Contents

Cato Cost Optimization Runbook

Current Cost Structure

Cost Breakdown by Component (at 10M users)

Component	On-Demand Cost	Optimized Cost	Savings
Shadow Self (SageMaker)	$275,000/mo	$100,000/mo	64%
Bedrock (Claude)	$130,000/mo	$52,000/mo	60%
OpenSearch Serverless	$90,000/mo	$90,000/mo	0%
DynamoDB Global Tables	$60,000/mo	$60,000/mo	0%
ElastiCache	$36,000/mo	$36,000/mo	0%
Other	$130,000/mo	$100,000/mo	23%
Total	**$721,000/mo**	**$438,000/mo**	**39%**

Optimization Strategies

1. SageMaker Savings Plans

Impact: 64% reduction on Shadow Self compute

How to Implement:

```
# Check current usage
aws ce get-savings-plans-utilization \
  --time-period Start=2024-01-01,End=2024-01-31

# View available plans
aws savingsplans describe-savings-plan-rates \
  --savings-plan-id sp-1234567890abcdef0

# Purchase via Console or API
aws savingsplans create-savings-plan \
  --savings-plan-offering-id <offering-id> \
  --commitment 100000 \
  --savings-plan-type Compute
```

Commitment: 1-year (20% savings) or 3-year (64% savings)

Break-even: Need 10+ ml.g5.2xlarge instances to justify.

---

2. Semantic Caching (86% LLM Cost Reduction)

Target: 86% cache hit rate

Current Performance:

```
curl https://api.cato.thinktank.ai/api/admin/cato/cache/stats
```

Optimization Actions:

1. **Increase cache size** if hit rate $< 80\%$

   ```
   aws elasticache modify-replication-group \
     --replication-group-id cato-cache \
     --cache-node-type cache.r7g.2xlarge
   ```

2. **Adjust similarity threshold** (default: 0.95)
   - Lower to 0.92 for higher hit rate
   - Higher to 0.97 for better quality

3. **Extend TTL** if knowledge is stable
   - Default: 23 hours
   - Increase to 47 hours for stable domains

4. **Selective invalidation** instead of full domain invalidation

---

3. Bedrock Batch API (50% Discount)

Use Case: Night-mode curiosity processing

How It Works: - Submit batch jobs between 2-6 AM UTC - Bedrock processes asynchronously - 50% cost reduction vs. real-time API

Implementation:

```python
import boto3

bedrock = boto3.client('bedrock-runtime')

# Submit batch job
response = bedrock.create_model_invocation_job(
    jobName='cato-curiosity-batch-2024-01-15',
    modelId='anthropic.claude-3-5-sonnet-20241022-v2:0',
    inputDataConfig={
        's3InputDataConfig': {
            's3Uri': 's3://cato-batch/input/curiosity-questions.jsonl'
        }
    },
    outputDataConfig={
        's3OutputDataConfig': {
            's3Uri': 's3://cato-batch/output/'
        }
    }
)
```

Savings: ~$65,000/month at scale

---

4. Spot Instances for Background Processing

Use Case: Curiosity processing, memory consolidation

Savings: 70% on compute

Risk: Interruption (acceptable for batch)

Implementation:

```yaml
# EKS spot node group
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: cato-eks
managedNodeGroups:
  - name: spot-curiosity
    instanceTypes: ["m5.xlarge", "m5a.xlarge", "m5n.xlarge"]
    spot: true
    minSize: 0
    maxSize: 20
    labels:
      workload: curiosity
    taints:
      - key: spot
```

```
        value: "true"
        effect: NoSchedule
```

---

5. FP8 Quantization for Shadow Self

Impact: 50% reduction in GPU memory and compute

How It Works: - Llama-3-8B uses 16-bit weights (16GB) - FP8 reduces to 8GB - 50% fewer GPU instances needed

Implementation:

```
from transformers import AutoModelForCausalLM
import torch

model = AutoModelForCausalLM.from_pretrained(
    "meta-llama/Meta-Llama-3-8B-Instruct",
    torch_dtype=torch.float8_e4m3fn,  # FP8
    device_map="auto"
)
```

