Event-Driven Processing Pipeline

A serverless log processing pipeline built with Go, demonstrating event-driven architecture patterns on AWS. This project compares deployment and performance characteristics between LocalStack (local development) and AWS (production).

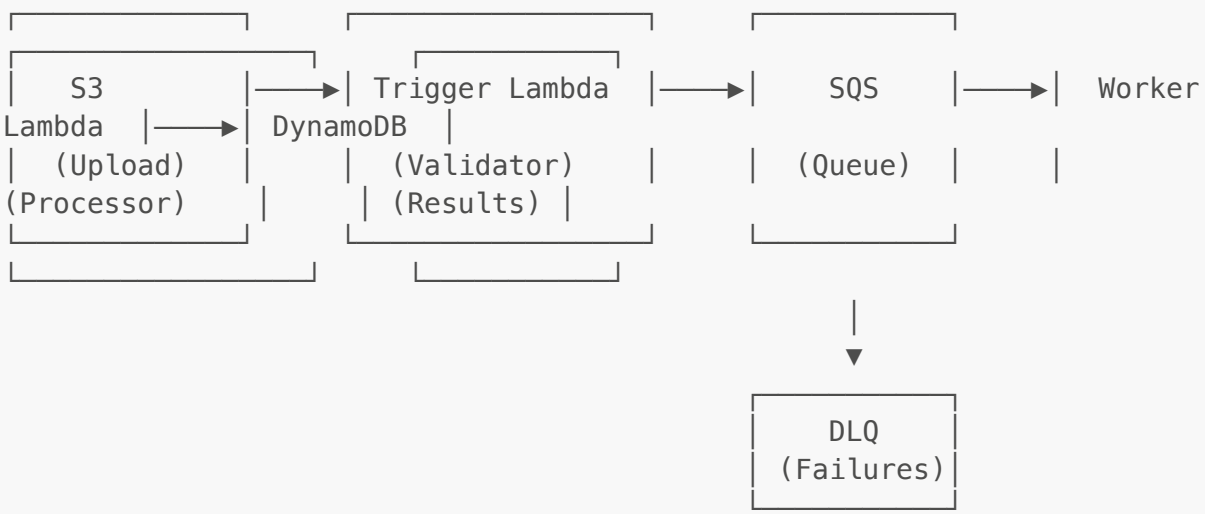
Why Log Processing Pipelines?

Modern distributed systems generate massive volumes of operational data. Organizations process this data to:

- **Detect security incidents** before they escalate
- **Debug production issues** across distributed microservices
- **Meet compliance requirements** (SOX, HIPAA, PCI-DSS audit logging)
- **Optimize costs** by identifying inefficient services
- **Understand user behavior** for product decisions

This serverless architecture processes logs at 70-90% lower cost than commercial solutions (Datadog, Splunk) while maintaining full customization control.

Architecture



Data Flow

1. **Upload**: JSON log files are uploaded to S3 bucket under `logs/` prefix
2. **Trigger**: S3 event notification invokes Trigger Lambda
3. **Validate**: Trigger Lambda validates the file and creates a processing job
4. **Queue**: Job is sent to SQS for decoupled processing
5. **Process**: Worker Lambda downloads file, parses logs, aggregates statistics
6. **Store**: Results are written to DynamoDB with 7-day TTL

Components

Component	Technology	Purpose
Upload Storage	S3	Receives JSON log file uploads
Trigger	Lambda (Go)	Validates uploads, publishes jobs to SQS
Queue	SQS	Decouples ingestion from processing
Worker	Lambda (Go)	Parses logs, aggregates statistics
Results Store	DynamoDB	Stores processing results
Dead Letter Queue	SQS	Captures failed processing attempts

Prerequisites

- Go 1.21+
- Terraform 1.5+
- Docker & Docker Compose
- AWS CLI v2
- Python 3.8+ (for analysis)

Configuration

LocalStack (`infrastructure/terraform/local.tfvars`)

```
environment         = "local"
localstack_endpoint = "http://localhost:4566"
lambda_endpoint     = "http://host.docker.internal:4566"
use_lab_role        = false
lab_role_arn        = ""
```

