# Apache Flume Lab Summary

Great! Below is **Lab A: Log Aggregation from Multiple Services** written in the same polished and instructional style as your existing PDF. It's designed for beginners but with a real-world context to engage students in a meaningful way.

---

## 🔧 Apache Flume Hands-On Lab – Beginner Level

### ✅ Lab A: Log Aggregation from Multiple Services

---

### 🎯 Objective

In this lab, students will simulate log aggregation from two separate services (e.g., `app1` and `app2`) using Apache Flume. They will configure a single Flume agent with **multiple sources** to collect logs and write them into a **unified output file**, mimicking a real-world log aggregation system used in distributed applications.

---

### 🧰 Lab Setup

- **Pre-installed:** Apache Flume
- **Environment:** Linux (CLI)
- **Files Needed:**
  - Two directories simulating service log output
  - Sample log files (`app1.log`, `app2.log`)
- **Flume Components Used:**
  - `SpoolDir` Sources (2)
  - `Memory` Channel
  - `File Roll Sink`

---

### Step-by-Step Instructions

#### 1. Create Simulated Log Directories

```bash
mkdir -p ~/flume-lab/app1_logs
mkdir -p ~/flume-lab/app2_logs
```

#### 2. Create Sample Log Files

```bash
bash

echo "INFO App1 - Service started" > ~/flume-lab/app1_logs/app1.log
echo "INFO App2 - Service initialized" > ~/flume-lab/app2_logs/app2.log
```

### 3. Create Output Directory

```bash
bash

mkdir -p ~/flume-lab/aggregated_logs
```

### 4. Create Flume Configuration File

Save the following as `multi-source-flume.conf` in your home directory:

```properties
properties

# Define the agent
agent1.sources = source1 source2
agent1.sinks = sink1
agent1.channels = channel1

# Source 1 - App1 logs
agent1.sources.source1.type = spooldir
agent1.sources.source1.spoolDir = /home/YOUR_USERNAME/flume-lab/app1_logs
agent1.sources.source1.fileHeader = true

# Source 2 - App2 logs
agent1.sources.source2.type = spooldir
agent1.sources.source2.spoolDir = /home/YOUR_USERNAME/flume-lab/app2_logs
agent1.sources.source2.fileHeader = true

# Channel
agent1.channels.channel1.type = memory
agent1.channels.channel1.capacity = 1000
agent1.channels.channel1.transactionCapacity = 100

# Sink
agent1.sinks.sink1.type = file_roll
agent1.sinks.sink1.sink.directory = /home/YOUR_USERNAME/flume-lab/aggregated_logs
agent1.sinks.sink1.sink.rollInterval = 0

# Bind sources and sinks to channel
agent1.sources.source1.channels = channel1
agent1.sources.source2.channels = channel1
agent1.sinks.sink1.channel = channel1
```

> 🔁 **Note:** Replace `YOUR_USERNAME` with your actual Linux username.

---

### 5. Run the Flume Agent

```bash
bash

flume-ng agent --conf $FLUME_HOME/conf --conf-file ~/multi-source-flume.conf --name agent1 -Dflume.
```

---

## ✅ Expected Outcome

- A new file will be created in `~/flume-lab/aggregated_logs/` containing the combined content of both `app1.log` and `app2.log`.
- The file will have Flume headers (if enabled), and you'll observe live output in the terminal.

---

## 🧪 Verification

```
bash

cat ~/flume-lab/aggregated_logs/*
```

Expected content:

```
pgsql

INFO App1 - Service started
INFO App2 - Service initialized
```

---

## 🧯 Troubleshooting Tips

- Ensure log file names in `spooldir` directories are unique to avoid reprocessing issues.
- Check agent name matches in the command line and configuration.
- If no data appears, validate directory paths and permissions.

---

## 🧠 Concepts Reinforced

- Log collection from multiple services
- Use of multiple sources in one agent
- Centralized log aggregation for monitoring and debugging

---

Would you like me to proceed with Lab B next (IoT sensor data with `exec` source)?

