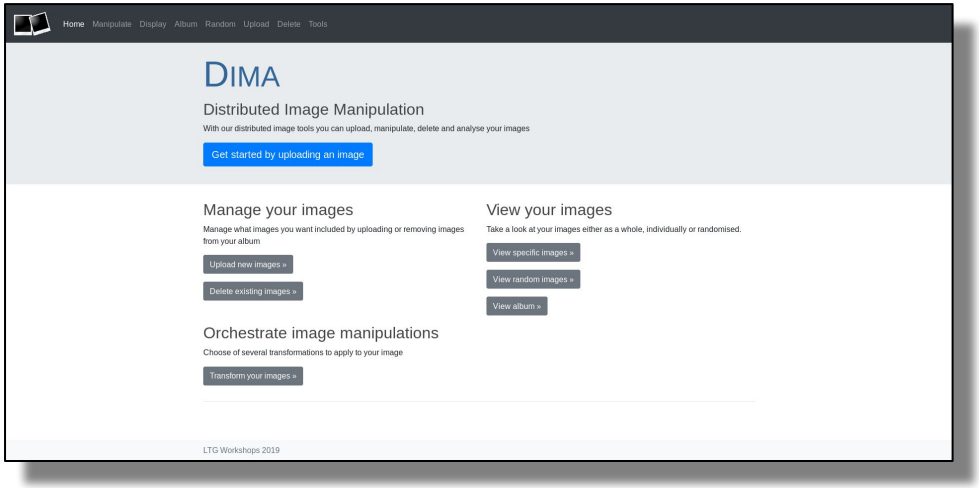
Exploring systems quality in a distributed world

With Abby Bangser and Benjamin Hofmann
Supported by Jon Barber

Welcome

- Join https://tlk.io/atd_o11y
- Open DIMA website from link in chat (http://46.101.170.74)
- Upload an image you think represents yourself
- Paste the ID of the image into the chat



Say hello!

(name, home city, tech role)

What are your observability goals?

What fears do you have for this workshop?

Understanding our domain

Create a model of the application which can help you design tests and identify risks

Sharing your domain models

As you share, let’s collect and group all the questions and risks that we have about the system

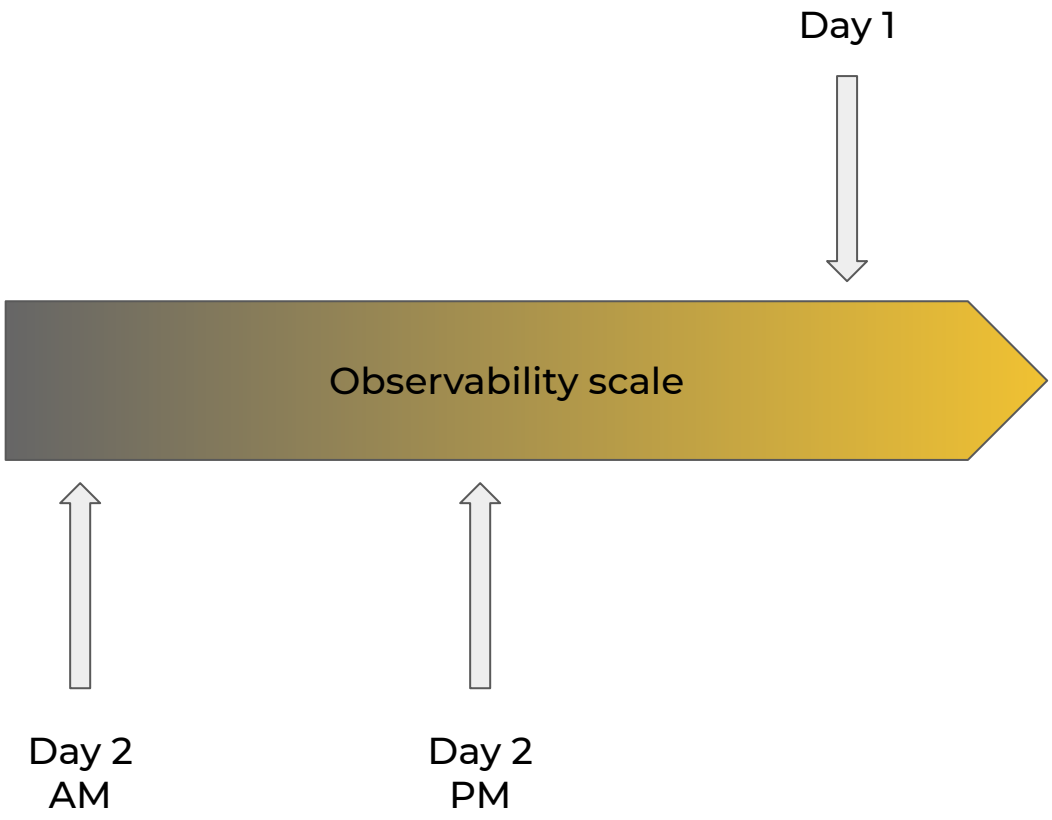
Our two days together

Goal:

To introduce the power of arbitrarily wide, context rich events and how to leverage them to explore a distributed environment

Takeaways:

- ➔ Identify what can impact levels of observability
- ➔ Exposure to tools and techniques that are used when exploring observable systems
- ➔ Understand how the industry has moved towards observability



Our day today

Goal:

