LOG AGGREGATION

Log aggregation is the process of **collecting, centralizing, and organizing log data** from many different sources (servers, apps, devices) into **one single location for easier analysis, monitoring, and troubleshooting.**

Telemetry is the automated process of collecting, transmitting, and analyzing data from remote or inaccessible sources to monitor and control systems, providing real-time insights into performance, status, and behavior without physical presence.

There are two major approaches for achieving this purpose.

Log Forwarding (Push Model) – SaaS model
This is the most common and robust method. The application or system where the source runs is configured to **push** its logs to a separate, centralized logging server or service.

Retrieval (Pull Model) – self hosted
In this approach, the logs are stored locally (or in a vendor-specific storage), and a separate client or script **pulls** them when needed, typically via a dedicated logging API/agent.

In here we are going to discuss about how to manage and record server logs (container logs) from a centralized location. We proceed with push model with minimal infrastructure configuration. In the other method is able to store logs into local system, setup agent for push logs to the centralized server or else logging agent install itself within the local/remote server (as standalone applications or docker images)

Our plan is

> Structured application logs with Winston (for NodeJS API)
> Setup Grafana cloud account (cloud is able to act as a Saas for log management)
> Setup Grafana Loki as a data source
> Install Docker plugin that will read logs from Docker containers and ship them to Loki.
> Configure Docker Deamon. Json for sending logs
> Inspect logs in Grafana dashboard

Node.js App
↓ (JSON logs)
Docker stdout
↓
Promtail (log agent) (in our case log agent is not necessary and configured with docker plugin)
↓
Loki (log database)
↓
Grafana (UI for team)

Structured application logs with Winston (for NodeJS API)

One of the best practices is avoiding using plain console logs and moving to structured and leveled comprehensive and detailed application logs. Winston is one of the better libraries for managing structured logs and many more. Additionally, Winston Transporters can be used to ship logs to remote destinations as well.

