Activity: FinOps at Scale for Serverless Applications

Background

RetailNova, a global e-commerce and logistics company, runs dozens of digital products that rely heavily on **serverless computing** (AWS Lambda, Azure Functions, and GCP Functions).

Its cloud footprint grew rapidly over the past three years as more internal teams adopted serverless for:

- API processing
- ETL pipelines
- ML workloads
- Data transformation
- Notifications and messaging
- Image and video processing
- IoT signal ingestion
- Payment and fraud workflows

Today, RetailNova runs **over 150 serverless functions**, with environments spanning **production, staging, and development**.

RetailNova's monthly cloud bill exceeded **$420,000**, with serverless functions contributing ~20% of total spend.
However, leadership believes **at least 30% of serverless costs can be optimized**, due to:

1. Over-provisioned memory

Many functions allocated high memory (e.g., 2–4 GB) but did not require the CPU gains.

2. Spikes in invocation volume

Flash sales and seasonal traffic surged workloads unexpectedly.

3. High data-transfer charges

Functions transferring large outputs across regions/S3 buckets incurred additional fees.

4. Provisioned Concurrency misuse

Some teams enabled provisioned concurrency "just in case," leading to unnecessary fixed costs.

5. Redundant dev/staging functions

Development environments accumulated unused test functions that no one cleaned up.

6. Long-running workloads better suited for containers

ETL functions running 15–70 seconds inflated GB-second usage dramatically.

The FinOps engineering team built the dataset to analyze patterns and provide recommendations across teams.

Dataset:

Field	Description	Useful For Optimization
FunctionName	Name of the serverless function	Identification & grouping
Environment	prod, staging, dev	Environment cost separation
InvocationsPerMonth	Number of times function executed	Helps find unused or over-invoked functions
AvgDurationMs	Avg execution time in milliseconds	Right-sizing memory & CPU
MemoryMB	Configured memory	Cost/performance tradeoff
ColdStartRate	% of invocations causing cold starts	Optimize via provisioned concurrency
ProvisionedConcurrency	Number of PC units allocated	Evaluating PC cost vs cold-start reduction
GBSeconds	Total compute consumption	Directly tied to Lambda cost
DataTransferGB	GB transferred out	Egress cost optimization
CostUSD	Estimated monthly cost	Cost benchmarking

Exercise 1: Identify top cost contributors

- Which functions contribute 80% of total spend?
- Plot cost vs invocation frequency.

Exercise 2: Memory right-sizing

- Look for functions where **duration is low but memory is high**.
- Predict cost impact of lowering memory.

Exercise 3: Provisioned concurrency optimization

- Compare cold start rate vs PC cost.
- Decide whether to reduce or remove provisioned concurrency.

Exercise 4: Detect unused or low-value workloads

- Any function with *<1% of total invocations but high cost*?

Exercise 5: Cost forecasting

- Build a model:
  Cost ≈ Invocations × Duration × Memory × PricingCoefficients + DataTransfer

Exercise 6: Spot workloads that would benefit from containerization

- Long-running (>3s)
- High memory (>2GB)
- Low invocation frequency

Submissions: Create a streamlit dashboard showing the solution for all the exercises above and show the demo