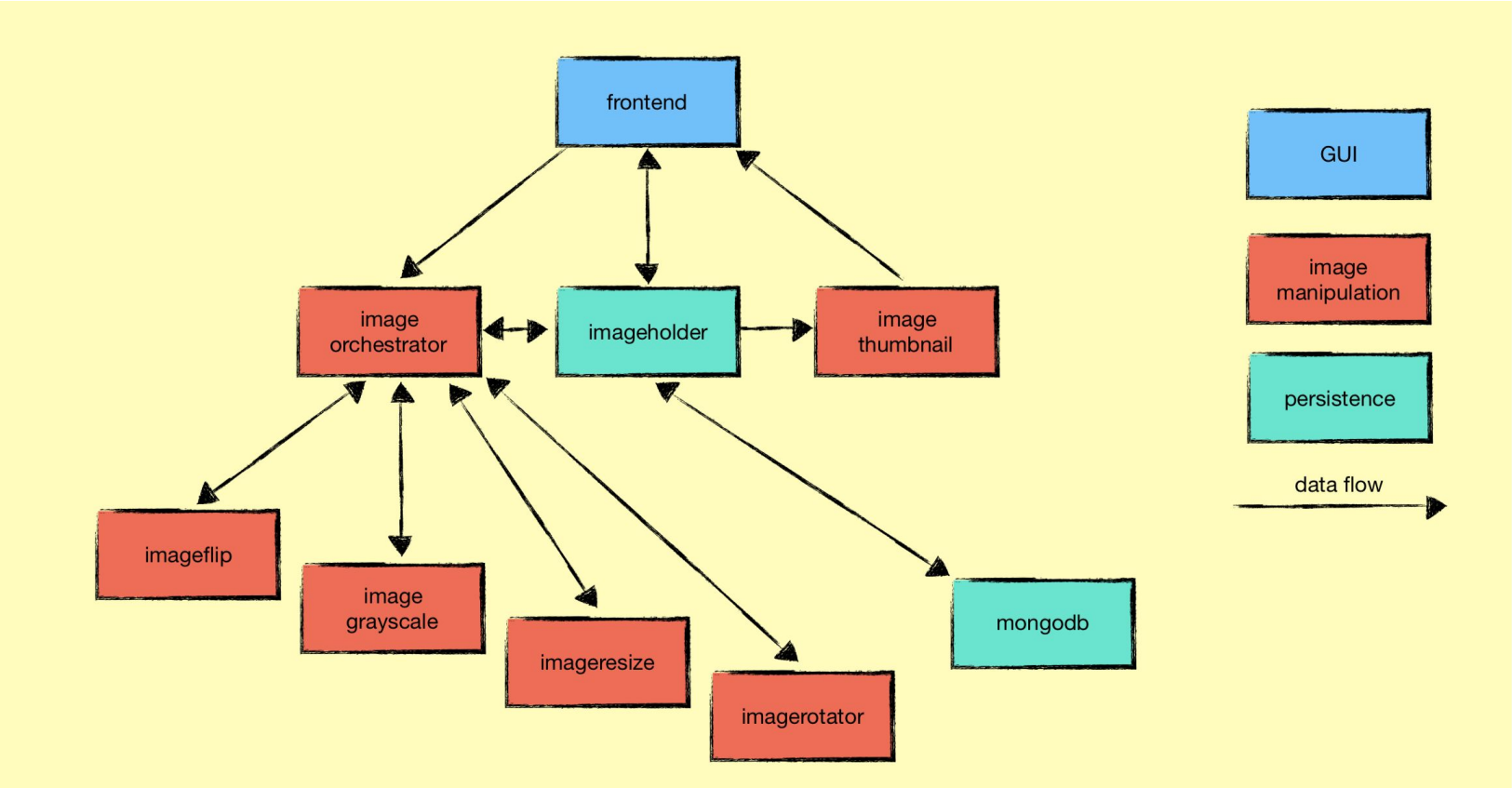
To introduce the power of arbitrarily wide, context rich events and how to leverage them to explore a distributed environment

Takeaways:

- ➔ Understanding "normal" for an application
- ➔ Experience with Kibana data visualisation tool
- ➔ Exposure to the data structures of logs, metrics and traces

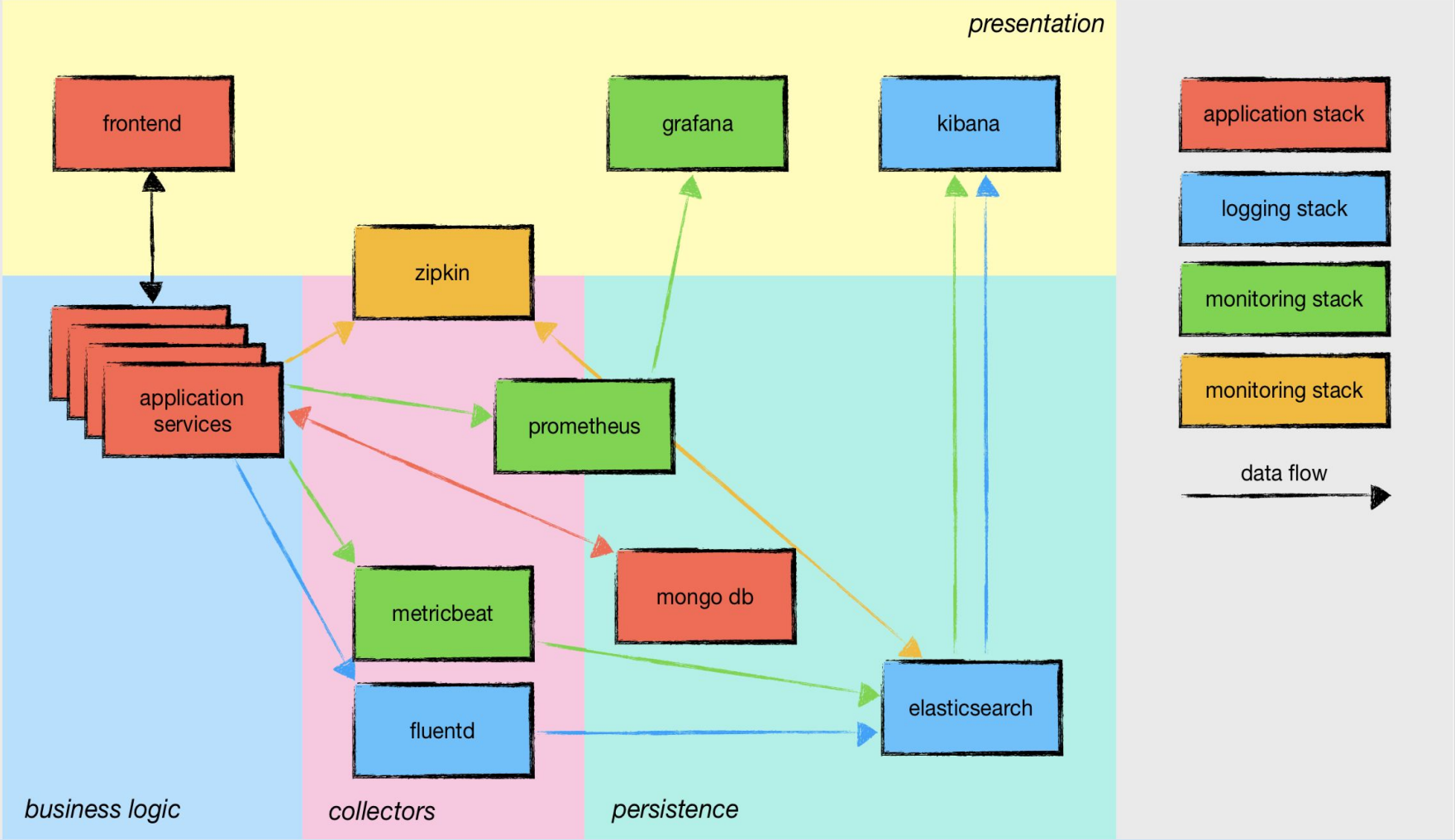
- ✅ Welcome
- ✅ Understanding our business domain
- ❑ Architecture and observability intro
- ❑ Observability within kibana
  - ❑ Explore APM
  - ❑ Dive into detailed logs and events
  - ❑ Understand the bigger trend of metrics
- ❑ More experience with logs
- ❑ More experience with metrics (prometheus)
- ❑ Preparing for tomorrow