Perfect! Here's the next polished lab in the same format:

---

# 🔧 Apache Flume Hands-On Lab – Beginner Level

## ✅ Lab B: Collecting and Storing IoT Sensor Data

---

## 🎯 Objective

In this lab, students will simulate real-time IoT sensor data collection using a shell script and Apache Flume. The goal is to demonstrate how Flume can act as a lightweight data pipeline from edge devices to centralized storage—an essential concept in **smart systems** and **communication networks**.

---

## 🧰 Lab Setup

- **Pre-installed:** Apache Flume
- **Environment:** Linux (CLI)
- **Files Needed:**
  - A simple bash script to simulate sensor readings (temperature, signal strength)
- **Flume Components Used:**
  - `Exec` Source
  - `Memory` Channel
  - `File Roll Sink`

---

## 🧪 Step-by-Step Instructions

### 1. Create a Directory for the Lab

```
bash

mkdir -p ~/flume-lab/iot_logs
```

### 2. Create the Sensor Simulation Script

Create a file named `sensor_simulator.sh` in `~/flume-lab/` with the following content:

```
bash

#!/bin/bash
while true
do
  echo "TEMP: $((20 + RANDOM % 10)) C | SIGNAL: $((60 + RANDOM % 20)) dBm | $(date)"
  sleep 2
done
```

### 3. Make the Script Executable

```
bash

chmod +x ~/flume-lab/sensor_simulator.sh
```

### 4. Create Flume Configuration File

Save the following as `**iot-sensor-flume.conf**` in your home directory:

```
properties

# Define the agent
agent1.sources = sensorSource
agent1.sinks = sink1
agent1.channels = memoryChannel

# Source - Exec (simulate sensor)
agent1.sources.sensorSource.type = exec
agent1.sources.sensorSource.command = /home/YOUR_USERNAME/flume-lab/sensor_simulator.sh
agent1.sources.sensorSource.shell = /bin/bash

# Channel
agent1.channels.memoryChannel.type = memory
agent1.channels.memoryChannel.capacity = 1000
agent1.channels.memoryChannel.transactionCapacity = 100

# Sink - File output
agent1.sinks.sink1.type = file_roll
agent1.sinks.sink1.sink.directory = /home/YOUR_USERNAME/flume-lab/iot_logs
agent1.sinks.sink1.sink.rollInterval = 0

# Bind source and sink to channel
agent1.sources.sensorSource.channels = memoryChannel
agent1.sinks.sink1.channel = memoryChannel
```

> 🔁 **Note:** Replace `**YOUR_USERNAME**` with your actual Linux username.

---

### 5. Run the Flume Agent

```
bash

flume-ng agent --conf $FLUME_HOME/conf --conf-file ~/iot-sensor-flume.conf --name agent1 -Dflume.ro
```

---

## ✅ Expected Outcome

- Flume continuously captures simulated sensor output from the shell script.
- Data is written to files in `**~/flume-lab/iot_logs/**`.

---

## 🪛 Verification

```
bash

cat ~/flume-lab/iot_logs/*
```

Example output:

```
yaml

TEMP: 25 C | SIGNAL: 75 dBm | Sat May  3 11:04:21 UTC 2025
TEMP: 28 C | SIGNAL: 65 dBm | Sat May  3 11:04:23 UTC 2025
```

---

## 🧯 Troubleshooting Tips

- If no output is captured, check that the script runs manually and generates output.
- Ensure the script path is correct and executable.
- Confirm the Flume agent has permission to access the script and write to the output directory.

---

## 🧠 Concepts Reinforced

- Real-time streaming from a sensor-like source
- Shell integration with Flume (`**exec**` source)
- Use of Flume in IoT and communication systems

---

Would you like me to continue with Lab C: **Routing Logs Based on Severity (Interceptors)**?

Excellent! Here's **Lab C: Routing Logs Based on Severity (Interceptors)** in the same structured and polished format:

---

# 🔧 Apache Flume Hands-On Lab – Beginner Level

## ✅ Lab C: Routing Logs Based on Severity (Interceptors)

---

## 🎯 Objective

This lab teaches students how to use **Flume Interceptors** to route log events to different destinations based on their **severity level** (e.g., INFO, WARN, ERROR). This reflects real-world log processing scenarios used in **monitoring systems, log filtering**, and **security auditing**.

