Getting the most out of your logs at 3AM

Shyamkumar Kalariya — Software Engineer at **Zenduty**

Whoami ?

Shyamkumar Kalariya

Software Engineer at Zenduty

You can connect with me on LinkedIn here:

What's this talk about?

- The What and Why of logging?(ELI5)
- Common Logging Scenarios
- Logging Best Practices and common mistakes
- Using Logs at 3AM

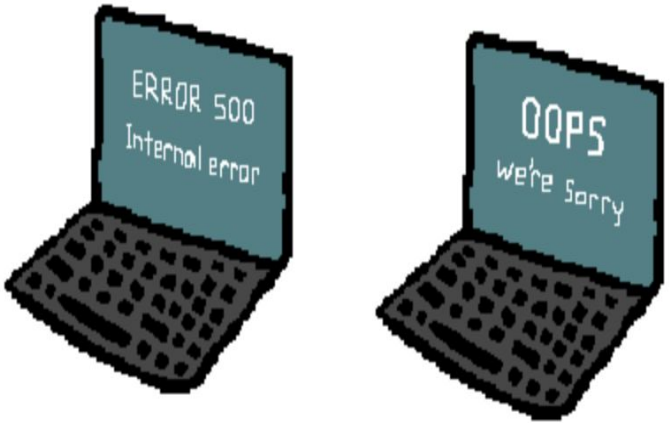
What is logging?

- Logging is the act of capturing and storing messages generated by an application to provide insights into its operation.

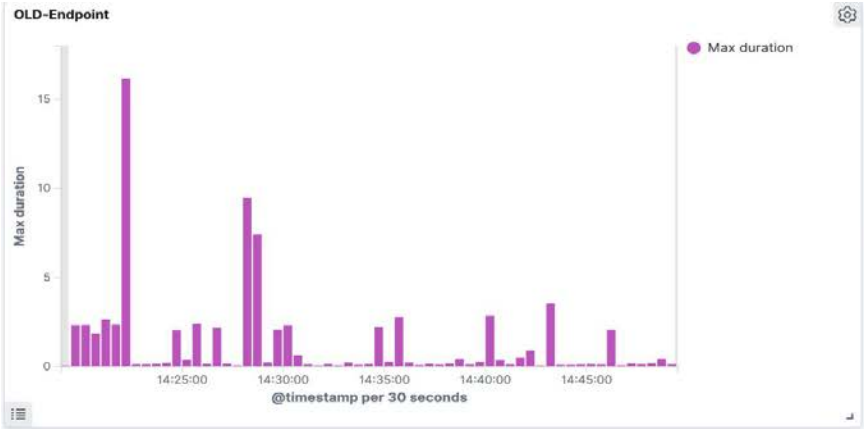
Why is Logging?

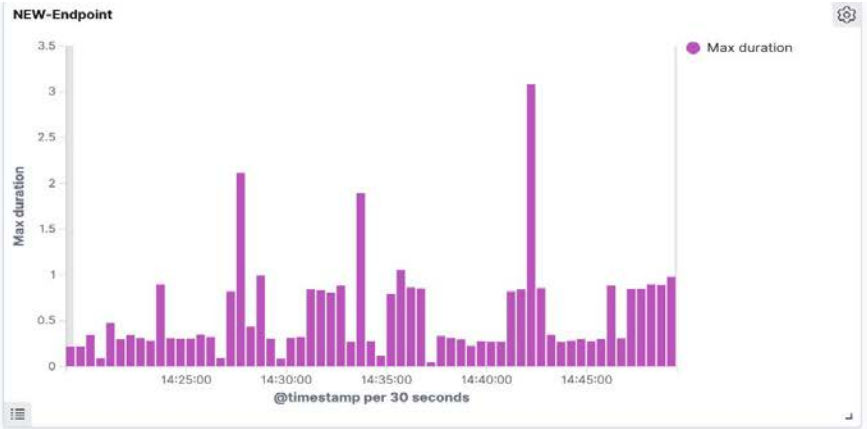
- Troubleshooting issues in production
- Monitoring application health and performance
- Security auditing and compliance
- Understanding user behavior and application flow