Architecture



Truths about distributed systems...

- The entire system is never fully healthy
- Users and use cases are unpredictable
- It’s impossible to predict all kinds of failure states
- Failure can occur at every phase
- Debugging is a cornerstone activity


presentation
frontend
grafana
kibana
zipkin
application services
prometheus
mongo db
metricbeat
fluentd
elasticsearch
business logic
collectors
persistence
application stack
logging stack
monitoring stack
monitoring stack
data flow

Three common data types...

Logs

Metrics

Traces

Three common data types...

Logs

```
June 9th 2019, 23:21:47.313  container_id: 7539e37f488e61476c18aab4304e035148dc13463edcee0599ef42e1ebaa1a15 container_name: dima_imageholde
r_1 source: stdout message: To enable URLs as dynamic configuration sources, define System property archaius.c
onfigurationSource.additionalUrls or make config.properties available on classpath. service: imageholder @time
stamp: June 9th 2019, 23:21:47.313 logger_name: com.netflix.config.sources.URLConfigurationSource level: INFO
@log_name: 7539e37f488e _id: 3JdVPmsBVWqi44Upcum- _type: access_log _index: fluentd-20190609 _score: -
```

Metrics

Traces

Three common data types...

Logs: An immutable, timestamped record of discrete events that happened.


```
June 9th 2019, 23:21:47.313  container_id: 7539e37f488e61476c18aab4304e035148dc13463edcee0599ef42e1ebaa1a15 container_name: dima_imageholder_1 source: stdout message: To enable URLs as dynamic configuration sources, define System property archaius.configurationSource.additionalUrls or make config.properties available on classpath. service: imageholder @timestamp: June 9th 2019, 23:21:47.313 logger_name: com.netflix.config.sources.URLConfigurationSource level: INFO @log_name: 7539e37f488e _id: 3JdVPmsBVWqi44Upcum- _type: access_log _index: fluentd-20190609 _score: -
```


Metrics

Traces

Three common data representations...

Logs: An immutable, timestamped record of discrete events that happened.

```
▸ June 9th 2019, 23:21:47.313  container_id: 7539e37f488e61476c18aab4304e035148dc13463edcee0599ef42e1ebaa1a15 container_name: dima_imageholde
r_1 source: stdout message: To enable URLs as dynamic configuration sources, define System property archaius.c
onfigurationSource.additionalUrls or make config.properties available on classpath. service: imageholder @time
stamp: June 9th 2019, 23:21:47.313 logger_name: com.netflix.config.sources.URLConfigurationSource level: INFO
@log_name: 7539e37f488e _id: 3JdVPmsBVWqi44Upcum- _type: access_log _index: fluentd-20190609 _score: -
```

Metrics

```
# TYPE application_images_uploaded_total counter
application_images_uploaded_total{type="image/png",} 371.0
application_images_uploaded_total{type="image/jpeg",} 162.0
application_images_uploaded_total{type="image/tiff",} 333.0
application_images_uploaded_total{type="image/gif",} 303.0
```

Traces

Three common data representations...

Logs: An immutable, timestamped record of discrete events that happened.

```
▸ June 9th 2019, 23:21:47.313  container_id: 7539e37f488e61476c18aab4304e035148dc13463edcee0599ef42e1ebaa1a15 container_name: dima_imageholde
r_1 source: stdout message: To enable URLs as dynamic configuration sources, define System property archaius.c
onfigurationSource.additionalUrls or make config.properties available on classpath. service: imageholder @time
stamp: June 9th 2019, 23:21:47.313 logger_name: com.netflix.config.sources.URLConfigurationSource level: INFO
@log_name: 7539e37f488e _id: 3JdVPmsBVWqi44Upcum- _type: access_log _index: fluentd-20190609 _score: -
```

Metrics: a numeric representation of data measured over time as either a rising counter or a gauge (point in time value)

```
# TYPE application_images_uploaded_total counter
application_images_uploaded_total{type="image/png",} 371.0
application_images_uploaded_total{type="image/jpeg",} 162.0
application_images_uploaded_total{type="image/tiff",} 333.0
application_images_uploaded_total{type="image/gif",} 303.0
```

Traces

Three common data representations...

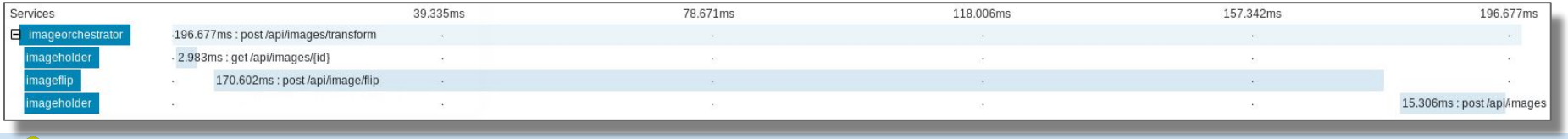
Logs: An immutable, timestamped record of discrete events that happened.

```
June 9th 2019, 23:21:47.313  container_id: 7539e37f488e61476c18aab4304e035148dc13463edcee0599ef42e1ebaa1a15 container_name: dima_imageholde
r_1 source: stdout message: To enable URLs as dynamic configuration sources, define System property archaius.c
onfigurationSource.additionalUrls or make config.properties available on classpath. service: imageholder @time
stamp: June 9th 2019, 23:21:47.313 logger_name: com.netflix.config.sources.URLConfigurationSource level: INFO
@log_name: 7539e37f488e _id: 3JdVPmsBVWqi44Upcum- _type: access_log _index: fluentd-20190609 _score: -
```

Metrics: a numeric representation of data measured over time as either a rising counter or a gauge (point in time value)

```
# TYPE application_images_uploaded_total counter
application_images_uploaded_total{type="image/png",} 371.0
application_images_uploaded_total{type="image/jpeg",} 162.0
application_images_uploaded_total{type="image/tiff",} 333.0
application_images_uploaded_total{type="image/gif",} 303.0
```

Traces



Three common data representations...