---

## 🧰 Lab Setup

- **Pre-installed:** Apache Flume
- **Environment:** Linux (CLI)
- **Files Needed:**
  - A log file containing mixed severity levels (INFO, WARN, ERROR)
- **Flume Components Used:**
  - `SpoolDir` Source
  - `Memory` Channel (2)
  - `File Roll` Sink (2)
  - `Regex Filtering Interceptor`

---

## 🛠 Step-by-Step Instructions

### 1. Create Directories for Input and Output

```
bash

mkdir -p ~/flume-lab/severity_logs/input
mkdir -p ~/flume-lab/severity_logs/output_info
mkdir -p ~/flume-lab/severity_logs/output_error
```

### 2. Create a Sample Log File

```
bash

cat <<EOF > ~/flume-lab/severity_logs/input/system.log
INFO  - System startup completed.
WARN  - CPU usage nearing threshold.
ERROR - Failed to load driver.
INFO  - Scheduled task executed.
```

```
ERROR - Disk read failure.
EOF
```

### 3. Create Flume Configuration File

Save the following as `severity-routing-flume.conf` in your home directory:

```
properties

# Agent definition
agent1.sources = logSource
agent1.sinks = infoSink errorSink
agent1.channels = infoChannel errorChannel

# Source - SpoolDir
agent1.sources.logSource.type = spooldir
agent1.sources.logSource.spoolDir = /home/YOUR_USERNAME/flume-lab/severity_logs/input
agent1.sources.logSource.includePattern = .*\.log
agent1.sources.logSource.fileHeader = true

# Interceptor - Regex Filtering
agent1.sources.logSource.interceptors = i1 i2

# Interceptor for INFO logs
agent1.sources.logSource.interceptors.i1.type = regex_filter
agent1.sources.logSource.interceptors.i1.regex = ^INFO
agent1.sources.logSource.interceptors.i1.excludeEvents = false

# Interceptor for ERROR logs
agent1.sources.logSource.interceptors.i2.type = regex_filter
agent1.sources.logSource.interceptors.i2.regex = ^ERROR
agent1.sources.logSource.interceptors.i2.excludeEvents = false

# Channels
agent1.channels.infoChannel.type = memory
agent1.channels.errorChannel.type = memory

# Sinks
agent1.sinks.infoSink.type = file_roll
agent1.sinks.infoSink.sink.directory = /home/YOUR_USERNAME/flume-lab/severity_logs/output_info

agent1.sinks.errorSink.type = file_roll
agent1.sinks.errorSink.sink.directory = /home/YOUR_USERNAME/flume-lab/severity_logs/output_error

# Bindings
agent1.sources.logSource.channels = infoChannel errorChannel
agent1.sinks.infoSink.channel = infoChannel
agent1.sinks.errorSink.channel = errorChannel
```

🔁 **Note:** Replace `YOUR_USERNAME` with your actual Linux username.

---

### 4. Run the Flume Agent

```
bash

flume-ng agent --conf $FLUME_HOME/conf --conf-file ~/severity-routing-flume.conf --name agent1 -Dflu
```

---

## ✅ Expected Outcome

- Flume reads `system.log` and separates log entries into two destinations:

- `output_info/` will store INFO logs.
- `output_error/` will store ERROR logs.

---

## 🪛 Verification

```
bash

echo "--- INFO LOGS ---"
cat ~/flume-lab/severity_logs/output_info/*
echo "--- ERROR LOGS ---"
cat ~/flume-lab/severity_logs/output_error/*
```

Example Output:

```
pgsql

--- INFO LOGS ---
INFO  - System startup completed.
INFO  - Scheduled task executed.

--- ERROR LOGS ---
ERROR - Failed to load driver.
ERROR - Disk read failure.
```

---

## 🧯 Troubleshooting Tips

- Ensure your regex matches the log line beginning (e.g., `^INFO`).
- Make sure the input log filename matches the `includePattern`.
- Logs with unmatched severity will be ignored unless an interceptor is defined for them.