Troubleshooting Prod issue



Monitoring application health and performance



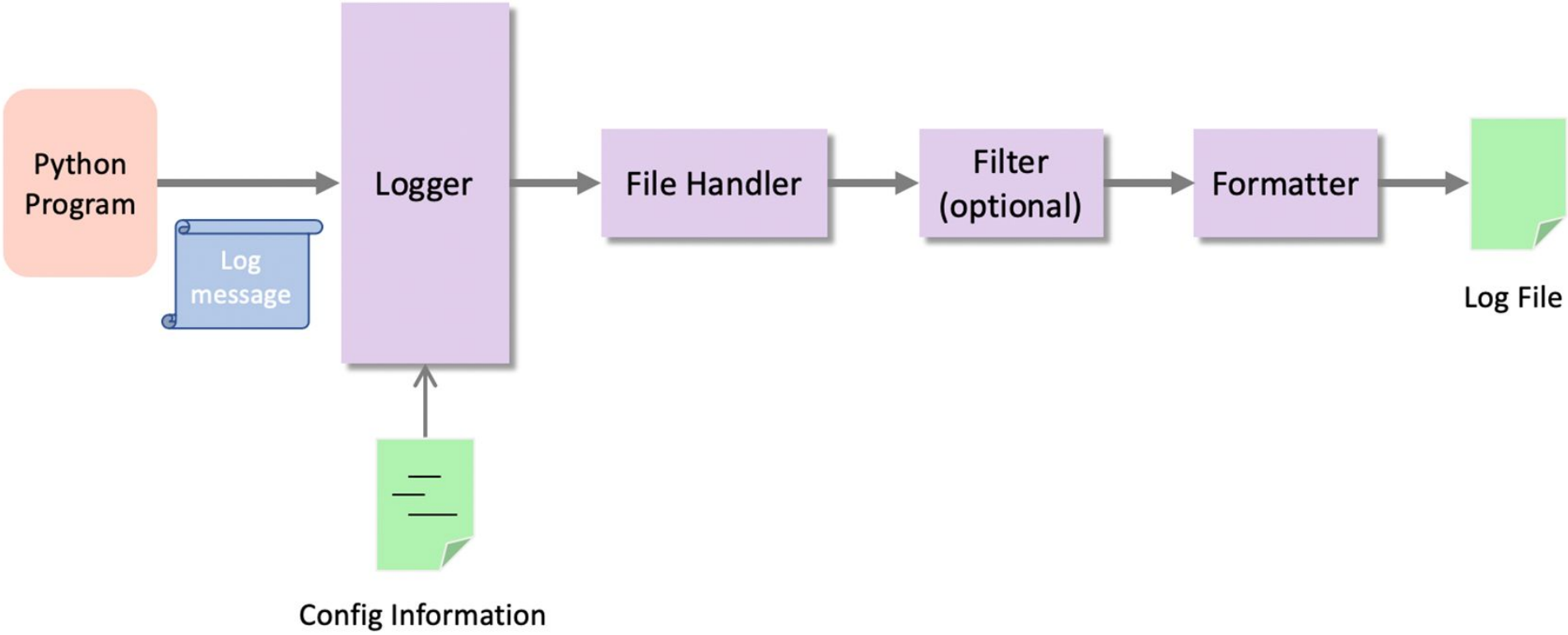


Security Auditing

Logging Framework

Logging framework consists of 4 components

- Loggers
- Handlers
- Filters
- Formatters


Python Program
Log message
Logger
File Handler
Filter (optional)
Formatter
Log File
Config Information

Common Logging Scenario

Debugging Issues

```
logger.debug('This is a debug message')
```

Error Handling

```
try:
    risky_operation()
except Exception as e:
    logger.error('An error occurred', exc_info=True)
```

Common Logging Scenario (Contd)

Performance Monitoring

```
start_time = time.time()
operation()
end_time = time.time()
logger.info(f'Operation took {end_time - start_time} seconds')
```

Security Auditing

```
logger.info(f'User {user_id} logged in from {ip_address}')
```

Common Logging Mistakes

Ignoring context with log message

- While writing log message developer tends to think that it obvious and understandable message, anyone can understand what is the error here.

```
try:
    resposne = requests.post(url,data=data, headers=headers, timeout=5)
except Exception:
    logger.error("Request failed")
```

