CloudWatch Logs Setup

CloudWatch Logs Setup — Drop Production

Date: 2026-02-22 **Priority:** P0 (Production Blocker) **Effort:** 2 hours **Cost:** ~$5/month (30 GB ingestion)

Overview

AWS App Runner automatically streams application logs (stdout/stderr) to CloudWatch Logs. This setup guide configures **retention policies**, **log insights queries**, and **alarms** for production monitoring.

Prerequisites

- AWS CLI configured with credentials
- App Runner service deployed to `eu-west-1`
- Application writes JSON logs to stdout (already implemented via `src/lib/logger.ts`)

Configuration

1. Set Log Retention Policy

Default: CloudWatch Logs retain forever (expensive) **Recommendation:** 30 days (production), 7 days (staging)

```
# Production: 30 days retention
aws logs put-retention-policy \
```

```
  --log-group-name /aws/apprunner/drop-production \
  --retention-in-days 30 \
  --region eu-west-1

# Staging: 7 days retention
aws logs put-retention-policy \
  --log-group-name /aws/apprunner/drop-staging \
  --retention-in-days 7 \
  --region eu-west-1
```

Verify retention:

```
aws logs describe-log-groups \
  --log-group-name-prefix /aws/apprunner/drop \
  --region eu-west-1 \
  | jq '.logGroups[] | {name: .logGroupName, retention: .retentionInDays}'

# Expected:
# {
#   "name": "/aws/apprunner/drop-production",
#   "retention": 30
# }
```

---

2. Create Log Insights Queries

Purpose: Pre-built queries for common investigations.

Query 1: All Errors (Last Hour)

```
fields @timestamp, level, message, metadata.error, metadata.userId, requestId
| filter level = "error"
| sort @timestamp desc
| limit 100
```

Save as: `drop-errors-last-hour`

Query 2: User Activity Trace

```
fields @timestamp, level, message, metadata.userId, metadata.action, requestId
| filter metadata.userId = "usr_123"
```

```
| sort @timestamp desc
| limit 500
```

Save as: `drop-user-activity-trace`

Query 3: Request Trace by ID

```
fields @timestamp, level, message, metadata
| filter requestId = "req_abc123"
| sort @timestamp asc
```

Save as: `drop-request-trace`

Query 4: API Endpoint Performance

```
fields @timestamp, message, metadata.endpoint, metadata.latencyMs
| filter metadata.latencyMs > 1000
| stats avg(metadata.latencyMs) as avg_latency, max(metadata.latencyMs) as max_latency,
count() as slow_requests by metadata.endpoint
| sort slow_requests desc
```

Save as: `drop-slow-endpoints`

Query 5: Authentication Events

```
fields @timestamp, level, message, metadata.action, metadata.userId, metadata.ip
| filter metadata.action in ["login_success", "login_failure", "logout"]
| sort @timestamp desc
| limit 100
```

Save as: `drop-auth-events`

Query 6: Payment Failures

```
fields @timestamp, level, message, metadata.errorCode, metadata.transactionId, metadata.userId
| filter metadata.errorCode in ["INSUFFICIENT_FUNDS", "PAYMENT_REJECTED", "TIMEOUT"]
| sort @timestamp desc
| limit 50
```

Save as: `drop-payment-failures`

---

3. Create CloudWatch Alarms

Alarm 1: High Error Rate

Metric: Error log entries per minute **Threshold:** >10 errors/minute for 2 consecutive periods
Action: Send SNS notification → Slack webhook

```
# Create metric filter
aws logs put-metric-filter \
  --log-group-name /aws/apprunner/drop-production \
  --filter-name drop-error-count \
  --filter-pattern '{ $.level = "error" }' \
  --metric-transformations \
    metricName=ErrorCount,metricNamespace=Drop/Logs,metricValue=1,unit=Count \
  --region eu-west-1

# Create alarm
aws cloudwatch put-metric-alarm \
  --alarm-name drop-high-error-rate \
  --alarm-description "Alert when error rate exceeds threshold" \
  --metric-name ErrorCount \
  --namespace Drop/Logs \
  --statistic Sum \
  --period 60 \
  --evaluation-periods 2 \
  --threshold 10 \
  --comparison-operator GreaterThanThreshold \
  --treat-missing-data notBreaching \
  --alarm-actions <SNS-TOPIC-ARN> \
  --region eu-west-1
```

Alarm 2: No Logs Received (Service Down)

