
PLARIUM

COMFORTABLE LOGGING

IN MICROSERVICES ENVIRONMENT

Serge Pitutin

.NET Developer

UDEV TECH MEET UPS

CONTENTS

WHY CARE ABOUT LOGGING?

WHY CARE ABOUT LOGGING?

- *Collect information about how the system is working.*
- *Collect business events & metrics.*
- *All team member should be able to view logs (developers, QA, PM, etc).*
- *Improve collaboration.*
- *Amount of debugging information can be huge.*
- *Server – agnostic data access (no RDP to production server!)*
- *Multiple environments? Double requirements.*
- *Microservices? This is heavy.*

WHAT INFORMATION SHOULD BE COLLECTED?

WHAT INFORMATION SHOULD BE COLLECTED?

- *Business events.*
- *Requests & responses.*
- *Errors & failures (with stack traces).*
- *Technical information (processing details, profiling data).*
- *No cleartext credentials!*

LOGS STORAGE

LOGS STORAGE

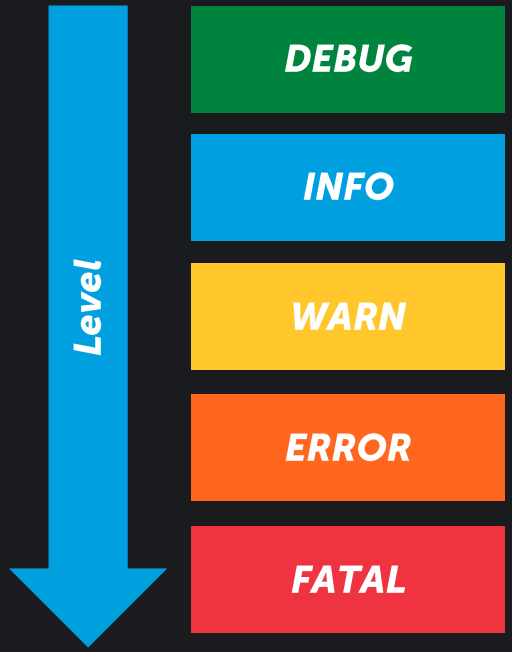
- *Centralized.*
- *HA-ready.*
- *Scalable.*
- *Support fast indexing.*
- *Support fast requests.*
- *Support structured logging.*
- *Local copy for emergency cases.*

LOGGING
BEST PRACTICES

LOGGING BEST PRACTICES

- *Consistent log levels.*
- *Structured logging.*
- *Write logs async.*
- *Identifiers (application instance identifier, correlation Id, transaction Id).*
- *UTC time.*
- *Centralized logging.*

CONSISTENT LOG LEVELS



Fine-grained informational events that are most useful to debug an application.

Informational messages that highlight the progress of the application.

Potentially harmful situations.

Error events that might still allow the application to continue running.

Very severe error events that will presumably lead the application to abort.

LOGGING ~~BEST~~ WORST PRACTICES: CONSISTENT LOG LEVELS

```
try
    {
        log.Info($"Started doing something ");
        //do something
        log.Info($"Intermediate results of doing something, user {userId} Invoice: {sum}");
        //do something more
        log.Info($"Completed doing something total invoice {sumTotal}");
    }
    catch (Exception ex)
    {
        log.Info($"Error trying to do something {ex.Message}");
    }
```