Not setting proper log level

- In a hurry or without the proper context of a code, developer sets the same level for all logs or setting incorrect log level.

```
try:
    operation1()
    logger.error("Operation 1 completed successfully")
except Exception as e:
    logger.error("Operation 1 failed", extra = {"status_obj": status_obj, "err": str(e)})

try:
    operation2()
    logger.info("Operation 2 completed successfully")
except Exception as e:
    logger.info("Operation 2 failed", extra = {"status_obj": status_obj, "err": str(e)})
```

Unstructured data

```
try:
    operation1()
    logger.info("Operation 1 completed successfully")
except Exception as e:
    status_obj = {
        "function": "operation1",
        "user_id": "1",
        "ip": "192.168.1.100"
    }
    logger.error("Operation 1 failed", extra = {"status_obj": status_obj, "err": str(e)})
try:
    operation2()
    logger.info("Operation 2 completed successfully")
except Exception as e:
    status_obj = {
        "file name": "main.py",
        "line_no": "15",
        "message": "Operation 2 failed"
    }
    logger.error(status_obj)
```

Value type mismatch for a same key

```python
try:
    operation1()
    logger.info("Operation 1 completed successfully")
except Exception as e:
    status_obj = {
        "function": "operation1",
        "user_id": "1",
        "ip": "192.168.1.100",
        "file": "main.py",
        "line": 15,
        "data": {
            "key": "value"
        }
    }
    logger.error("Operation 1 failed", extra = {"status_obj": status_obj, "err": str(e)})
try:
    operation2()
    logger.info("Operation 2 completed successfully")
except Exception as e:
    status_obj = {
        "function": "operation1",
        "user_id": 1,
        "ip": "192.168.1.100",
        "file": "main.py",
        "line": "15",
        "data": "data"
    }
    logger.error("Operation 2 failed", extra = {"status_obj": status_obj, "err": str(e)})
```

Logging Checklist

What to log?

1. Include all necessary metadata to to help pinpoint events and root causes
2. Not to include much extraneous information. This adversely affects scaling and cost
3. Define proper log level
4. Identify PII data that shouldn’t be logged
5. Add correlating key in the logs to correlate logs from different sources to get best out of the logs

Keep the logs structured

Decide the structure of your logs and stick to them

Benefits of Structured Logging

1. Better Visualization
2. Precise alerting
3. Better incident response time
4. Improve various teams collaboration
5. Reduces alert fatigue
6. Improve Alert Configuration

Logging Scenarios

1. While writing logs define the error boundary and log level like what exception you expecting in that specific code block
2. Keep in mind the common logging scenarios

```
try:
    response = requests.post(url, data=data, headers=headers, timeout=5)
except TimeoutError:
    logger.error(msg="send msg request timedout", extra = {"status_obj": status_obj})
except ConnectionError:
    logger.error(msg="cannot reach slack server", extra = {"status_obj": status_obj})
```

Log storage

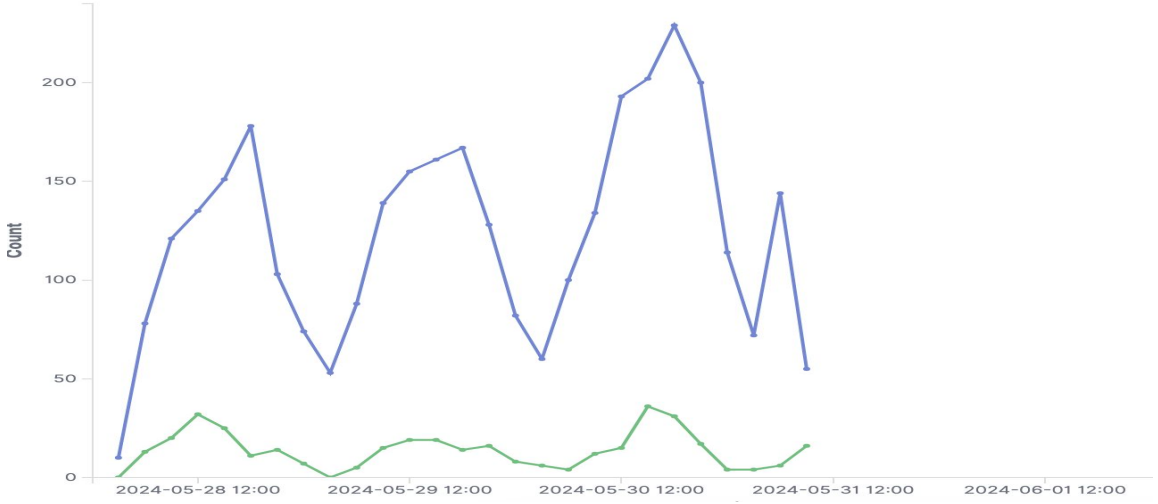
1. Centralize logs from all sources to ensure easy access and correlation.
2. Define retention policies to manage log storage costs and compliance
3. Use indexing to improve log searchability and retrieval speeds.
4. Implement regular backups to prevent data loss.