Metric: Log ingestion stopped **Threshold:** No logs for 5 minutes **Action:** Send SNS notification

```
aws cloudwatch put-metric-alarm \
  --alarm-name drop-no-logs-received \
  --alarm-description "Alert when no logs received (service may be down)" \
  --metric-name IncomingLogEvents \
  --namespace AWS/Logs \
  --dimensions Name=LogGroupName,Value=/aws/apprunner/drop-production \
```

```
  --statistic Sum \
  --period 300 \
  --evaluation-periods 1 \
  --threshold 1 \
  --comparison-operator LessThanThreshold \
  --treat-missing-data breaching \
  --alarm-actions <SNS-TOPIC-ARN> \
  --region eu-west-1
```

Alarm 3: Database Errors

Metric: Database connection errors **Threshold:** >5 DB errors in 5 minutes

```
aws logs put-metric-filter \
  --log-group-name /aws/apprunner/drop-production \
  --filter-name drop-db-errors \
  --filter-pattern '{ $.message = "*database*" && $.level = "error" }' \
  --metric-transformations \
    metricName=DatabaseErrors,metricNamespace=Drop/Logs,metricValue=1,unit=Count \
  --region eu-west-1

aws cloudwatch put-metric-alarm \
  --alarm-name drop-database-errors \
  --metric-name DatabaseErrors \
  --namespace Drop/Logs \
  --statistic Sum \
  --period 300 \
  --evaluation-periods 1 \
  --threshold 5 \
  --comparison-operator GreaterThanThreshold \
  --alarm-actions <SNS-TOPIC-ARN> \
  --region eu-west-1
```

---

4. SNS Topic for Alerts

Create SNS topic (if not exists):

```
aws sns create-topic \
  --name drop-cloudwatch-alerts \
```

```
  --region eu-west-1

# Output:
# {
#   "TopicArn": "arn:aws:sns:eu-west-1:324480209768:drop-cloudwatch-alerts"
# }
```

Subscribe Slack webhook:

```
# Option 1: Email subscription (immediate)
aws sns subscribe \
  --topic-arn arn:aws:sns:eu-west-1:324480209768:drop-cloudwatch-alerts \
  --protocol email \
  --notification-endpoint alem@alai.no \
  --region eu-west-1

# Confirm subscription via email link

# Option 2: Lambda → Slack (requires Lambda function)
# See: infrastructure/cloudwatch-to-slack-lambda.md (future enhancement)
```

---

5. Export Logs to S3 (Compliance/Archival)

Purpose: Long-term storage (>30 days) for compliance, cheaper than CloudWatch.

Create S3 bucket:

```
aws s3 mb s3://drop-logs-archive --region eu-west-1

# Set lifecycle policy (move to Glacier after 90 days)
cat > lifecycle.json <<EOF
{
  "Rules": [
    {
      "Id": "archive-old-logs",
      "Status": "Enabled",
      "Transitions": [
        {
          "Days": 90,
```

```
          "StorageClass": "GLACIER"
        }
      ],
      "Expiration": {
        "Days": 1825
      }
    }
  ]
}
EOF

aws s3api put-bucket-lifecycle-configuration \
  --bucket drop-logs-archive \
  --lifecycle-configuration file://lifecycle.json
```

Create export task (manual, run monthly):

```
# Export last 30 days to S3 (run on day 1 of each month)
START_TIME=$(date -u -d '60 days ago' +%s)000
END_TIME=$(date -u -d '30 days ago' +%s)000

aws logs create-export-task \
  --log-group-name /aws/apprunner/drop-production \
  --from $START_TIME \
  --to $END_TIME \
  --destination drop-logs-archive \
  --destination-prefix logs/$(date +%Y-%m) \
  --region eu-west-1

# Check export status
aws logs describe-export-tasks --region eu-west-1
```

Automate with Lambda (future):

- Schedule Lambda to run monthly
- Export previous month's logs to S3
- Delete from CloudWatch after successful export

Log Format

Current Format (Structured JSON)

Example log entry:

```
{
  "timestamp": "2026-02-22T10:30:45.123Z",
  "level": "info",
  "message": "User logged in",
  "requestId": "req_abc123",
  "metadata": {
    "userId": "usr_456",
    "email": "user@example.com",
    "ip": "1.2.3.4",
    "action": "login_success"
  }
}
```

CloudWatch Logs Insights automatically parses JSON fields, enabling queries like:

```
| filter metadata.userId = "usr_456"
```