STRUCTURED LOGGING

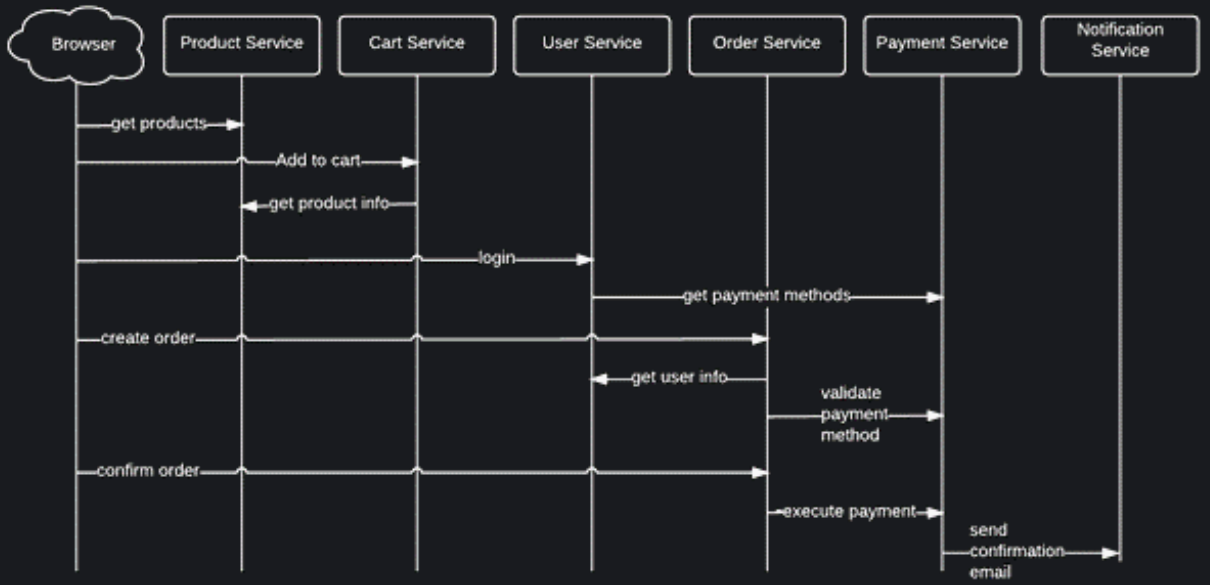
- **Messages.** *Be brief. No metadata here.*
- **Metadata**. *Include everything related.*
- **Errors**. *Include full exception details (as metadata). Include additional data.*
- *Persist same metadata schema across application.*

```
DEBUG 2018-06-06 16:00:00 – Incoming metrics data client:1234 [invoice:12.34]
```

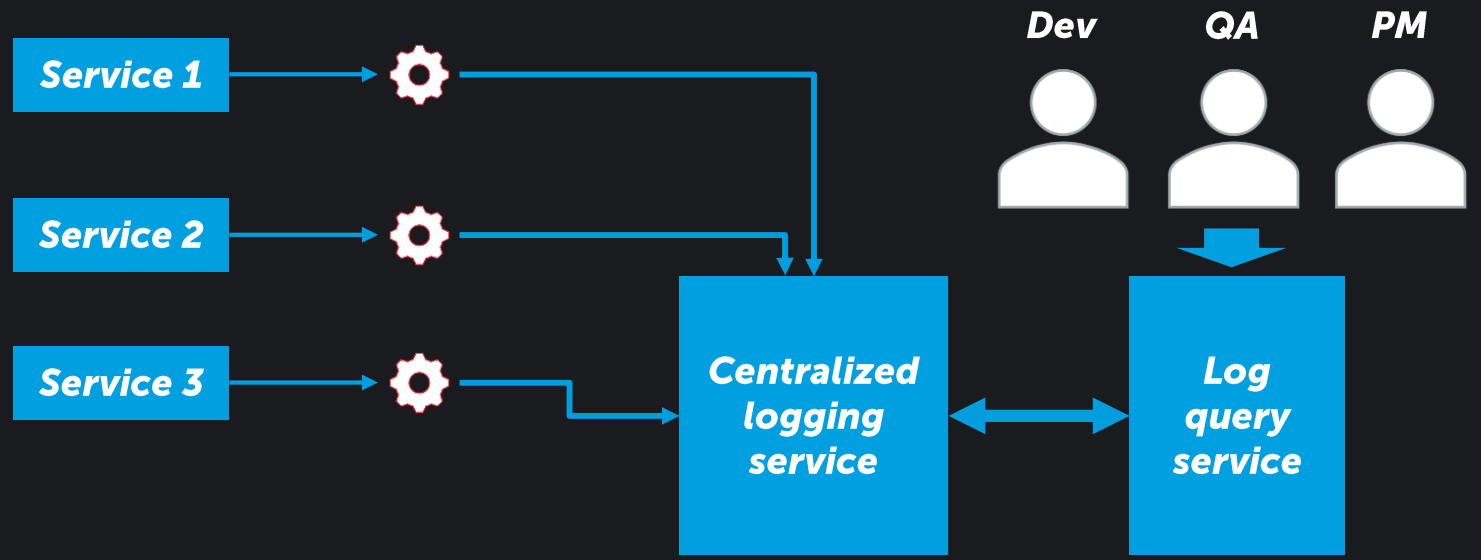
```
{
    "time": "2018-06-06 16:00:00.0000",
    "level": "DEBUG",
    "message": "Incoming metrics data",
    "client": 1234,
    "invoice": 12.34
}
```