AWS Learner Lab (`infrastructure/terraform/aws-learner-lab.tfvars`)

```
environment  = "aws"
use_lab_role = true
lab_role_arn = "arn:aws:iam::YOUR_ACCOUNT_ID:role/LabRole"
```

Quick Start

LocalStack Deployment

```
# Start LocalStack and deploy
make localstack-up
make deploy-local

# Test the pipeline
make test-upload-local
```

```
# View results
make view-results-local
```

AWS Learner Lab Deployment

```
# 1. Start your AWS Academy Learner Lab and configure credentials
aws configure
aws configure set aws_session_token <YOUR_TOKEN>
# Or export the environment variables from AWS Details → AWS CLI

# 2. Get your LabRole ARN
aws iam get-role --role-name LabRole --query 'Role.Arn' --output text

# 3. Update infrastructure/terraform/aws-learner-lab.tfvars with your ARN
#    lab_role_arn = "arn:aws:iam::YOUR_ACCOUNT_ID:role/LabRole"

# 4. Deploy
make deploy-aws

# 5. Test the pipeline
make test-upload-aws

# 6. View results
make view-results-aws

# 7. Clean up before session expires
make destroy-aws
```

Project Structure

```
event-pipeline/
├── cmd/                        # Lambda entry points
│   ├── trigger/               # S3 trigger handler
│   └── worker/                # SQS consumer handler
├── internal/                   # Shared internal packages
│   ├── models/                # Data structures
│   ├── processor/             # Log parsing logic
│   └── metrics/               # CloudWatch metrics
├── infrastructure/
│   ├── terraform/             # AWS/LocalStack deployment
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   ├── lambda.tf
│   │   ├── s3.tf
│   │   ├── sqs.tf
│   │   ├── dynamodb.tf
│   │   ├── outputs.tf
│   │   ├── local.tfvars
```

```
│   │   └── aws-learner-lab.tfvars
│   └── localstack/
│       └── docker-compose.yml
├── analysis/                    # Performance analysis
│   ├── run_tests.py            # Test runner
│   ├── generate_charts.py      # Chart generation
│   ├── results/                # Raw test data
│   └── charts/                 # Generated visualizations
├── test/
│   └── data/                   # Test data files
├── docs/
│   └── report.md               # Analysis report
├── build/                       # Compiled Lambda zips (generated)
├── go.mod
├── go.sum
├── Makefile
└── README.md
```

Log File Format

The pipeline processes NDJSON (Newline-Delimited JSON) log files. Each line is a separate JSON object:

```
{"timestamp": "2024-01-15T10:00:00Z", "level": "INFO", "endpoint":
"/api/users", "response_time_ms": 45, "status_code": 200, "user_id":
"user_1"}
{"timestamp": "2024-01-15T10:00:01Z", "level": "ERROR", "endpoint":
"/api/orders", "response_time_ms": 2500, "status_code": 500, "user_id":
"user_2"}
```

Note: Your IDE may show a JSON validation error because it expects a single JSON document. This is expected - NDJSON format is correct for log processing.

Each log entry should contain:

Field	Type	Description
`timestamp`	string	ISO 8601 timestamp
`level`	string	Log level: INFO, WARN, ERROR, DEBUG
`endpoint`	string	API endpoint path
`response_time_ms`	integer	Response time in milliseconds
`status_code`	integer	HTTP status code
`user_id`	string	User identifier

Processing Results

The Worker Lambda aggregates statistics and stores them in DynamoDB:

Field	Description
`job_id`	Unique identifier for the processing job
`status`	"completed" or "failed"
`line_count`	Total number of log lines processed
`error_count`	Count of ERROR level logs
`warn_count`	Count of WARN level logs
`info_count`	Count of INFO level logs
`avg_response_time_ms`	Average response time across all logs
`max_response_time_ms`	Maximum response time
`unique_users`	Number of unique user IDs
`unique_endpoints`	Number of unique endpoints
`processing_time_ms`	Time taken to process the file
`file_size_bytes`	Size of the processed file

Running the Analysis

The analysis compares pipeline performance between LocalStack and AWS. This generates the data and charts used in the report.