Alerting

1. Set alerts based on event frequency.
2. Alert on abnormal request durations.
3. Monitor specific status code occurrences.
4. Configure alerts to match specific needs.
5. Look for logs patterns and setup the rule accordingly for anomaly detection

Using Logs at 3AM

- You have the better visualization of your logs. Make most out of the charts
- Once you have better visualization, most of the scenario you will be able get the idea of an anomaly



Using Logs at 3AM

If the issue is not the within the expected error boundary or need to debug specific request then utilize Trace IDs and Stack Traces in Logs

Extract Request IDs:

- Retrieve request IDs from the stack trace or alert details.

Search Logs:

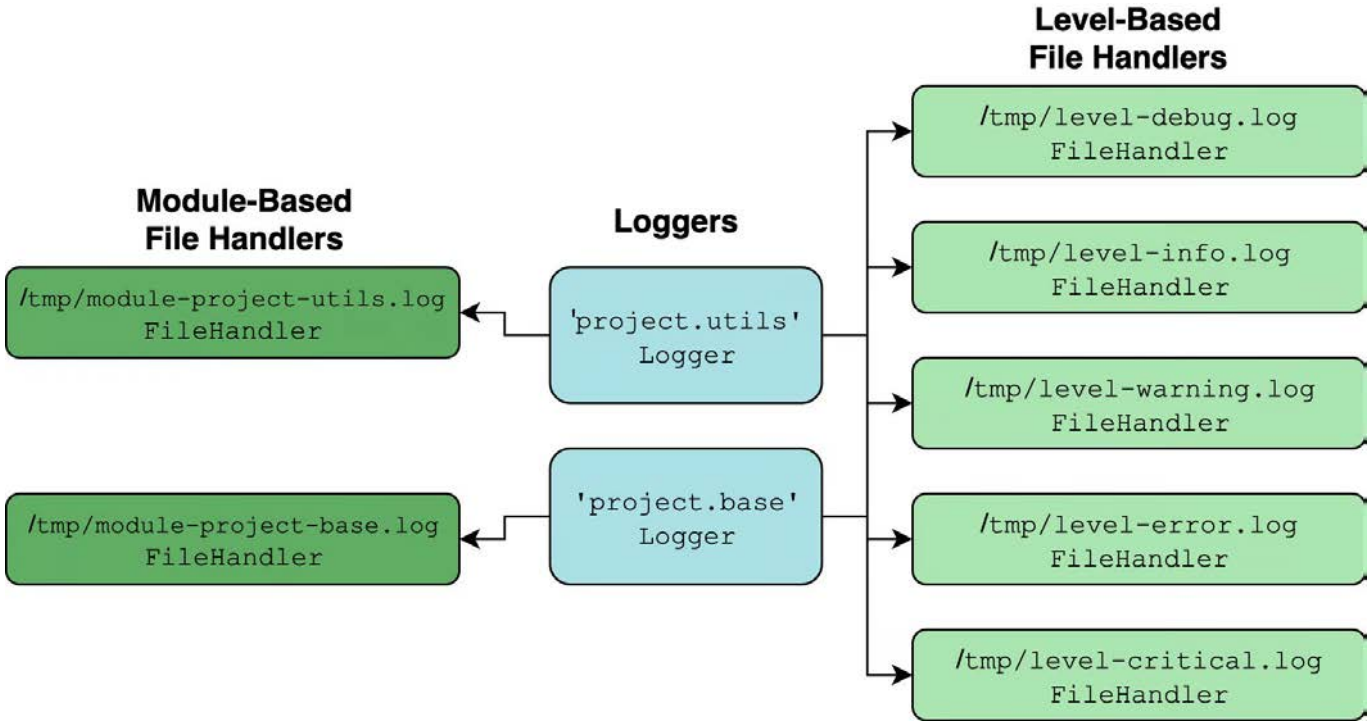
- Use request IDs to locate relevant logs.

