**Research Report**

Study a minimum of 6 real-world use cases, technical blogs, or whitepapers relevant to event-driven data processing, automated reporting systems, and infrastructure automation.

For each paper, include the following six points:

* Link to the resource
* Overview of the solution
* Key components or technologies used
* Triggering mechanism
* Processing flow
* Automation on and deployment approach

1. **AWS Lambda + Amazon EventBridge**

* **Link:** https://aws.amazon.com/blogs/compute/
* **Overview:** AWS Lambda is a serverless computing solution that eliminates the need to manage servers by executing code automatically in response to events.  The ability to create real-time, event-driven systems for automated workflows possible when used with Amazon EventBridge.  The EventBridge service sends events to Lambda for processing or transformation from AWS services or bespoke apps.  In activities involving messaging, streaming, and storage, Lambda communicates with services such as SQS, and S3.  If combined, they provide a fully controlled, scalable, and economical way to create event-driven systems.
* **Key components or technologies:** AWS Lambda, Amazon EventBridge, SQS, CloudWatch, S3.
* **Triggering mechanism:** EventBridge trigger Lambda functions when events arrive.
* **Processing flow:** Events -> Event Router (EventBridge) -> Lambda (process) -> store results in Redshift -> downstream reports.
* **Automation & deployment approach:** Use Terraform to provision Lambda, EventBridge rules, IAM roles, and use GitHub Actions or CodePipeline for CI/CD to run terraform plan.

1. **Apache Kafka + Stream Processing**

* **Link:** https://kafka.apache.org
* **Overview:** Apache Kafka is a distributed event streaming technology designed for large-scale, real-time, high-throughput data processing.  It enables fault-tolerant and scalable posting, storing, and consumption of data streams.  It facilitates data pipeline integration between microservices, log aggregation, and real-time analytics. It makes stream processing and smooth system interaction possible with Kafka Streams and Connect.  All things considered, Kafka supports event-driven, scalable, and dependable architectures for contemporary data applications.
* **Key components or technologies:** Kafka brokers, Kafka Connect, Kafka Streams, producers and consumers, Schema Registry.
* **Triggering mechanism:** Producers write messages to topics; consumers process messages in near-real-time.
* **Processing flow:** Producers -> Kafka topics -> stream processors (filtering, aggregation) -> sinks (S3 or Databases) -> daily report generation.
* **Automation & deployment approach:** Deploy Kafka via Kubernetes (Helm) or managed service (Confluent Cloud). IaC: Helm charts. CI/CD: pipeline builds stream apps, runs integration tests, deploys to k8s.

1. **Azure Event Grid + Functions + Data Factory**

* **Link:** https://learn.microsoft.com/azure/event-grid
* **Overview:** A fully managed, event-driven data processing solution is made possible by the combination of Azure Event Grid, Functions, and Data Factory.  The Event Grid automatically initiates downstream activities by routing events from sources such as storage or the Internet of Things.  Data transformation and validation are examples of real-time ETL processes that Azure Functions handles. Data Factory orchestrates pipelines and analytics, while Blob or Data Lake stores processed data.  For real-time data operations, this integration provides a serverless, scalable, and economical architecture.
* **Key components or technologies:** Event Grid, Azure Functions, Azure Storage (Blob), Azure Data Factory, Azure SQL, Monitor.
* **Triggering mechanism:** Event Grid topics or Storage Blob-created events trigger Azure Functions or pipelines.
* **Processing flow:** Event -> Event Grid -> Azure Function (transform) -> Data Lake -> Data Factory scheduled pipelines generate daily aggregates -> store reports.
* **Automation & deployment approach:** Use ARM templates, Bicep, or Terraform for infra; Azure DevOps or GitHub Actions for CI/CD.

1. **AWS Glue + Amazon S3 + Redshift for reporting**

* **Link:** https://aws.amazon.com/glue/
* **Overview:** For large-scale data processing, AWS Glue offers a fully managed ETL and analytics solution in conjunction with Redshift and Amazon S3.  In this effective preparation, Glue jobs employing PySpark catalog and process raw data stored in S3.  Comprehensive data analysis and quick querying are made possible by Glue's integration with Redshift and Athena.  The EventBridge or CloudWatch scheduling is supported for automated data pipelines in this configuration. When combined, they provide a serverless, scalable architecture for reporting, aggregation, and data transformation.
* **Key components/technologies:** AWS Glue (ETL jobs & Data Catalog), S3 (data lake), AWS Glue Jobs (PySpark), Redshift, QuickSight.
* **Triggering mechanism:** Glue jobs can be triggered on schedule or via CloudWatch Events.
* **Processing flow:** Raw events landed in S3 -> Glue crawlers register schema -> Glue ETL jobs transform and store curated data -> Athena/Redshift used to run daily aggregation queries -> reports exported to S3.
* **Automation & deployment approach:** Glue resources via CloudFormation; schedule Glue jobs using EventBridge and manage via CI/CD.

1. **Generating automated reports**

* **Link:** https://aws.amazon.com/blogs/big-data/
* **Overview:** Automated report generation uses processed datasets to schedule and generate summary reports on a daily or recurring basis.  Tools like Python scripts, AWS Lambda, or Glue can be used to create reports in CSV, PDF, or dashboard formats.  Cron jobs or EventBridge events that initiate reporting workflows automatically handle scheduling.  Data is gathered, packaged, and stored in S3, then delivered via email, SNS, Slack, or BI applications. Through automation, this method guarantees quick, accurate, and consistent insights with little manual intervention.
* **Key components or technologies:** EventBridge schedules, reporting engine (Python scripts, AWS Lambda, Glue, Spark), storage (S3), distribution (email via SES, Slack, dashboards).
* **Triggering mechanism:** Daily scheduled trigger (EventBridge scheduled event or cron job) or after end-of-day event.
* **Processing flow:** Aggregation queries run against curated data -> format report (CSV) -> store in S3 and distribute via email or upload to BI tool.
* **Automation & deployment approach:** Build report generator as a deployable function (Lambda or container), deploy via IaC and CI/CD, keep tests for accuracy.

1. **Observability and fault-tolerance patterns for event-driven systems**

* **Link:** https://aws.amazon.com/builders-library/
* **Overview:** In event-driven architectures, fault-tolerance and observability patterns guarantee robustness and dependability.  They include techniques for gracefully handling failures, such as idempotency, exponential backoff, retries, and Dead-Letter Queues (DLQs).  Deep insight into system health can be obtained through monitoring and tracing using CloudWatch, Prometheus, Grafana, or Open Telemetry.  For quicker resolution, engineers are alerted of problems or abnormalities via social networking sites or other methods.  When combined, these patterns allow for resilient, self-repairing event pipelines that reduce data loss and downtime.
* **Key components or technologies:** Dead-Letter Queues (SQS DLQ), CloudWatch, X-Ray, SNS alerts, IAM, retries and backoff.
* **Triggering mechanism:** Error handling is triggered on exception; messages moved to DLQ after max retries.
* **Processing flow:** Message -> consumer -> if processing fails, automatic retry -> after N attempts push to DLQ -> alerting pipeline informs engineers -> replay mechanism to reprocess DLQ.
* **Automation & deployment approach:** Deploy monitoring infra via IaC, add alerting rules, automate runbooks and remediation scripts via CI/CD.