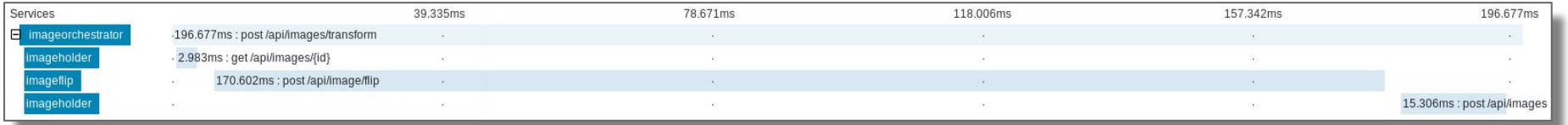
Logs: An immutable, timestamped record of discrete events that happened.

```
June 9th 2019, 23:21:47.313  container_id: 7539e37f488e61476c18aab4304e035148dc13463edcee0599ef42e1ebaa1a15 container_name: dima_imageholde
r_1 source: stdout message: To enable URLs as dynamic configuration sources, define System property archaius.c
onfigurationSource.additionalUrls or make config.properties available on classpath. service: imageholder @time
stamp: June 9th 2019, 23:21:47.313 logger_name: com.netflix.config.sources.URLConfigurationSource level: INFO
@log_name: 7539e37f488e _id: 3JdVPmsBVWqi44Upcum- _type: access_log _index: fluentd-20190609 _score: -
```

Metrics: a numeric representation of data measured over time as either a rising counter or a gauge (point in time value)

```
# TYPE application_images_uploaded_total counter
application_images_uploaded_total{type="image/png",} 371.0
application_images_uploaded_total{type="image/jpeg",} 162.0
application_images_uploaded_total{type="image/tiff",} 333.0
application_images_uploaded_total{type="image/gif",} 303.0
```

Traces: representation of a series of events that describe the end-to-end request



Our day today

Goal:

To introduce the power of arbitrarily wide, context rich events and how to leverage them to explore a distributed environment

Takeaways:

- ➔ Understanding "normal" for an application
- ➔ Experience with Kibana data visualisation tool
- ➔ Exposure to the data structures of logs, metrics and traces

- ✅ Welcome
- ✅ Understanding our business domain
- ✅ Architecture and observability intro
- ❑ Observability within kibana
  - ❑ Explore APM
  - ❑ Dive into detailed logs and events
  - ❑ Understand the bigger trend of metrics
- ❑ More experience with logs
- ❑ More experience with metrics (prometheus)
- ❑ Preparing for tomorrow

Coffee break

Our day today

Goal:

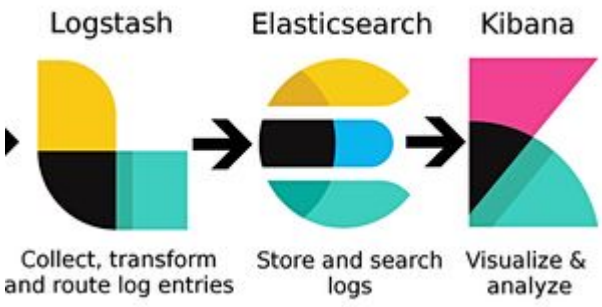
To introduce the power of arbitrarily wide, context rich events and how to leverage them to explore a distributed environment

Takeaways:

- ➔ Understanding "normal" for an application
- ➔ Experience with Kibana data visualisation tool
- ➔ Exposure to the data structures of logs, metrics and traces

- ✅ Welcome
- ✅ Understanding our business domain
- ✅ Architecture and observability intro
- ❑ Observability within kibana
  - ❑ Explore APM
  - ❑ Dive into detailed logs and events
  - ❑ Understand the bigger trend of metrics
- ❑ More experience with logs
- ❑ More experience with metrics (prometheus)
- ❑ Preparing for tomorrow

Time to get down and dirty with the data



46.101.170.74:5601

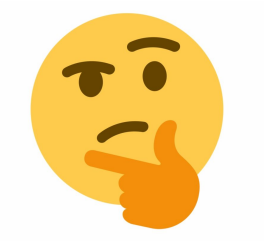
And join us on the **APM** page

Not much to see here,
but what about within a service?

Select the **`imageflip`** service and explore the available information

Not much to see here,
but what about within a service?

Select the **`imageflip`** service and explore the available information



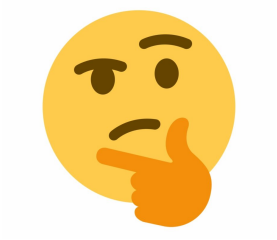
Where would we get more detailed information about the execution of image flips by the service?

As calm as a service may appear at the surface, we need to dive below to understand more

Select the `ImageController#flipImage` transaction and explore the `Transactions duration distribution` work

As calm as a service may appear at the surface, we need to dive below to understand more

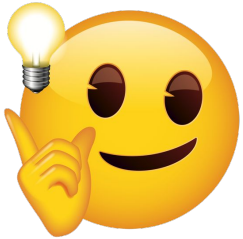
Select the `ImageController#flipImage` transaction and explore the `Transactions duration distribution` work



Why does that gap exist, how could we review uptime?

Looking at uptime

Select the `Actions > View monitor status` and open in a new tab



Ah that downtime blip looks at about the same time!

Drilling into a specific trace (sample)

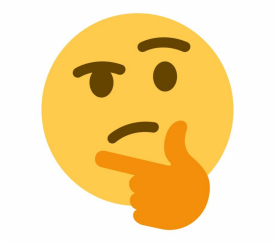
Returning to the trace details...