Extract events from request cycle

>	Jun 6, 2024 @ 11:37:56.157	Verification code successfully created	send_verification_code	verify_contact_method.py	455	efb253e1b8e5481cb2b817864045a8b7
>	Jun 6, 2024 @ 11:37:56.157	SMS verification required	set_is_call_verification_required	verify_contact_method.py	244	efb253e1b8e5481cb2b817864045a8b7
>	Jun 6, 2024 @ 11:37:56.153	Either country doesn't require voice OTP or Country requires voice otp and country account is already enabled	validate_voice_otp_country_account_enable	verify_contact_method.py	230	efb253e1b8e5481cb2b817864045a8b7
>	Jun 6, 2024 @ 11:37:56.152	Country is not enabled for this account	set_country_account_enable	verify_contact_method.py	195	efb253e1b8e5481cb2b817864045a8b7
>	Jun 6, 2024 @ 11:37:56.152	No need to verify price	set_should_verify_price	verify_contact_method.py	206	efb253e1b8e5481cb2b817864045a8b7
>	Jun 6, 2024 @ 11:37:56.149	Setting iso country code successfully	set_iso_country_code	verify_contact_method.py	177	efb253e1b8e5481cb2b817864045a8b7
>	Jun 6, 2024 @ 11:37:56.148	Setting network provider price successfully	set_network_provider	verify_contact_method.py	162	efb253e1b8e5481cb2b817864045a8b7
>	Jun 6, 2024 @ 11:37:55.914	Setting network provider client successfully	set_network_provider	verify_contact_method.py	147	efb253e1b8e5481cb2b817864045a8b7
>	Jun 6, 2024 @ 11:37:55.899	Active subscription found	set_account_active_subscription	verify_contact_method.py	132	efb253e1b8e5481cb2b817864045a8b7
>	Jun 6, 2024 @ 11:37:55.896	Requester user is not blocked	set_is_user_blocked	verify_contact_method.py	96	-
>	Jun 6, 2024 @ 11:37:55.894	Setting the parsed phone number	set_parsed_phone_number	verify_contact_method.py	112	efb253e1b8e5481cb2b817864045a8b7
>	Jun 6, 2024 @ 11:37:55.873	Contact method yet to be verified	set_is_contact_already_verified	verify_contact_method.py	82	-
>	Jun 6, 2024 @ 11:37:55.850	API Invoked	__call__	logger.py	47	efb253e1b8e5481cb2b817864045a8b7

Don’t get lost in logs!

Most of the scenario there are two log streams (Info, Debug) configured and running parallely.



