Contents

Cato Migration Runbook: LiteLLM → vLLM + Ray Serve

Overview

This runbook covers the 16-week migration from LiteLLM to the new hybrid orchestration architecture.

Why Migrate: - LiteLLM has ~504 concurrent request limit - Cannot scale to 10MM+ users - No stateful context management - No hidden state extraction

Target Architecture: - vLLM on SageMaker for self-hosted inference - Ray Serve on EKS for stateful orchestration - Bedrock for managed Claude models

Prerequisites

Before starting migration:

- ☐ EKS cluster provisioned
- ☐ SageMaker endpoints deployed
- ☐ DynamoDB tables created
- ☐ ElastiCache cluster running
- ☐ Ray Serve deployment tested
- ☐ Feature flags infrastructure ready

Migration Phases

Phase 1: Infrastructure (Weeks 1-4)

Week 1: Deploy vLLM on SageMaker

```bash
# Build custom container
cd infrastructure/docker/shadow-self
docker build -t cato-shadow-self:v1 .

# Push to ECR
aws ecr get-login-password | docker login --username AWS --password-stdin $ECR_REPO
docker push $ECR_REPO/cato-shadow-self:v1

# Deploy endpoint
python scripts/deploy_sagemaker.py --model shadow-self --environment staging
```

Validation:

```bash
# Test endpoint
curl -X POST https://runtime.sagemaker.us-east-1.amazonaws.com/endpoints/cato-shadow-self-stag
  -H "Content-Type: application/json" \
  -d '{"inputs": "Hello, Cato!"}'
```

Week 2: Deploy NLI Model

```bash
# Deploy DeBERTa-MNLI on SageMaker MME
python scripts/deploy_sagemaker.py --model nli --environment staging
```

Week 3: Deploy Ray Serve on EKS

```bash
# Install Ray on EKS
helm install ray-cluster ray/ray-cluster \
  --namespace cato \
  --values k8s/ray-serve/values.yaml

# Deploy Cato orchestrator
kubectl apply -f k8s/ray-serve/cato-orchestrator.yaml
```

Week 4: Integration Testing

```bash
# Run integration tests
pytest tests/integration/test_orchestration.py -v

# Load test
locust -f tests/load/locustfile.py --host=https://staging.cato.thinktank.ai
```

Exit Criteria: - [ ] Shadow Self responding with hidden states - [ ] NLI classifying correctly - [ ] Ray Serve routing working - [ ] $<$ 500ms p99 latency

---

Phase 2: Custom Orchestration (Weeks 5-8)

Week 5: Implement Model Routing

```bash
# Verify routing logic
async def test_routing():
```

```
    orchestrator = CatoOrchestrator()

    # Simple query → Haiku
    decision = orchestrator._determine_routing("What is 2+2?", "general")
    assert decision.primary_model == ModelTier.BEDROCK_HAIKU

    # Complex query → Sonnet
    decision = orchestrator._determine_routing(
        "Explain the implications of quantum computing on cryptography",
        "general"
    )
    assert decision.primary_model == ModelTier.BEDROCK_SONNET
```

Week 6: Implement Stateful Context

```
# Test context persistence
async def test_context():
    orchestrator = CatoOrchestrator()

    # First message
    result1 = await orchestrator.route_query(
        "My name is Alice",
        session_id="test-123"
    )

    # Second message (should remember name)
    result2 = await orchestrator.route_query(
        "What is my name?",
        session_id="test-123"
    )

    assert "Alice" in result2["response"]
```

Week 7: Implement Circuit Breaker

```
# Test circuit breaker
async def test_circuit_breaker():
    orchestrator = CatoOrchestrator()

    # Simulate failures
    for _ in range(5):
        orchestrator.circuit_breakers[ModelTier.BEDROCK_SONNET].record_failure()

    # Should be open
    assert not orchestrator.circuit_breakers[ModelTier.BEDROCK_SONNET].can_execute()

    # Should fall back to Haiku
    result = await orchestrator.route_query("Hello", "test")
```

```
    assert result["model"] == "bedrock_haiku"
```

Week 8: Integrate Semantic Cache

```
# Test cache integration
async def test_cache():
    orchestrator = CatoOrchestrator()
    cache = SemanticCacheService()

    # First query (cache miss)
    result1 = await orchestrator.route_query("What is Python?", "test")
    assert not result1["cached"]

    # Second similar query (cache hit)
    result2 = await orchestrator.route_query("What is Python programming?", "test")
    assert result2["cached"]
```

Exit Criteria: - [ ] Routing logic validated - [ ] Context persists across turns - [ ] Circuit breaker activates on failures - [ ] Cache hit rate $> 80\%$

Phase 3: State Management (Weeks 9-12)