---

## 🧠 Concepts Reinforced

- Conditional routing of events
- Use of Flume interceptors
- Real-world log filtering for observability and incident response

---

Would you like to proceed with **Lab D: Monitoring Real-Time Network Event Logs** next?

Excellent! Here's **Lab D: Monitoring Real-Time Network Event Logs**, crafted in the same polished format:

---

# 🔧 Apache Flume Hands-On Lab – Beginner Level

## ✅ Lab D: Monitoring Real-Time Network Event Logs

---

## 🎯 Objective

In this lab, students will simulate **network event monitoring** by continuously reading and streaming system-level logs (such as connection attempts and firewall alerts) using Apache Flume. This lab highlights Flume's utility in **real-time log collection** and **security monitoring**, areas relevant to **communication networks** and **cybersecurity**.

---

## 🧰 Lab Setup

- **Pre-installed:** Apache Flume
- **Environment:** Linux (CLI)
- **Files Needed:**
  - A simulated network log stream (bash script)
- **Flume Components Used:**
  - `Exec` Source
  - `Memory` Channel
  - `File Roll` Sink

---

## 📝 Step-by-Step Instructions

### 1. Create the Required Directory Structure

```
bash

mkdir -p ~/flume-lab/network_monitoring/logs
```

### 2. Create the Network Log Simulator Script

Create a file named `network_simulator.sh` in `~/flume-lab/`:

```
bash

#!/bin/bash
while true
do
  echo "CONNECTION from 192.168.$((RANDOM % 255)).$((RANDOM % 255)) | PORT: $((1024 + RANDOM % 4817(
  sleep 3
done
```

### 3. Make the Script Executable

```
bash

chmod +x ~/flume-lab/network_simulator.sh
```

### 4. Create Flume Configuration File

Save the following as `network-monitoring-flume.conf` in your home directory:

```
properties

# Define agent
agent1.sources = netSource
agent1.sinks = sink1
agent1.channels = memoryChannel

# Source - Exec (simulate network log stream)
agent1.sources.netSource.type = exec
agent1.sources.netSource.command = /home/YOUR_USERNAME/flume-lab/network_simulator.sh
agent1.sources.netSource.shell = /bin/bash

# Channel
agent1.channels.memoryChannel.type = memory
agent1.channels.memoryChannel.capacity = 1000
agent1.channels.memoryChannel.transactionCapacity = 100

# Sink
agent1.sinks.sink1.type = file_roll
agent1.sinks.sink1.sink.directory = /home/YOUR_USERNAME/flume-lab/network_monitoring/logs
agent1.sinks.sink1.sink.rollInterval = 0

# Bind source and sink to channel
agent1.sources.netSource.channels = memoryChannel
agent1.sinks.sink1.channel = memoryChannel
```

> 🔁 **Note:** Replace `YOUR_USERNAME` with your actual Linux username.

---

### 5. Run the Flume Agent

```
bash

flume-ng agent --conf $FLUME_HOME/conf --conf-file ~/network-monitoring-flume.conf --name agent1 -D
```

---

## ✅ Expected Outcome

- Flume continuously reads the simulated network event stream.
- Logs are written in near real-time to `~/flume-lab/network_monitoring/logs/`.

---

## 🔪 Verification

```
bash

tail -f ~/flume-lab/network_monitoring/logs/*
```

Sample output:

```
yaml

CONNECTION from 192.168.12.87 | PORT: 33034 | Sat May  3 12:07:43 UTC 2025
CONNECTION from 192.168.202.39 | PORT: 52011 | Sat May  3 12:07:46 UTC 2025
```

---

## 🧯 Troubleshooting Tips

- Make sure your simulator script outputs to stdout and runs continuously.
- Check directory paths and permissions if logs do not appear.
- Ensure the correct shell is specified (`**`/bin/bash`**`).

---

## 🧠 Concepts Reinforced

- Continuous data streaming
- Real-time log processing
- Applications in network monitoring and cybersecurity
- Use of `**exec**` source in Flume for live data pipelines

---

Would you like to proceed with **Lab E: Sending Events to HDFS for Archival** next?