Select the **`2nd histogram bucket`**, and focus on **`Transaction sample`** data

Drilling into a specific trace (sample)

Returning to the trace details...

Select the **`2nd histogram bucket`**, and focus on **`Transaction sample`** data



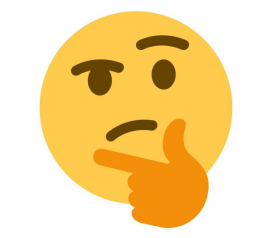
What does that **`% of trace`** mean?

imageFlip is just a small cog in a big wheel. What ever happened to that big distributed system?

Select the `View full trace` and explore the other pieces of the puzzle

imageFlip is just a small cog in a big wheel. What ever happened to that big distributed system?

Select the `View full trace` and explore the other pieces of the puzzle



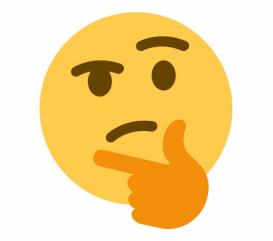
How would we get more details on these spans?

Now that we know where, how do we learn about what

Select the `Action > View trace logs` what does this new view give us?

Now that we know where, how do we learn about what

Select the **`Action > View trace logs`** what does this new view give us?



How can we use **`EVENT`** lines different from non-**`EVENT`** lines?

Our day today

Goal:

To introduce the power of arbitrarily wide, context rich events and how to leverage them to explore a distributed environment

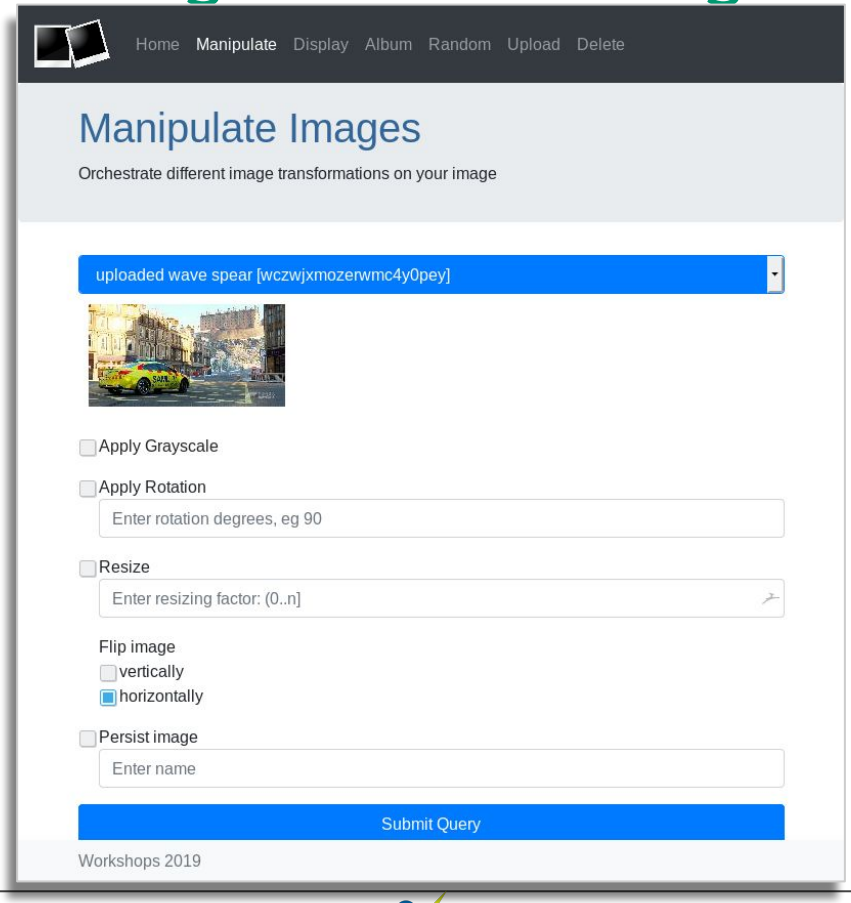
Takeaways:

- ➔ Understanding "normal" for an application
- ➔ Experience with Kibana data visualisation tool
- ➔ Exposure to the data structures of logs, metrics and traces

- ✅ Welcome
- ✅ Understanding our business domain
- ✅ Architecture and observability intro
- ✅ Observability within kibana
  - ✅ Explore APM
  - ❑ Dive into detailed logs and events
  - ❑ Understand the bigger trend of metrics
- ❑ More experience with logs
- ❑ More experience with metrics (prometheus)
- ❑ Preparing for tomorrow

Rewind…how are logs written during a request?

Rewind…how are logs written during a request?



Detailing how logs get written during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(@RequestParam("image") MultipartFile file,
                                @RequestParam(value = "vertical") Boolean vertical,
                                @RequestParam(value = "horizontal") Boolean horizontal) {

     if (file.getContentType() != null) {
        LOGGER.warn("Wrong content type uploaded: {}", file.getContentType());
        return new ResponseEntity<>("Wrong content type uploaded: " + file.getContentType());
    }

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

    if (flippedImage == null) {
        return new ResponseEntity<>("Failed to flip image", HttpStatus.INTERNAL_SERVER_ERROR);
    }

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

Detailing how logs get written during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(@RequestParam("image") MultipartFile file,
                                @RequestParam(value = "vertical") Boolean vertical,
                                @RequestParam(value = "horizontal") Boolean horizontal) {

    if (file.getContentType() != null) {
        LOGGER.warn("Wrong content type uploaded: {}", file.getContentType());
        return new ResponseEntity<>("Wrong content type uploaded: " + file.getContentType());
    }

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

    if (flippedImage == null) {
        return new ResponseEntity<>("Failed to flip image", HttpStatus.INTERNAL_SERVER_ERROR);
    }

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

Detailing how logs get written during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(@RequestParam("image") MultipartFile file,
                                @RequestParam(value = "vertical") Boolean vertical,
                                @RequestParam(value = "horizontal") Boolean horizontal) {

    if (file.getContentType() != null) {
        LOGGER.warn("Wrong content type uploaded: {}", file.getContentType());
        return new ResponseEntity<>("Wrong content type uploaded: " + file.getContentType());
    }

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

    if (flippedImage == null) {
        return new ResponseEntity<>("Failed to flip image", HttpStatus.INTERNAL_SERVER_ERROR);
    }