npm install winston

Example logger object with winston (src/logger.index.js)

```
const logger = require('./logger');

logger.info('Server started', { port: 5005 });
logger.error('DB connection failed', { error });
```

```
const winston = require("winston");
const { Logtail } = require("@logtail/node");
// const { LogtailTransport } = require("winston-transport-logtail");
const LokiTransport = require('winston-loki');
const environment = require("../config/environmentConfig");
const { environmentTypes } = require("../constants/commonConstants");

const customFormat = winston.format.printf(
  ({ level, message, timestamp, stack, ...meta }) => {
    const metaString = Object.keys(meta).length
      ? JSON.stringify(meta, null, 2)
      : "";

    return `${timestamp} [${level}]: ${stack || message} ${metaString}`;
  }
);

const dev_logger = () => {
  return winston.createLogger({
    level: "info",
    format: winston.format.combine(
      winston.format.colorize(),
      winston.format.timestamp(),
      customFormat
      // winston.format.errors({ stack: true }),
      // winston.format.json()
    ),
    transports: [
      new winston.transports.Console(), // for Docker stdout
    ],
  });
};

const prod_logger = () => {
  return winston.createLogger({
    level: "info",
    format: winston.format.combine(
      // winston.format.colorize(),
      winston.format.timestamp(),
      customFormat
      // winston.format.errors({ stack: true }),
      // winston.format.json()
    ),
    transports: [
      new winston.transports.Console(), // for Docker stdout
    ],
  });
};

// NOTE: injectingrespective logger based on the current environment
let logger = dev_logger();

if(environment.ENVIRONMENT === environmentTypes.PROD){
  logger = prod_logger();

}

module.exports = logger;
```

```
// NOTE: >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> COMMENT >>>>>>>>>>>>>>>>>>>>>

// const winston = require("winston");
// const { Logtail } = require("@logtail/node");
// // const { LogtailTransport } = require("winston-transport-logtail");
// const LokiTransport = require('winston-loki');
// const environment = require("../config/environmentConfig");
// const { environmentTypes } = require("../constants/commonConstants");

// const customFormat = winston.format.printf(
//   ({ level, message, timestamp, stack, ...meta }) => {
//     const metaString = Object.keys(meta).length
//       ? JSON.stringify(meta, null, 2)
//       : "";

//     return `${timestamp} [${level}]: ${stack || message} ${metaString}`;
//   }
// );

// const dev_logger = () => {
//   return winston.createLogger({
//     level: "info",
//     format: winston.format.combine(
//       winston.format.colorize(),
//       winston.format.timestamp(),
//       customFormat
//       // winston.format.errors({ stack: true }),
//       // winston.format.json()
//     ),
//     transports: [
//       new winston.transports.Console(), // for Docker stdout
//       // new LogtailTransport(logtail), // send to cloud
//       new LokiTransport({
//         host: environment.GRAFANA_LOKI_ENDPOINT,
//         basicAuth: `${environment.GRAFANA_LOKI_USER}:${environment.GRAFANA_LOKI_API_KEY}`,
//         onConnectionError: (err) => {
//           console.error("Loki connection error:", err);
//         },
//         labels: { app: "my-node-api" },
//         json: true
//       }),
//     ],
//   });
// };

// const prod_logger = () => {
//   return winston.createLogger({
//     level: "info",
//     format: winston.format.combine(
//       // winston.format.colorize(),
//       winston.format.timestamp(),
//       customFormat
//       // winston.format.errors({ stack: true }),
//       // winston.format.json()
//     ),
//     transports: [
//       new winston.transports.Console(), // for Docker stdout
//       // new LogtailTransport(logtail), // send to cloud
//       new LokiTransport({
//         host: environment.GRAFANA_LOKI_ENDPOINT,
//         basicAuth: `${environment.GRAFANA_LOKI_USER}:${environment.GRAFANA_LOKI_API_KEY}`,
//         onConnectionError: (err) => {
//           console.error("Loki connection error:", err);
//         },
//         labels: { app: "my-node-api" },
//         json: true
//       }),
//     ],
//   });
// };

// // NOTE: injectingrespective logger based on the current environment
// let logger = dev_logger();

// if(environment.ENVIRONMENT === environmentTypes.PROD){
//   logger = prod_logger();

// }

// // console.log(">>>>>>>>>>>>>>>>>>>>>> LOGGER");
// // // console.log(Object.keys(logger));
// // console.log(logger._readableState.pipes[1]);


// module.exports = logger;

// NOTE: >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> COMMENT >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

// const winston = require("winston");
// const LokiTransport = require("winston-loki");
```

```
// NOTE: >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> COMMENT >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

// const winston = require("winston");
// const LokiTransport = require("winston-loki");
// const environment = require("../config/environmentConfig");

// // Define log levels
// const levels = {
//   error: 0,
//   warn: 1,
//   info: 2,
//   http: 3,
//   debug: 4,
// };

// // Define colors for console output
// const colors = {
//   error: "red",
//   warn: "yellow",
//   info: "green",
//   http: "magenta",
//   debug: "white",
// };

// winston.addColors(colors);

// // Determine log level based on environment
// const level = () => {
//   const env = environment.ENVIRONMENT || "DEV";
//   const isDevelopment = env === "DEV";
//   return isDevelopment ? "debug" : "info";
// };

// // Custom format for console (development)
// const consoleFormat = winston.format.combine(
//   winston.format.timestamp({ format: "YYYY-MM-DD HH:mm:ss:ms" }),
//   winston.format.colorize({ all: true }),
//   winston.format.printf(
//     (info) => `${info.timestamp} ${info.level}: ${info.message}`
//   )
// );

// // Custom format for Loki (production)
// const lokiFormat = winston.format.combine(
//   winston.format.timestamp(),
//   winston.format.errors({ stack: true }),
//   winston.format.json()
// );

// // Create transports array
// const transports = [];

// // Console transport (always enabled for local debugging)
// transports.push(
//   new winston.transports.Console({
//     format: consoleFormat,
//   })
// );

// // Loki transport (production only)
// if (environment.ENVIRONMENT=== "PROD" && environment.GRAFANA_LOKI_ENDPOINT) {
//   transports.push(
//     new LokiTransport({
//       host: environment.GRAFANA_LOKI_ENDPOINT,
//       labels: {
//         app: "nodejs-api",
//         environment: environment.ENVIRONMENT,
//         version: "1.0.0",
//         host: environment.GRAFANA_LOKI_ENDPOINT,
//       },
//       json: true,
//       format: lokiFormat,
//       replaceTimestamp: true,
//       onConnectionError: (err) =>
//         console.error("Loki connection error:", err),
//       basicAuth: `${environment.GRAFANA_LOKI_USER}:${environment.GRAFANA_LOKI_API_KEY}`,
//       // Batching configuration for better performance
//       batching: true,
//       interval: 5, // Send logs every 5 seconds
//     })
//   );
// }

// // Create logger instance
// const logger = winston.createLogger({
//   level: level(),
//   levels,
//   transports,
//   // Handle exceptions and rejections
//   exceptionHandlers: transports,
//   rejectionHandlers: transports,
//   exitOnError: false,
// });

// module.exports = logger;
```