CORRELATION ID

- *Generate a unique identifier for each transaction that can be used later to correlate events and trace your transactions easily between services.*



LOGS STORAGE: CENTRALIZED

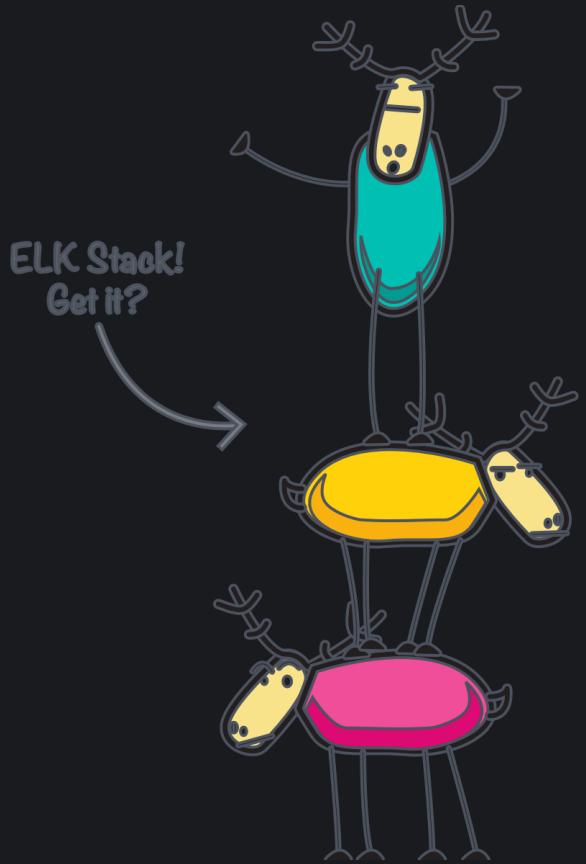


INTRODUCING ELK STACK

E *Elasticsearch*

L *Logstash*

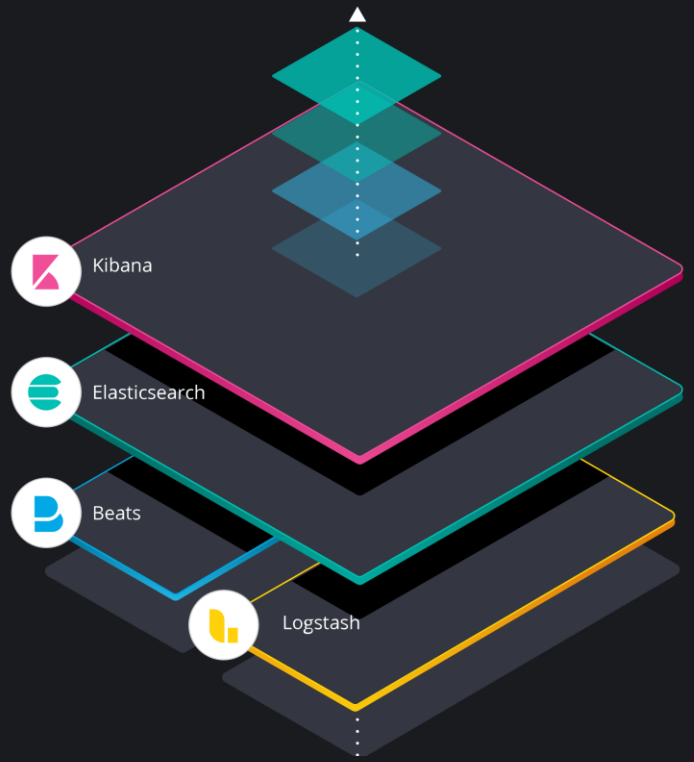
K *Kibana*



INTRODUCING ELK STACK

"ELK" is the acronym for three open source projects: Elasticsearch, Logstash, and Kibana.