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

Detailing how logs get written during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(@RequestParam("image") MultipartFile file,
                                @RequestParam(value = "vertical") Boolean vertical,
                                @RequestParam(value = "horizontal") Boolean horizontal) {

    if (file.getContentType() != null) {
        LOGGER.warn("Wrong content type uploaded: {}", file.getContentType());
        return new ResponseEntity<>("Wrong content type uploaded: " + file.getContentType());
    }

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

    if (flippedImage == null) {
        return new ResponseEntity<>("Failed to flip image", HttpStatus.INTERNAL_SERVER_ERROR);
    }

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

Detailing how logs get written during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(@RequestParam("image") MultipartFile file,
                                @RequestParam(value = "vertical") Boolean vertical,
                                @RequestParam(value = "horizontal") Boolean horizontal) {

    if (file.getContentType() != null) {
        LOGGER.warn("Wrong content type uploaded: {}", file.getContentType());
        return new ResponseEntity<>("Wrong content type uploaded: " + file.getContentType());
    }

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

    if (flippedImage == null) {
        return new ResponseEntity<>("Failed to flip image", HttpStatus.INTERNAL_SERVER_ERROR);
    }

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

Detailing how logs get written during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(@RequestParam("image") MultipartFile file,
                                @RequestParam(value = "vertical") Boolean vertical,
                                @RequestParam(value = "horizontal") Boolean horizontal) {

    if (file.getContentType() != null) {
        LOGGER.warn("Wrong content type uploaded: {}", file.getContentType());
        return new ResponseEntity<>("Wrong content type uploaded: " + file.getContentType());
    }

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

    if (flippedImage == null) {
        return new ResponseEntity<>("Failed to flip image", HttpStatus.INTERNAL_SERVER_ERROR);
    }

    LOGGER.info("Successfully flipped image id: {}", file.getId()");
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

Detailing how logs get written during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(@RequestParam("image") MultipartFile file,
                                @RequestParam(value = "vertical") Boolean vertical,
                                @RequestParam(value = "horizontal") Boolean horizontal) {

    if (file.getContentType() != null) {
        LOGGER.warn("Wrong content type uploaded: {}", file.getContentType());
        return new ResponseEntity<>("Wrong content type uploaded: " + file.getContentType());
    }

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

    if (flippedImage == null) {
        return new ResponseEntity<>("Failed to flip image", HttpStatus.INTERNAL_SERVER_ERROR);
    }

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

Log outputs

	Time	message
>	Oct 21, 2019 @ 20:17:57.899	Successfully flipped image id: f1eqrievdiwxt0d7vknf
>	Oct 21, 2019 @ 20:17:57.822	Receiving image/png image to flip.

In contrast, how an event is created during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(...) {
    EVENT.addField("content.type", file.getContentType());
    EVENT.addField("action", "flip");
    EVENT.addField("image_id", file.getId());
    EVENT.addField("flip_vertical", vertical);
    EVENT.addField("flip_horizontal", horizontal);
...

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

...

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    EVENT.addField("action.success", "true");
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

In contrast, how an event is created during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(...) {
    EVENT.addField("content.type", file.getContentType());
    EVENT.addField("action", "flip");
    EVENT.addField("image_id", file.getId());
    EVENT.addField("flip_vertical", vertical);
    EVENT.addField("flip_horizontal", horizontal);
...

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

...

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    EVENT.addField("action.success", "true");
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

In contrast, how an event is created during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(...) {
    EVENT.addField("content.type", file.getContentType());
    EVENT.addField("action", "flip");
    EVENT.addField("image_id", file.getId());
    EVENT.addField("flip_vertical", vertical);
    EVENT.addField("flip_horizontal", horizontal);
...

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

...

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    EVENT.addField("action.success", "true");
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

In contrast, how an event is created during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(...) {
    EVENT.addField("content.type", file.getContentType());
    EVENT.addField("action", "flip");
    EVENT.addField("image_id", file.getId());
    EVENT.addField("flip_vertical", vertical);
    EVENT.addField("flip_horizontal", horizontal);
...

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

...

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    EVENT.addField("action.success", "true");
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

In contrast, how an event is created during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(...) {
    EVENT.addField("content.type", file.getContentType());
    EVENT.addField("action", "flip");
    EVENT.addField("image_id", file.getId());
    EVENT.addField("flip_vertical", vertical);
    EVENT.addField("flip_horizontal", horizontal);
...

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

...

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    EVENT.addField("action.success", "true");
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

In contrast, how an event is created during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(...) {
    EVENT.addField("content.type", file.getContentType());
    EVENT.addField("action", "flip");
    EVENT.addField("image_id", file.getId());
    EVENT.addField("flip_vertical", vertical);
    EVENT.addField("flip_horizontal", horizontal);
...

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

...

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    EVENT.addField("action.success", "true");
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

In contrast, how an event is created during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(...) {
    EVENT.addField("content.type", file.getContentType());
    EVENT.addField("action", "flip");
    EVENT.addField("image_id", file.getId());
    EVENT.addField("flip_vertical", vertical);
    EVENT.addField("flip_horizontal", horizontal);
...

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

...

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    EVENT.addField("action.success", "true");
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

In contrast, how an event is created during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(...) {
    EVENT.addField("content.type", file.getContentType());
    EVENT.addField("action", "flip");
    EVENT.addField("image_id", file.getId());
    EVENT.addField("flip_vertical", vertical);
    EVENT.addField("flip_horizontal", horizontal);
...

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

...

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    EVENT.addField("action.success", "true");
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

In contrast, how an event is created during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(...) {
    EVENT.addField("content.type", file.getContentType());
    EVENT.addField("action", "flip");
    EVENT.addField("image_id", file.getId());
    EVENT.addField("flip_vertical", vertical);
    EVENT.addField("flip_horizontal", horizontal);
...

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

...

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    EVENT.addField("action.success", "true");
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

In contrast, how an event is created during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(...) {
    EVENT.addField("content.type", file.getContentType());
    EVENT.addField("action", "flip");
    EVENT.addField("image_id", file.getId());
    EVENT.addField("flip_vertical", vertical);
    EVENT.addField("flip_horizontal", horizontal);
...

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

...