Trade-off: Minor quality degradation (~1% on benchmarks)

---

6. Right-Sizing Instances

Monthly Review Process:

1. **Check utilization:**

   ```
   aws compute-optimizer get-ec2-instance-recommendations
   ```

2. **Common findings:**

   - SageMaker instances underutilized → reduce count
   - ElastiCache oversized → downgrade node type
   - EKS nodes too large → use smaller instances

3. **Target utilization:**

   - GPU: 70-80%
   - CPU: 60-70%
   - Memory: 70-80%

---

7. DynamoDB Optimization

On-Demand vs. Provisioned: - On-demand: Good for variable/unpredictable load - Provisioned: 20% cheaper for steady load

When to switch to provisioned: - RCU/WCU stable for 30+ days - Predictable traffic patterns

Enable DAX caching:

```
# DAX provides sub-ms reads, reduces RCU
aws dax create-cluster \
  --cluster-name cato-dax \
  --node-type dax.r5.large \
  --replication-factor 3
```

---

8. Bedrock Prompt Caching

Impact: 90% token cost reduction for repeated system prompts

How It Works: - Cato's system prompt is ~2000 tokens - Cache it with `cache_control: ephemeral` - Pay full price once, 10% thereafter

Implementation:

```
response = bedrock.invoke_model(
    modelId='anthropic.claude-3-5-sonnet-20241022-v2:0',
    body={
        "anthropic_version": "bedrock-2023-05-31",
        "max_tokens": 1024,
        "system": [
            {
                "type": "text",
                "text": CATO_SYSTEM_PROMPT,
                "cache_control": {"type": "ephemeral"}
            }
        ],
        "messages": [...]
    }
)
```

Cost Monitoring

Daily Checks

1. **Check budget status:**

   ```
   curl https://api.cato.thinktank.ai/api/admin/cato/budget/status
   ```

2. **Check Cost Explorer:**

   ```
   aws ce get-cost-and-usage \
     --time-period Start=2024-01-14,End=2024-01-15 \
     --granularity DAILY \
     --metrics UnblendedCost \
     --filter '{"Dimensions":{"Key":"SERVICE","Values":["Amazon SageMaker"]}}'
   ```

Weekly Review

1. Check Savings Plans utilization
2. Review instance right-sizing recommendations
3. Analyze cache hit rate trends
4. Review budget burn rate

Monthly Actions

1. Right-size instances based on utilization
2. Evaluate Savings Plans renewal/purchase
3. Review architecture for optimization opportunities
4. Update cost projections

Cost Alerts

Configure alerts in AWS Budgets:

Alert	Threshold	Action
Daily spend	> $5,000	Slack notification
Weekly spend	> $30,000	Email + Slack
Monthly forecast	> 110% budget	PagerDuty
Anomaly detection	> 20% spike	Slack + investigation

```
aws budgets create-budget \
  --account-id $AWS_ACCOUNT_ID \
  --budget file://budget.json \
  --notifications-with-subscribers file://notifications.json
```

Emergency Cost Reduction

If costs are spiraling:

Immediate Actions (< 1 hour)

1. **Enable emergency mode:**

   ```
   curl -X PUT \
     -H "Content-Type: application/json" \
     -d '{"emergencyThreshold": 0.5}' \
     https://api.cato.thinktank.ai/api/admin/cato/budget/config
   ```

2. **Disable curiosity processing:**

   ```
   curl -X PUT \
     -d '{"dailyExplorationLimit": 0}' \
     https://api.cato.thinktank.ai/api/admin/cato/budget/config
   ```

3. **Scale down non-critical components:**

```
kubectl scale deployment cato-curiosity -n cato --replicas=0
```

Short-term Actions (< 1 day)

1. Scale down Shadow Self instances
2. Reduce Bedrock calls (lower quality threshold)
3. Increase cache TTL
4. Disable non-critical features

Medium-term Actions (< 1 week)

1. Purchase Reserved Capacity / Savings Plans
2. Implement additional caching layers
3. Optimize query patterns
4. Right-size all resources

Contact

- **Cost questions:** #cato-costs
- **Budget alerts:** #cato-oncall
- **Finance review:** finance@thinktank.ai