- ***Elasticsearch** is a search and analytics engine.*
- ***Logstash** is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch.*
- ***Kibana** lets users visualize data with charts and graphs in Elasticsearch.*



LOGGING IMPLEMENTATION @ .NET

WITH NLOG

LOGGING IMPLEMENTATION

- *Building new abstraction layer: ILogger.*
- *Building Nlog target to send data directly to Elasticsearch instance.*
- *Building extensions for sending logs with metadata.*
- *Building layout renderers.*
- *Logstash.*

ILOGGER

```
namespace MarketingBI.Logging
{
    public interface ILogger
    {
        void Debug(string message);
        void Debug(string message, IDictionary<string, object> parameters);
        void Debug(string message, Exception exception, IDictionary<string, object> parameters);
        void Error(string message);
        void Error(string message, Exception exception, IDictionary<string, object> parameters);
        ....
        void Warn(string message);
        void Warn(string message, Exception exception);
        void Warn(string message, Exception exception, IDictionary<string, object> parameters);
    }
}
```

ELASTICSEARCH TARGET

- *Nest package.*
- *Support for structured logging.*
- *Support for CorrelationId.*

```
    var docData = logEvent
        .Properties
        .Where(x => !_excludedProperties.Contains(x.Key.ToString()))
        .ToDictionary(pair => pair.Key.ToString(), pair => pair.Value);

    if (docData.Any())
        doc.Data = docData;

    if (logEvent.Exception != null)
        doc.Exception = new ElasticSearchDocumentException(logEvent.Exception);

    if (CorrelationId != null)
        doc.CorrelationId = CorrelationId.Render(logEvent);

    descriptor.Index<ElasticSearchDocument>(i => i
        .Index(index)
        .Type(type)
        .Document(doc));
}

var response = _elasticClient.Bulk(descriptor);
```

EXTENSIONS

```
_logger.Trace($"Fraud user. Pixel is not fired. [{postBackData.Serialize()}]");

_logger.Trace($"Fraud user. Pixel is not fired. ", new Dictionary<string,string>
{
    {"postBackData",postBackData.Serialize()}
});
```

EXTENSIONS

```
_logger.Trace ($"Fraud user. Pixel is not fired. ", new { postBackData });

public static void Trace(this ILogger logger, string message, object data)
{
    var logData = ObjectToDictionaryConverter.GetProperties(data);
    logger.Trace(message, logData);
}
```

- *Use System.Reflection.Emit to reduce reflection usage.*

LAYOUT RENDERERS & ASYNC

- *Async wrapper*

```
<target name="elastic"
        type="AsyncWrapper"
        queueLimit="100000"
        overflowAction="Discard"
        timeToSleepBetweenBatches="10">
  <target type="ElasticSearch"
          layout="${message}${onexception:${newline}${newline}${exception:format=ToString,Data}}"
          uri="http://login:password@logging.company.com/"
          index="${elasticIndex}"
          documentType="logentry"
          timeout="2000"
          correlationId="${activityid}"
          />
</target>
```

- *Thread agnostic problem*

```
[LayoutRenderer("log-objects")]
[ThreadAgnostic]
0 references
public class LogObjectsLayoutRenderer : LayoutRenderer
{
    1 reference
    private static readonly JsonSerializerSettings _serializerSettings = new JsonSerializerSettings
    {
        NullValueHandling = NullValueHandling.Ignore
    };

    0 references
    protected override void Append(StringBuilder builder, LogEventInfo logEvent) => builder.Append(Get

    1 reference
    private static string GetData(LogEventInfo logEvent)
    {
        var props = logEvent.Properties;
        if (props.Count == 0 || (props.Count == 1 && props.Keys.Contains("_threadId")))
            return string.Empty;
```

LOGSTASH

- *Use Logstash*

  for business events data.

