/night |
| **EMERGENCY** | Any | Over budget, minimal ops | $0 all activity |

### Budget Hierarchy

```
Monthly Budget: $500 (admin-configurable)
   User Interactions: $400 (80%)
       Real-time inference
       Cache misses
   Autonomous Exploration: $100 (20%)
       Night-mode curiosity: $85
       Tool grounding: $10
       Memory consolidation: $5

Daily Exploration Cap: $15 (prevents single bad night)
```

### Night Mode Benefits

1. **Bedrock Batch API**: 50% discount on batch inference
2. **Lower traffic**: Less competition for resources
3. **Consolidation**: Natural time for memory consolidation
4. **Global timing**: 2-6 AM UTC covers low-traffic worldwide

## Architecture

```
                      Circadian Budget Manager



       Clock          Mode Decider      Enforcer
      (UTC)




       Cost            Budget
      Tracker            Checker
```

```
     DAY MODE          NIGHT MODE          EMERGENCY

     Queue              Process            Serve
     curiosity          queue              from
                        (batch)            cache
```

## Implementation

### TypeScript Service

```
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { DynamoDBDocumentClient, GetCommand, UpdateCommand } from '@aws-sdk/lib-dynamodb';

export enum OperatingMode {
  DAY = 'day',
  NIGHT = 'night',
  EMERGENCY = 'emergency'
}

export interface BudgetConfig {
  monthlyLimit: number;        // Default: $500
  dailyExplorationLimit: number; // Default: $15
  explorationRatio: number;    // Default: 0.20
  nightStartHour: number;      // Default: 2 (2 AM UTC)
  nightEndHour: number;        // Default: 6 (6 AM UTC)
  emergencyThreshold: number;  // Default: 0.90
}

export interface BudgetStatus {
  mode: OperatingMode;
  dailySpend: number;
  monthlySpend: number;
  dailyRemaining: number;
  monthlyRemaining: number;
  canExplore: boolean;
  nextModeChange: Date;
}

export class CircadianBudgetManager {
  private readonly docClient: DynamoDBDocumentClient;
  private readonly configTable: string;
  private readonly costsTable: string;
  private config: BudgetConfig | null = null;
  private lastConfigRefresh: Date | null = null;
```

```
constructor(
  configTable: string = 'cato-config',
  costsTable: string = 'cato-costs',
  region: string = 'us-east-1'
) {
  const client = new DynamoDBClient({ region });
  this.docClient = DynamoDBDocumentClient.from(client);
  this.configTable = configTable;
  this.costsTable = costsTable;
}

async getConfig(): Promise<BudgetConfig> {
  const now = new Date();

  // Refresh config every 5 minutes
  if (
    this.config === null ||
    this.lastConfigRefresh === null ||
    now.getTime() - this.lastConfigRefresh.getTime() > 300000
  ) {
    const response = await this.docClient.send(new GetCommand({
      TableName: this.configTable,
      Key: { pk: 'CONFIG', sk: 'BUDGET' }
    }));

    if (response.Item) {
      this.config = {
        monthlyLimit: response.Item.monthlyLimit ?? 500,
        dailyExplorationLimit: response.Item.dailyExplorationLimit ?? 15,
        explorationRatio: response.Item.explorationRatio ?? 0.20,
        nightStartHour: response.Item.nightStartHour ?? 2,
        nightEndHour: response.Item.nightEndHour ?? 6,
        emergencyThreshold: response.Item.emergencyThreshold ?? 0.90
      };
    } else {
      // Default config
      this.config = {
        monthlyLimit: 500,
        dailyExplorationLimit: 15,
        explorationRatio: 0.20,
        nightStartHour: 2,
        nightEndHour: 6,
        emergencyThreshold: 0.90
      };
    }

    this.lastConfigRefresh = now;
  }

  return this.config;
}

async getMode(): Promise<OperatingMode> {
  const config = await this.getConfig();
  const { dailySpend, monthlySpend } = await this.getSpendCounters();
  const now = new Date();
  const hour = now.getUTCHours();

  // Check emergency (budget exhausted)
  if (monthlySpend >= config.monthlyLimit * config.emergencyThreshold) {
    return OperatingMode.EMERGENCY;
  }

  // Check daily exploration limit
  if (dailySpend >= config.dailyExplorationLimit) {
    return OperatingMode.DAY; // No exploration but still serving
  }

  // Check time of day
  if (hour >= config.nightStartHour && hour < config.nightEndHour) {
    return OperatingMode.NIGHT;
  }

  return OperatingMode.DAY;
}

async canExplore(): Promise<boolean> {
  const mode = await this.getMode();
  if (mode === OperatingMode.EMERGENCY) {
    return false;
  }

  const config = await this.getConfig();
  const { dailySpend } = await this.getSpendCounters();

  return dailySpend < config.dailyExplorationLimit;
}

async recordCost(
  amount: number,
  category: 'inference' | 'curiosity' | 'grounding' | 'consolidation',
  model: string,
  tokensInput: number = 0,
  tokensOutput: number = 0
): Promise<void> {
    const now = new Date();
    const month = now.toISOString().slice(0, 7); // YYYY-MM
    const day = now.toISOString().slice(0, 10);  // YYYY-MM-DD

    await this.docClient.send(new UpdateCommand({
      TableName: this.costsTable,
      Key: { pk: `COST#${month}`, sk: now.toISOString() },
      UpdateExpression: 'SET #amount = :amount, #category = :category, #model = :model, #day =
      ExpressionAttributeNames: {
        '#amount': 'amount',
        '#category': 'category',
        '#model': 'model',
        '#day': 'day',
        '#tokensIn': 'tokensInput',
        '#tokensOut': 'tokensOutput'
      },
      ExpressionAttributeValues: {
        ':amount': amount,
        ':category': category,
        ':model': model,
        ':day': day,
        ':tokensIn': tokensInput,
        ':tokensOut': tokensOutput
      }
    }));
  }

  async getStatus(): Promise<BudgetStatus> {
    const config = await this.getConfig();
    const mode = await this.getMode();
    const { dailySpend, monthlySpend } = await this.getSpendCounters();
    const canExplore = await this.canExplore();

    // Calculate next mode change
    const now = new Date();
    const hour = now.getUTCHours();
    let nextModeChange: Date;

    if (hour < config.nightStartHour) {
      nextModeChange = new Date(now);
      nextModeChange.setUTCHours(config.nightStartHour, 0, 0, 0);
    } else if (hour < config.nightEndHour) {
      nextModeChange = new Date(now);
      nextModeChange.setUTCHours(config.nightEndHour, 0, 0, 0);
    } else {
      nextModeChange = new Date(now);
      nextModeChange.setUTCDate(nextModeChange.getUTCDate() + 1);
      nextModeChange.setUTCHours(config.nightStartHour, 0, 0, 0);
    }

    return {
      mode,
      dailySpend,
      monthlySpend,
      dailyRemaining: Math.max(0, config.dailyExplorationLimit - dailySpend),
      monthlyRemaining: Math.max(0, config.monthlyLimit - monthlySpend),
      canExplore,
      nextModeChange
    };
  }

  private async getSpendCounters(): Promise<{ dailySpend: number; monthlySpend: number }> {
    // Implementation queries DynamoDB for current spend
    // Aggregates by day and month
    return { dailySpend: 0, monthlySpend: 0 }; // Placeholder
  }
}
```