Setup Grafana cloud account (cloud is able to act as a Saas for log management)

https://grafana.com/cloud
create free account in Grafana cloud

if you have already an account go with -> My Account
This will display your current cloud stacks

If you are planning to create self-hosted your own data collector you should create data source manually
connections -> data sources -> add new data source (then configure and test your data source)

But Grafana cloud has already created default loki data source for our use ready for use.
We can verify in here. connections -> data sources

Now lets see how to obtain loki url and create api tokens

Setup Grafana Loki as a data source

https://grafana.com/cloud -> My account -> See the cloud stacks available
Select your cloud stack and -> details -> find the Loki data source available -> details
This will include the necessary details about Loki data source

go to section
Sending Logs to Grafana Cloud using Promtail -> From a standalone host

copy the loki url

```
clients:
  - url: https://1437567:<Your Grafana.com API Token>@logs-prod-028.grafana.net/loki/api/v1/push
```

Now we should generate api token with correct permissions
https://grafana.com/cloud -> My account
Navigate into “Access policies” under “security” in left side bar

Create access policy
Define scopes with permissions
logs: write
Create

Now created access policy will show on the list
Find the option “token -> add token” in created access policy
Give it a name and create token, copy one time token value and secure

inject the token into this url
url: https://1437567:<Your Grafana.com API Token>@logs-prod-028.grafana.net/loki/api/v1/push

Install Docker plugin that will read logs from Docker containers and ship them to Loki.

https://grafana.com/docs/loki/latest/send-data/docker-driver

The Docker plugin must be installed on each Docker host that will be running containers you want to collect logs from.

Run the following command to install the plugin, updating the release version, or changing the architecture (arm64 and amd64 are currently supported), if needed

docker plugin install grafana/loki-docker-driver:3.3.2-arm64 --alias loki --grant-all-permissions
change arm -> amd if needed

docker plugin ls

sudo vim /etc/docker/daemon.json

place this inside the file. save and exit

Configure Docker Deamon. Json for sending logs

sudo vim /etc/docker/daemon.json

place this inside the file. save and exit

```
{
        "log-driver": "loki",
        "log-opts": {
            "loki-url": "https://1437567:<token>logs-prod-028.grafana.net/loki/api/v1/push"
        }
}
```

sudo systemctl restart docker

Inspect logs in Grafana dashboard

Go to Grafana cloud dashboard
Left side bar -> explore
Left side bar -> drilldown -> logs

query logs
filter logs
select containers and filter specific log
filter based on time range
and many more

This is only a most fundamental approach for the related topic. Many more things to explore.