Great! Here's **Lab E: Sending Events to HDFS for Archival**, designed in the same professional format:

---

# 🔧 Apache Flume Hands-On Lab – Beginner Level

## ✅ Lab E: Sending Events to HDFS for Archival

---

### 🎯 Objective

In this lab, students will learn how Apache Flume integrates with **Hadoop Distributed File System (HDFS)** to enable long-term storage and archival of logs. This lab demonstrates how real-time data can be preserved for **big data analytics**, which is crucial in both **communication systems** and **computer science research**.

---

### 🧰 Lab Setup

- **Pre-installed:** Apache Flume, Hadoop with HDFS
- **Environment:** Linux (CLI)
- **Files Needed:**
  - Pre-existing log file with generic events
- **Flume Components Used:**
  - `SpoolDir` Source
  - `Memory` Channel
  - `HDFS` Sink

---

### 📝 Step-by-Step Instructions

#### 1. Create Input Directory and Sample Log File

```
bash

mkdir -p ~/flume-lab/hdfs_archival/input

cat <<EOF > ~/flume-lab/hdfs_archival/input/events.log
User login: user1
User logout: user1
File uploaded: report.pdf
Download started: video.mp4
EOF
```

#### 2. Create Output Directory in HDFS

```
bash

hdfs dfs -mkdir -p /user/YOUR_USERNAME/flume_archival
```

🔁 Replace `YOUR_USERNAME` with your actual HDFS username.

### 3. Create Flume Configuration File

Save the following as `flume-hdfs.conf` in your home directory:

```
properties

# Define agent
agent1.sources = spoolSource
agent1.channels = memoryChannel
agent1.sinks = hdfsSink

# Source - SpoolDir
agent1.sources.spoolSource.type = spooldir
agent1.sources.spoolSource.spoolDir = /home/YOUR_USERNAME/flume-lab/hdfs_archival/input
agent1.sources.spoolSource.fileHeader = true
agent1.sources.spoolSource.includePattern = .*\.log

# Channel
agent1.channels.memoryChannel.type = memory
agent1.channels.memoryChannel.capacity = 1000
agent1.channels.memoryChannel.transactionCapacity = 100

# Sink - HDFS
agent1.sinks.hdfsSink.type = hdfs
agent1.sinks.hdfsSink.hdfs.path = hdfs://localhost:9000/user/YOUR_USERNAME/flume_archival/
agent1.sinks.hdfsSink.hdfs.fileType = DataStream
agent1.sinks.hdfsSink.hdfs.writeFormat = Text
agent1.sinks.hdfsSink.hdfs.rollInterval = 0

# Bindings
agent1.sources.spoolSource.channels = memoryChannel
agent1.sinks.hdfsSink.channel = memoryChannel
```

> 🔁 Replace `YOUR_USERNAME` in both the local and HDFS paths.

---

### 4. Run the Flume Agent

```
bash

flume-ng agent --conf $FLUME_HOME/conf --conf-file ~/flume-hdfs.conf --name agent1 -Dflume.root.logg
```

---

## ✅ Expected Outcome

- Flume reads `events.log` from the local input folder.
- The contents are written and stored in HDFS under `/user/YOUR_USERNAME/flume_archival/`.

---

## 🪛 Verification

```
bash

hdfs dfs -cat /user/YOUR_USERNAME/flume_archival/*
```

Expected output:

```yaml
User login: user1
User logout: user1
File uploaded: report.pdf
Download started: video.mp4
```

---

## 🧯 Troubleshooting Tips

- Make sure Hadoop is running and HDFS is accessible.
- If no data appears in HDFS, verify that `events.log` has not already been consumed (SpoolDir does not reprocess files).
- Ensure file paths are correct and accessible.

---

## 🧠 Concepts Reinforced

- Flume-to-HDFS integration
- Reliable archival of streaming data
- HDFS file structure and usage
- Practical big data ingestion workflows

---

Would you like to continue with **Lab F: Custom Flume Agent to Process XML or JSON Logs** next?