- *Use Logstash*

  for 3d party services.

```
input {
  file {…
  }
}

filter {…
}

output {
  elasticsearch {
    hosts => [ "http://login:password@logging.company.com" ]
    index => "tracker.inc-%{+yyyy.MM.dd}"
  }
}
```

LOGSTASH

```
55.3.244.1 GET /index.html 15824 0.043
```

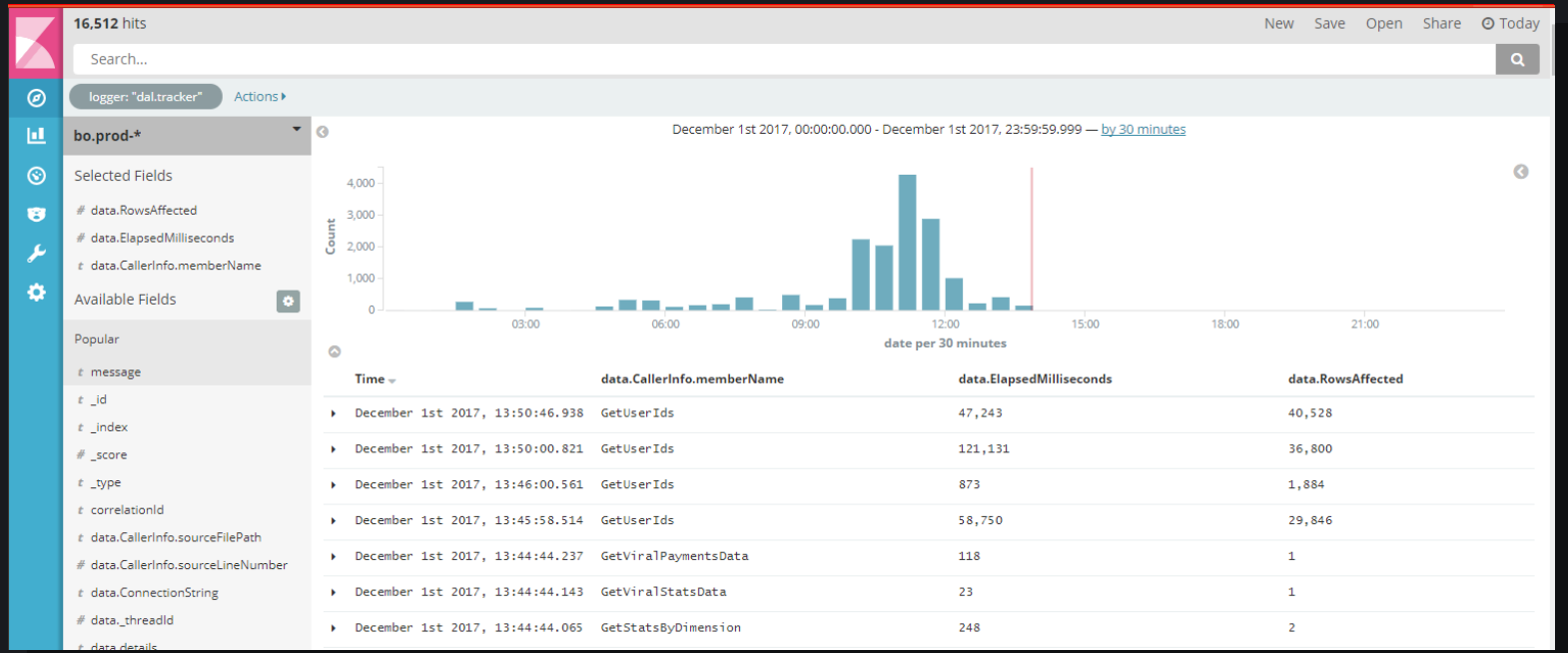
```
input {
  file {
    path => "/var/log/http.log"
  }
}
filter {
  grok {
    match => { "message" => "%{IP:client} %{WORD:method} %
    {URIPATHPARAM:request} %{NUMBER:bytes} %{NUMBER:duration}" }
  }
}
```

```
client: 55.3.244.1
method: GET
request: /index.html
bytes: 15824
duration: 0.043
```