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    EVENT.addField("action.success", "true");
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

In contrast, how an event is created during a request

```
@PostMapping("flip")
public ResponseEntity flipImage(...) {
    EVENT.addField("content.type", file.getContentType());
    EVENT.addField("action", "flip");
    EVENT.addField("image_id", file.getId());
    EVENT.addField("flip_vertical", vertical);
    EVENT.addField("flip_horizontal", horizontal);
...

    LOGGER.info("Receiving {} image to flip.", file.getContentType());
    byte[] flippedImage = imageService.flip(file, vertical, horizontal);

...

    LOGGER.info("Successfully flipped image id: {}", file.getId());
    EVENT.addField("action.success", "true");
    return new ResponseEntity<>(flippedImage, headers, HttpStatus.OK);
}
```

Comparing the outputs

	Time	message
>	Oct 21, 2019 @ 20:17:57.899	Successfully flipped image id: f1eqrievdiwxt0d7vknf
>	Oct 21, 2019 @ 20:17:57.822	Receiving image/png image to flip.

Multiple logs

A single event

Expanded document

Table JSON

🕒	@timestamp	Oct 21, 2019 @ 20:17:57.822
t	action	flip
t	action.success	true
t	content.type	image/png
t	image.id	5dae0465b43b742b635bb0016eb49014
t	flip.horizontal	false
t	flip.vertical	true

Our day today

Goal:

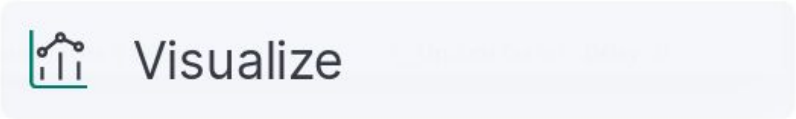
To introduce the power of arbitrarily wide, context rich events and how to leverage them to explore a distributed environment

Takeaways:

- ➔ Understanding "normal" for an application
- ➔ Experience with Kibana data visualisation tool
- ➔ Exposure to the data structures of logs, metrics and traces

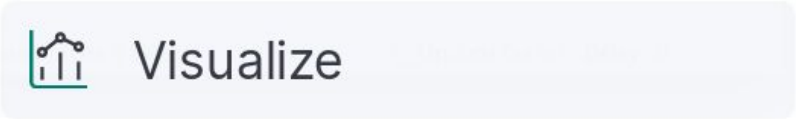
- ✅ Welcome
- ✅ Understanding our business domain
- ✅ Architecture and observability intro
- ✅ Observability within kibana
  - ✅ Explore APM
  - ✅ Dive into detailed logs and events
  - ❑ Understand the bigger trend of metrics
- ❑ More experience with logs
- ❑ More experience with metrics (prometheus)
- ❑ Preparing for tomorrow

Using those event details, let’s review trends over time

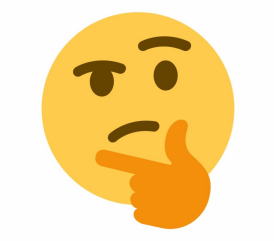


Create a new **`line graph`** visualisation based on our **`logs-*`** index

Using those event details, let’s review trends over time



Create a new `line graph` visualisation based on our `logs-*` index



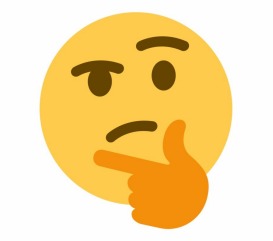
How do we get a count of requests over time?

Count is interesting, but what about latency?

Change from count to `p50, p95, p99` latency based on `span.duration_ms`

Count is interesting,
but what about latency?

Change from count to `p50, p95, p99` latency based on `span.duration_ms`



What if we wanted to see how `content_type` impacts latency

Latency over time sure…
but how about per image type?

Add a new `split series bucket` by the term `span.content_type`

And then...

And then...just kidding, it’s lunch time!

Our day today

Goal:

To introduce the power of arbitrarily wide, context rich events and how to leverage them to explore a distributed environment

Takeaways:

- ➔ Understanding "normal" for an application
- ➔ Experience with Kibana data visualisation tool
- ➔ Exposure to the data structures of logs, metrics and traces

- ✅ Welcome
- ✅ Understanding our business domain
- ✅ Architecture and observability intro
- ✅ Observability within kibana
  - ✅ Explore APM
  - ✅ Dive into detailed logs and events
  - ✅ Understand the bigger trend of metrics
- ❑ More experience with logs
- ❑ More experience with metrics (prometheus)
- ❑ Preparing for tomorrow

Deeper dive on **logging** with **Kibana** (ELK):

`46.101.170.74:5601`

Some Kibana tips and tricks

Discover

Filters 1 | Search | KQL | Last 15 minutes | Show dates

container_name: dima_imageorchestrator_1 × | + Add filter

logs-*

Selected fields

t transformationType

Available fields

Popular

t factor

t message

t service

t span.duration_ms

t @tag

@timestamp

t X-B3-ParentSpanId

t X-B3-SpanId

t X-B3-TraceId

t X-Span-Export

t _id

t _index

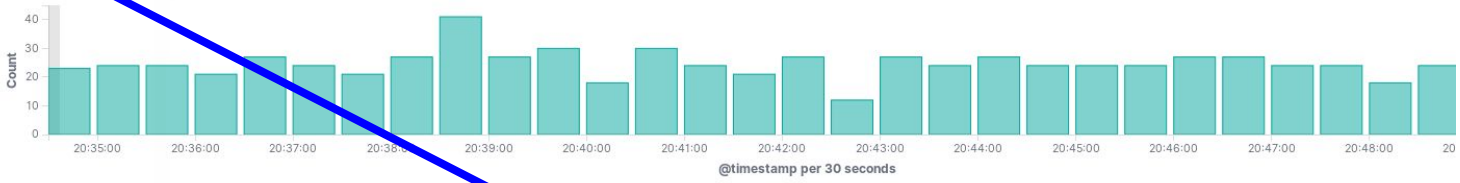
_score

t _type

t container_id

t container_name

Nov 2, 2019 @ 20:34:37.023 - Nov 2, 2019 @ 20:49:37.023 — Auto



Time	transformationType
Nov 2, 2019 @ 20:49:33.081	[rotate]

Column view

Expanded document | View surrounding documents | View sin

Table | JSON

t @tag	docker.stdout
@timestamp	Nov 2, 2019 @ 20:49:33.081
t X-B3-SpanId	3afa63aa55036e43
t X-B3-TraceId	5dbdebdd0c6cfe2e3afa63aa55036e43
t X-Span-Export	true
t _id	WVHhLW4BxHtNNOoCWiRA
t _index	logs-20191102
# _score	-
t _type	_doc
t container_id	a7ea6cf0ad1a70c54ce1b3e01d36[illegible]a7adbe8d2af6fea1809eef5782fad10
t container_name	dima_imageorchestrator_1

Filtering

Exploring with logs...