## Consequences

### Positive

- **Predictable costs**: Hard caps prevent budget overrun
- **Optimal pricing**: Night-mode uses Bedrock batch (50% off)
- **User priority**: Day mode focuses on user interactions
- **Natural rhythm**: Consolidation aligns with low-traffic periods

### Negative

- **Delayed learning**: Curiosity queued until night
- **Global timing**: 2-6 AM UTC may not suit all regions
- **Complexity**: Two operational modes to manage
- **Queue management**: Must handle curiosity queue

## Admin Configuration

The budget manager is admin-configurable via the Radiant Admin dashboard:

| Setting | Default | Range | Description |
|---|---|---|---|
| Monthly Limit | $500 | $100-$10,000 | Total monthly budget |
| Daily Exploration | $15 | $5-$100 | Max daily curiosity spend |
| Night Start | 2 AM | 0-23 | When night mode begins (UTC) |
| Night End | 6 AM | 0-23 | When night mode ends (UTC) |
| Emergency Threshold | 90% | 50-99% | When to enter emergency mode |

## Scaling Budget with Users

As user base grows, budget should scale:

| Users | Monthly Budget | Daily Exploration | Rationale |
|---|---|---|---|
| 0-10K | $500 | $15 | Starting budget |
| 10K-100K | $2,000 | $60 | 4x growth |
| 100K-1M | $10,000 | $300 | Supporting infrastructure |
| 1M-10M | $100,000 | $3,000 | At scale |

## References

- AWS Bedrock Batch Inference
- Circadian Rhythms in AI Systems
- Cost Optimization for ML Workloads