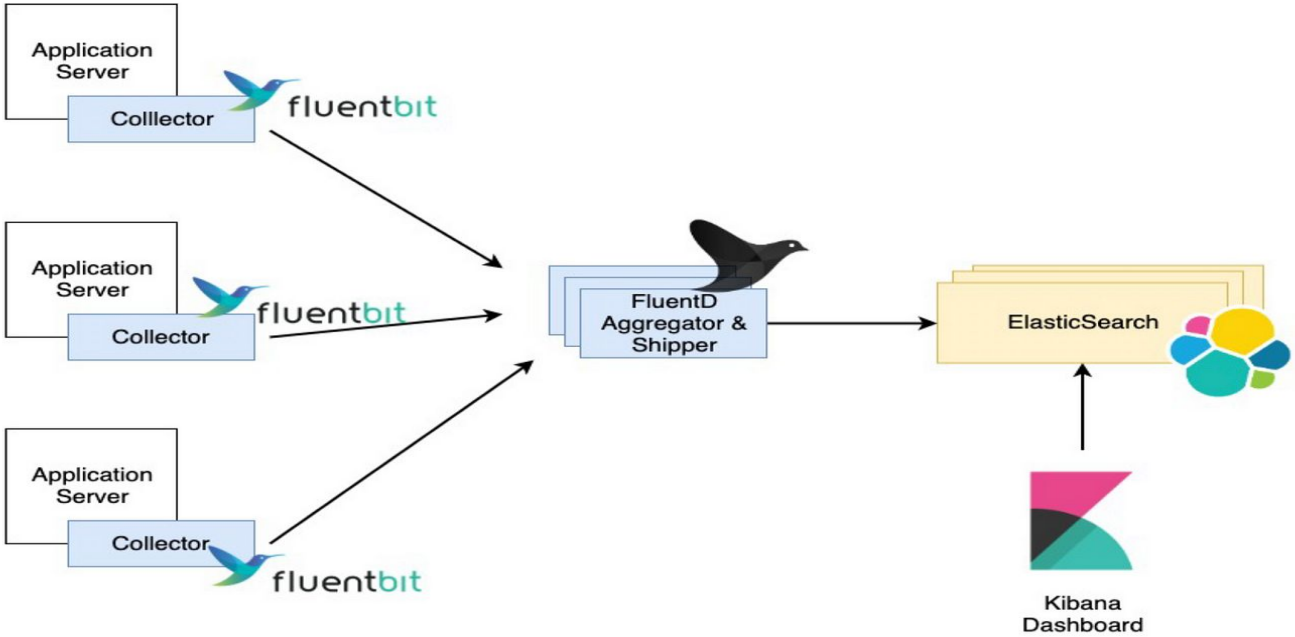
Common logging practices

There are many applications are available to aggregate and monitor logs.

Few of the most widely used open source logging stack are as below

1. ELK (ElasticSearch, Logstash, Kibana)
2. EFK (ElasticSearch, Fluentd, Kibana)
3. Grafana Loki

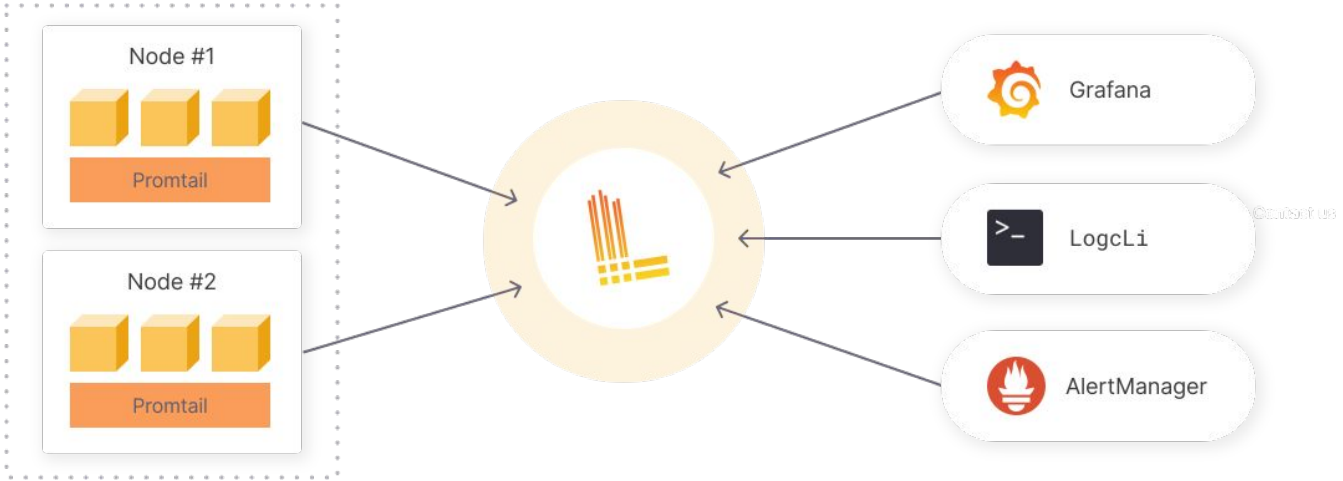
EFK


Application Server
Colllector
fluentbit
Application Server
Collector
fluentbit
Application Server
Collector
fluentbit
FluentD Aggregator & Shipper
ElasticSearch
Kibana Dashboard

Grafana Loki

Components

- **Promtail:** Log collector, forwards logs to Loki.
- **Loki:** Centralized log storage, optimized for log data.
- **Grafana:** Visualization tool, queries and visualizes logs from Loki.



zenduty

Thanks for listening.

Questions?

Looking for a cost-effective **facelift** in your incident management and response systems?
Check out our friendly neighbourhood alerting platform at:

www.zenduty.com