1. Run Tests

```
# LocalStack (must have infrastructure deployed first)
make localstack-up
make deploy-local
make run-tests-local     # Runs 10 iterations × 3 file sizes (100, 500,
1000 lines)

# AWS (must have infrastructure deployed first)
make deploy-aws
make run-tests-aws       # Same test matrix against real AWS
```

Each test run:

- Uploads log files of varying sizes to S3
- Measures end-to-end latency (upload → DynamoDB result)
- Records S3 upload time, Lambda processing time, and total duration
- Saves results to `analysis/results/`

2. Generate Charts

```
make generate-charts     # Reads results, outputs PNG charts to
analysis/charts/
```

This produces:

- `latency_comparison.png` — Distribution comparison between environments
- `percentile_comparison.png` — P50/P95/P99 latency breakdown
- `latency_by_filesize.png` — Performance across file sizes
- `component_breakdown.png` — Where time is spent (S3, Lambda, SQS)
- `latency_timeline.png` — Cold start visualization
- `summary_table.png` - Full testing summary

3. Full Analysis (Both Environments)

```
make full-analysis       # Runs tests on LocalStack + AWS, then generates
charts
```

Results are saved to `analysis/results/` and charts to `analysis/charts/`.

All Make Targets

```
make help                # Show all available targets

# Building
make build-lambda        # Build Lambda deployment packages (zip files)
make clean               # Remove build artifacts

# LocalStack
make localstack-up       # Start LocalStack containers
make localstack-down     # Stop LocalStack containers
make deploy-local        # Deploy to LocalStack
make destroy-local       # Destroy LocalStack resources

# AWS
make deploy-aws          # Deploy to AWS Learner Lab
make destroy-aws         # Destroy AWS resources

# Manual Testing
make test-upload-local   # Upload sample file to LocalStack
make test-upload-aws     # Upload sample file to AWS
make view-results-local  # View DynamoDB results (LocalStack)
make view-results-aws    # View DynamoDB results (AWS)

# Analysis
make run-tests-local     # Run systematic tests on LocalStack
make run-tests-aws       # Run systematic tests on AWS
make generate-charts     # Generate analysis charts
```

```
make full-analysis       # Run tests on both environments + charts

# Workflows
make full-local          # Start LocalStack, deploy, and run tests
make clean-all           # Clean artifacts, destroy infra, stop LocalStack

# Utility
make workspace-status    # Show current Terraform workspace
```

Troubleshooting

LocalStack Issues

Container not starting:

```
make localstack-down
make localstack-up
```

Terraform state drift (resources already exist):

```
# Easiest fix: reset LocalStack
make localstack-down
make localstack-up
make deploy-local
```

Lambda "connection refused" error:

- Lambda containers can't reach `localhost` on your host
- Ensure `lambda_endpoint = "http://host.docker.internal:4566"` in local.tfvars

Empty DynamoDB results:

- Check SQS queue for pending messages
- Check DLQ for failed messages
- View Lambda logs: `docker logs localstack-localstack-1 2>&1 | tail -50`

AWS Learner Lab Issues

IAM permission errors:

- Ensure `use_lab_role = true` in aws-learner-lab.tfvars
- Verify `lab_role_arn` contains the correct ARN

Session expired:

- Learner Lab sessions expire after ~4 hours
- Restart lab and reconfigure credentials
- Verify with: `aws sts get-caller-identity`

Always clean up before session ends:

```
make destroy-aws
```

Analysis Report

See `docs/report.md` for the full comparison between LocalStack and AWS deployments, including:

- Latency distribution comparisons (LocalStack 40% faster due to no network overhead)
- Component breakdown (75% of latency is SQS event delivery, not compute)
- Cost analysis at enterprise scale
- Recommendations for when to use each environment