Cost Estimate

CloudWatch Logs Pricing (EU-West-1)

- **Ingestion:** $0.50 per GB
- **Storage:** $0.03 per GB/month
- **Log Insights queries:** $0.005 per GB scanned

Expected Usage (Production)

- **Log volume:** ~1 GB/day (30 GB/month)
- **Ingestion cost:** 30 GB × $0.50 = $15/month
- **Storage cost (30-day retention):** 30 GB × $0.03 = $0.90/month
- **Query cost:** ~10 queries/day × 1 GB × $0.005 × 30 = $1.50/month

Total: ~$17/month

Cost Optimization

1. **Reduce log verbosity** (filter debug logs in production):

```
// src/lib/logger.ts
const minLevel = process.env.NODE_ENV === 'production' ? 'info' : 'debug';
```

2. **Use sampling for high-volume events**:

```
if (Math.random() < 0.1) { // Log 10% of requests
  logger.debug('Request details', { ... });
}
```

3. **Export to S3 for long-term storage** ($0.023/GB/month, 23% cheaper)

Querying Logs

Via AWS Console

1. Open CloudWatch Console: https://console.aws.amazon.com/cloudwatch/
2. Navigate to: Logs → Log groups → `/aws/apprunner/drop-production`
3. Click "Search log group" or "Insights queries"
4. Select saved query or write custom query

Via AWS CLI

```
# Run saved query
aws logs start-query \
  --log-group-name /aws/apprunner/drop-production \
  --start-time $(date -u -d '1 hour ago' +%s) \
  --end-time $(date -u +%s) \
  --query-string 'fields @timestamp, level, message | filter level = "error" | sort @timestamp
desc' \
  --region eu-west-1

# Get query results (use queryId from previous command)
aws logs get-query-results --query-id <query-id> --region eu-west-1
```

Via Log Streaming (Real-Time)

```
# Stream logs in real-time (like tail -f)
aws logs tail /aws/apprunner/drop-production \
  --follow \
  --format short \
  --region eu-west-1

# Filter by error level
aws logs tail /aws/apprunner/drop-production \
  --follow \
  --filter-pattern '{ $.level = "error" }' \
  --region eu-west-1
```

Troubleshooting

Issue: No logs appearing in CloudWatch

Diagnosis:

```
# Check if log group exists
aws logs describe-log-groups \
  --log-group-name-prefix /aws/apprunner/drop \
  --region eu-west-1

# Check App Runner service logs integration
aws apprunner describe-service \
  --service-arn <ARN> \
  --region eu-west-1 \
  | jq '.Service.ObservabilityConfiguration'
```

Solution:

- App Runner auto-creates log group on first log output
- Verify app is writing to stdout (not file)
- Check IAM permissions (App Runner role needs `logs:CreateLogStream`, `logs:PutLogEvents`)

Issue: Logs not in JSON format

Diagnosis:

```
# Check log entries
aws logs tail /aws/apprunner/drop-production --format short --region eu-west-1 | head -10
```

Solution:

- Ensure app uses `logger.ts` for all logging (not `console.log`)
- Verify `process.stdout.write(JSON.stringify(entry) + "\n")` is used

Checklist

- ☐ Retention policy set (30 days production, 7 days staging)
- ☐ Log Insights queries saved (6 queries)
- ☐ Metric filters created (error count, DB errors)
- ☐ CloudWatch alarms configured (3 alarms)
- ☐ SNS topic created and subscribed (email/Slack)
- ☐ S3 export bucket created (with lifecycle policy)
- ☐ Cost estimate reviewed and approved
- ☐ Team trained on log querying (AWS Console + CLI)
- ☐ Documentation updated

Next Steps

1. **Deploy retention policies** (run commands above)
2. **Test alarms** (trigger error spike, verify alert received)
3. **Save Log Insights queries** (via AWS Console)
4. **Schedule monthly S3 export** (manual for now, automate later)
5. **Monitor costs** (set billing alert at $20/month)

Related Documentation

- `docs/infrastructure/MONITORING.md` — Overall monitoring setup
- `src/lib/logger.ts` — Structured logging implementation
- `infrastructure/error-tracking-setup.md` — Sentry integration
- AWS CloudWatch Logs docs: https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/

Last Updated: 2026-02-22 **Owner:** John (AI Director)

Revision #7
Created 2026-02-23 11:28:58 UTC by John
Updated 2026-06-28 20:02:24 UTC by John