VISUALIZING DATA WITH KIBANA

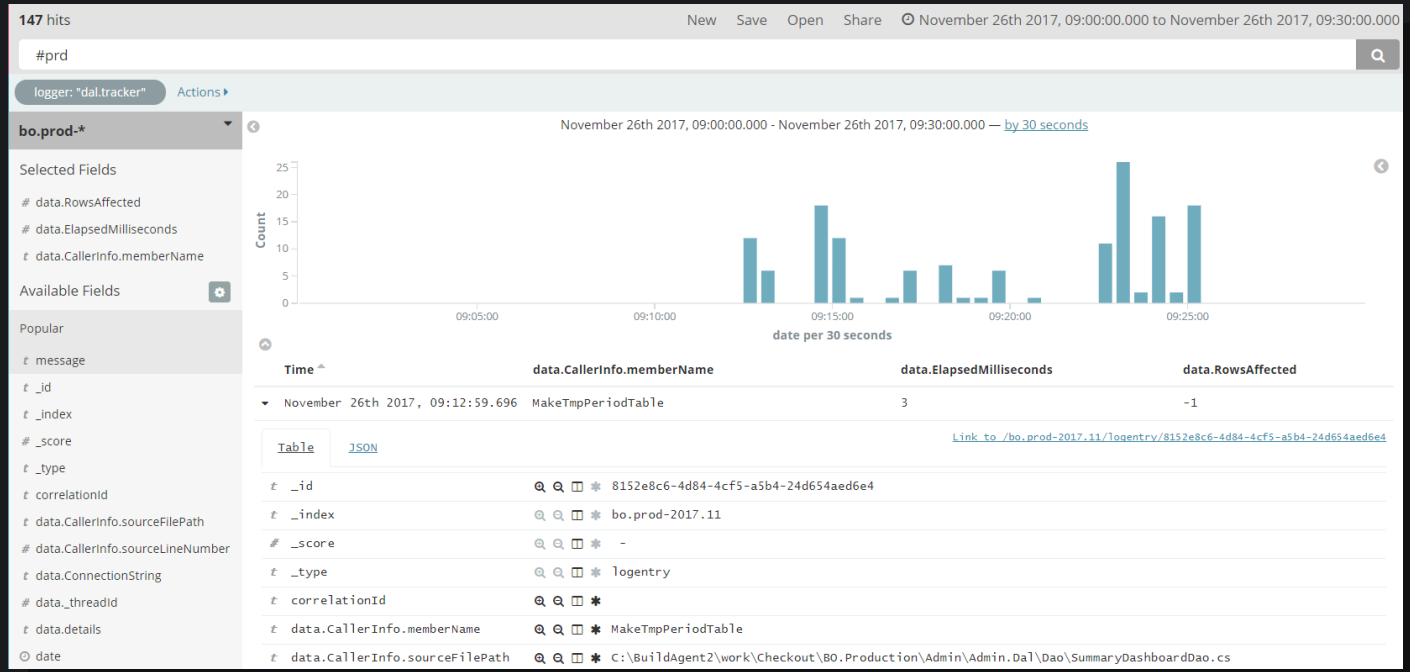
DISCOVER

Analyze raw data, filter, sort.