Week 9: Deploy DynamoDB Global Tables

```
# Apply Terraform
cd infrastructure/terraform/environments/production
terraform apply -target=aws_dynamodb_table.semantic_memory
terraform apply -target=aws_dynamodb_table.working_memory
```

Week 10: Deploy OpenSearch Serverless

```
terraform apply -target=aws_opensearchserverless_collection.episodic_memory
```

Week 11: Build Memory Consolidation Pipeline

```
# Test consolidation
async def test_consolidation():
    memory = GlobalMemoryService()

    # Store interaction
    await memory.storeInteraction({
        "query": "What is machine learning?",
        "response": "Machine learning is...",
        "domain": "ai"
    })

    # Consolidate (extract facts)
    facts = await consolidation_pipeline.extract_facts(interaction)
```

```
    # Verify facts stored
    stored = await memory.getFactsByDomain("ai")
    assert len(stored) > 0
```

Week 12: Integration Testing

```
# Full integration test
pytest tests/integration/test_memory.py -v

# Verify global table replication
aws dynamodb describe-table --table-name cato-semantic-memory \
  --query 'Table.Replicas'
```

Exit Criteria: - [ ] DynamoDB replicating across regions - [ ] OpenSearch indexing episodic memory - [ ] Consolidation pipeline extracting facts - [ ] Sub-ms reads via DAX

Phase 4: Traffic Migration (Weeks 13-16)

Week 13: Canary Deployment (10%)

```
# Enable feature flag for 10% of traffic
aws appconfig create-hosted-configuration-version \
  --application-id cato \
  --configuration-profile-id orchestration \
  --content '{"use_new_orchestrator": {"percentage": 10}}'

# Monitor
watch -n 5 'curl -s https://api.cato.thinktank.ai/api/admin/cato/health'
```

Metrics to Monitor: - Latency p50, p99 - Error rate - Cache hit rate - User satisfaction

Week 14: Gradual Rollout (50%)

```
# Increase to 50%
aws appconfig create-hosted-configuration-version \
  --content '{"use_new_orchestrator": {"percentage": 50}}'
```

Rollback Trigger: - Error rate > 1% - Latency p99 > 2s - User complaints spike

Rollback Command:

```
aws appconfig create-hosted-configuration-version \
  --content '{"use_new_orchestrator": {"percentage": 0}}'
```

Week 15: Full Rollout (100%)

```
# Full rollout
aws appconfig create-hosted-configuration-version \
  --content '{"use_new_orchestrator": {"percentage": 100}}'
```

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

Week 16: Decommission LiteLLM

```
# Stop LiteLLM service
kubectl scale deployment litellm -n cato --replicas=0

# After 1 week of monitoring, delete
kubectl delete deployment litellm -n cato

# Archive configuration
git mv config/litellm.yaml config/archive/litellm.yaml.deprecated
git commit -m "Decommission LiteLLM - migration complete"
```

Exit Criteria: - [ ] 100% traffic on new architecture - [ ] No LiteLLM dependencies - [ ] Documentation updated - [ ] Team trained on new system

---

Rollback Procedures

Quick Rollback (< 5 minutes)

```
# Revert feature flag
aws appconfig create-hosted-configuration-version \
  --content '{"use_new_orchestrator": {"percentage": 0}}'

# LiteLLM immediately handles all traffic
```

Full Rollback (< 1 hour)

```
# Scale up LiteLLM
kubectl scale deployment litellm -n cato --replicas=10

# Disable new orchestrator
kubectl scale deployment cato-orchestrator -n cato --replicas=0

# Revert feature flag
aws appconfig create-hosted-configuration-version \
  --content '{"use_new_orchestrator": {"percentage": 0}}'
```

Communication Plan

Week 1 (Kickoff)

- Announce migration timeline to team
- Share runbook with on-call

Week 8 (Mid-point)

- Status update to stakeholders
- Risk assessment review

Week 13 (Canary Start)

- Alert on-call team
- Prepare rollback procedures

Week 16 (Completion)

- Announce completion
- Schedule retrospective
- Update documentation

Success Metrics

Metric	Before	Target	Actual
Max concurrent requests	504	50,000+	
p99 latency	1.5s	< 1s	
Context persistence	No	Yes	
Hidden state extraction	No	Yes	
Cost efficiency	Baseline	-30%	

Risk Register

Risk	Likelihood	Impact	Mitigation
Performance regression	Medium	High	Extensive load testing
Data loss during migration	Low	Critical	Dual-write during transition
Team unfamiliarity	Medium	Medium	Training sessions
Cost overrun	Medium	Medium	Budget monitoring

Contact

- **Migration Lead:** @migration-lead
- **On-call:** #cato-oncall
- **Escalation:** VP Engineering