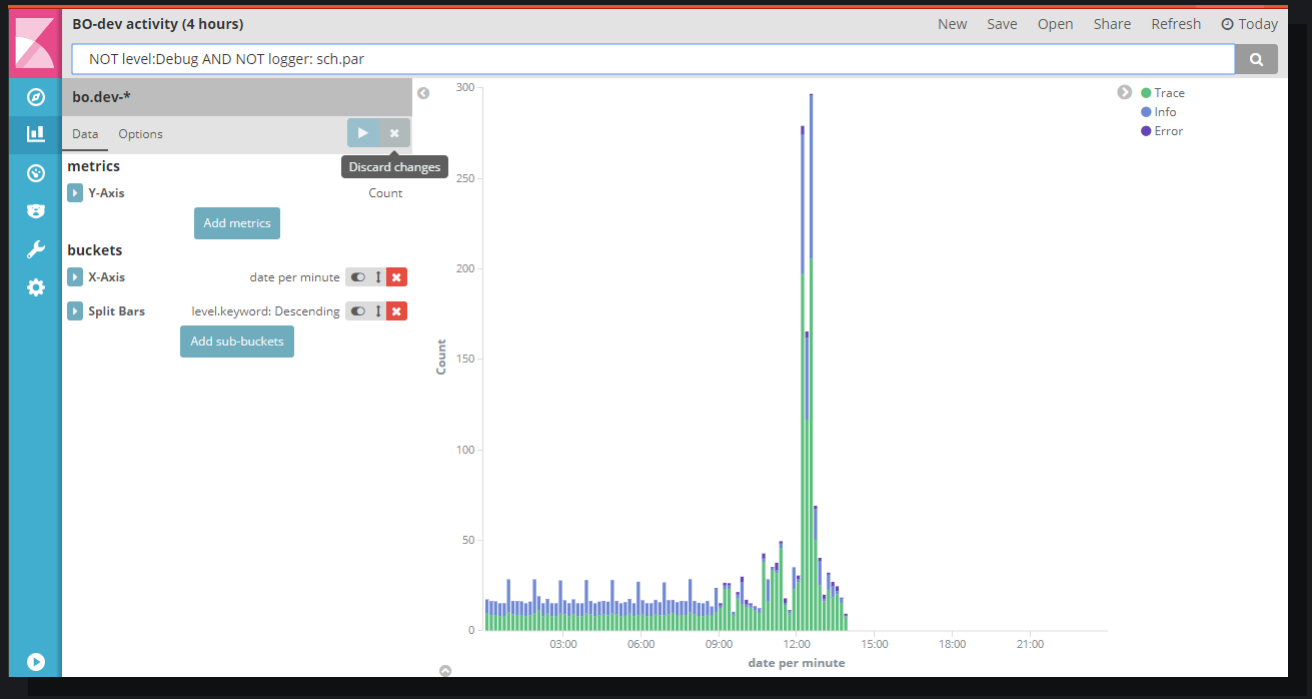
FILTER

Analyze raw data, filter, sort.



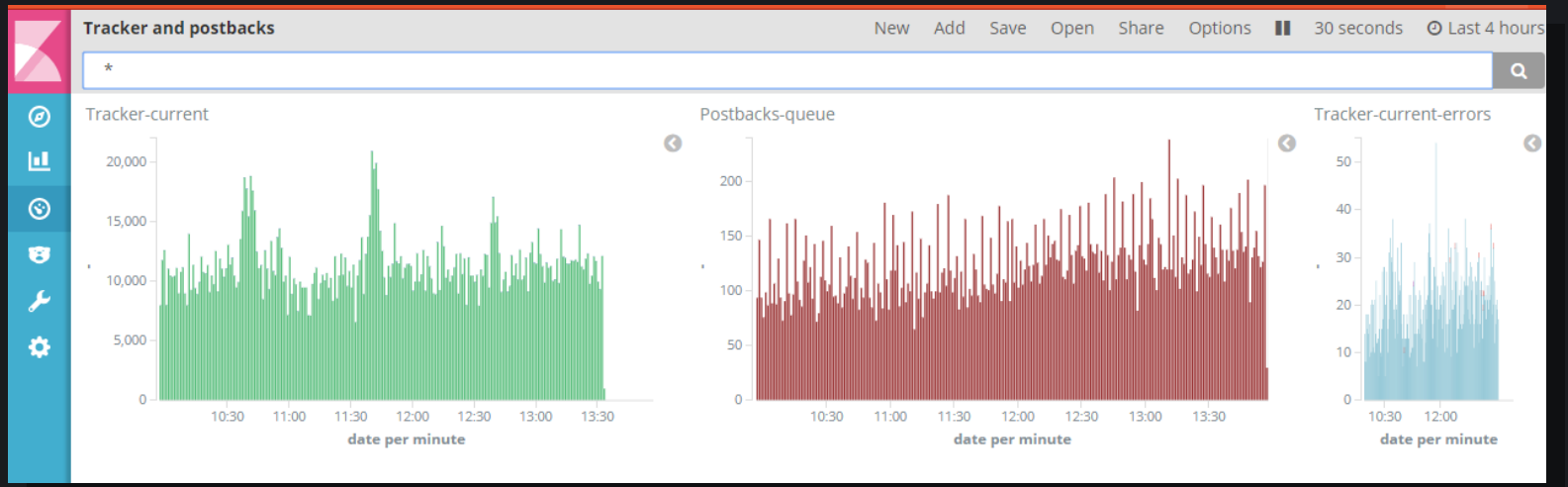
VISUALIZE

Aggregate, build graphs, tables.



DASHBOARD

Combine visualizations.



CONCLUSION

- *Structured centralized logging is a must.*
- *Prefer centralized logging, ElasticSearch is a good choice.*
- *Logstash & Kibana make logging comfortable.*

THANK YOU

TAKE THE WORLD