- Understanding log aggregation
  - Compare two different services, what value is there having logs for both in the same search
- Add a custom filter
  - Make a filter based on different field types (string, date, number)
- Working with time
  - Find the oldest log we have available in kibana
  - Select a specific time range using the graph as your guide
- Collaborating with Kibana
  - Make a filter based on different field try saving it and sharing with a teammate
  - What scenarios would best fit an “Absolute” time frame vs a “Relative” when sharing queries?

Exploring with logs...

- ➔ How did that feel?
- ➔ What similarities and differences did you feel from metrics?
- ➔ What do you want to ask next?
- ➔ What tool would best answer that question?

Our day today

Goal:

To introduce the power of arbitrarily wide, context rich events and how to leverage them to explore a distributed environment

Takeaways:

- ➔ Understanding "normal" for an application
- ➔ Experience with Kibana data visualisation tool
- ➔ Exposure to the data structures of logs, metrics and traces

- ✅ Welcome
- ✅ Understanding our business domain
- ✅ Architecture and observability intro
- ✅ Observability within kibana
  - ✅ Explore APM
  - ✅ Dive into detailed logs and events
  - ✅ Understand the bigger trend of metrics
- ✅ More experience with logs
- ❑ More experience with metrics (prometheus)
- ❑ Preparing for tomorrow

Coffee break

Our day today

Goal:

To introduce the power of arbitrarily wide, context rich events and how to leverage them to explore a distributed environment

Takeaways:

- ➔ Understanding "normal" for an application
- ➔ Experience with Kibana data visualisation tool
- ➔ Exposure to the data structures of logs, metrics and traces

- ✅ Welcome
- ✅ Understanding our business domain
- ✅ Architecture and observability intro
- ✅ Observability within kibana
  - ✅ Explore APM
  - ✅ Dive into detailed logs and events
  - ✅ Understand the bigger trend of metrics
- ✅ More experience with logs
- ❑ More experience with metrics (prometheus)
- ❑ Preparing for tomorrow

Answer a **metrics** based question by using **Prometheus**:

`46.101.170.74.48:9090`

Some Prometheus tips and tricks

35.178.167.176:9090/targets

Prometheus Alerts Graph Status Help

- Runtime & Build Information
- Command-Line Flags
- Configuration
- Rules
- Targets
- Service Discovery

Targets

All Unhealthy

elasticsearch (1/1 up) show

Endpoint	State	Labels
http://elasticsearch-exporter:9108/metrics	UP	instance="elasticsearch-exporter:9108" job="elasticsearch"

elcep (0/1 up) show less

Endpoint	State	Labels
http://elcep:8080/metrics	DOWN	instance="elcep:8080" job="elcep"

grafana (1/1 up) show less

Endpoint	State	Labels
http://grafana:3000/metrics	UP	instance="grafana:3000" job="grafana"

imageflip (1/1 up) show less

Endpoint	State	Labels
http://imageflip:8080/actuator/prometheus	UP	instance="imageflip:8080" job="imageflip"

35.178.167.176:9090/graph?g0.range_input=1h&g0.expr=application_image_transformed_total&g0.tab=1

Prometheus Alerts Graph Status Help

Enable query history

application_image_transformed_total

Execute - insert metric at cursor -

Graph Console

Moment

Element

application_image_transformed_total{instance="imageorchestrator:8080",job="imageorchestrator",type="image/gif"}

application_image_transformed_total{instance="imageorchestrator:8080",job="imageorchestrator",type="image/jpeg"}

application_image_transformed_total{instance="imageorchestrator:8080",job="imageorchestrator",type="image/png"}

application_image_transformed_total{instance="imageorchestrator:8080",job="imageorchestrator",type="image/tiff"}

Add Graph

Thought you may ask...ports the services run on:

xx.xx.xx.xx:8080	imageorchestrator
xx.xx.xx.xx:8081	imageholder
xx.xx.xx.xx:8082	imagerotator
xx.xx.xx.xx:8083	imagegrayscale
xx.xx.xx.xx:8084	imageresize
xx.xx.xx.xx:8085	imageflip
xx.xx.xx.xx:8086	imagethumbnail

A first query

- Identify the metric you care about
  - Go to the `status` menu and select `targets` to view the metrics endpoint as a web page
  - Browse available metrics from the drop down next to the `execute` button
  - Use the fuzzy search in the `Expression` input field
- Then get a feel for the “shape” of the metric
  - Just run a query asking for the metric without any filters, time periods, or functions
  - Start to formulate what is important about the metric through some questions:
    - What are the common labels vs those that are very unique?
    - What are the datatypes of the labels?
    - How variant are the values across unique label combinations?

Moving to more powerful queries

- Filter and select by string labels
  - Copy and paste a label in as a filter, add a second filter
  - Filter based on not equals, or a partial match
- Work with the three different types of metrics
  - View the graph of a metric whose value could only increase in value over time
  - View the graph of a metric whose value should fluctuate up and down over time
  - Find a bucketed metric and get the value of the highest two bucket values
- Understand metrics over time
  - Get a metrics value for the last 5 minutes
  - Compare a metric between now and 5 minutes ago (and every 5 minutes for an hour)
- Compare metrics
  - Select only the top 5 results of a query
  - Find values over a certain number
  - Select metrics which are both under a certain count and over a different count

Exploring with metrics...

- ➔ How did that feel?
- ➔ What do you want to ask next?
- ➔ What tool would best answer that question?

Our day today

Goal:

To introduce the power of arbitrarily wide, context rich events and how to leverage them to explore a distributed environment

Takeaways:

- ➔ Understanding "normal" for an application
- ➔ Experience with Kibana data visualisation tool
- ➔ Exposure to the data structures of logs, metrics and traces

- ✅ Welcome
- ✅ Understanding our business domain
- ✅ Architecture and observability intro
- ✅ Observability within kibana
  - ✅ Explore APM
  - ✅ Dive into detailed logs and events
  - ✅ Understand the bigger trend of metrics
- ✅ More experience with logs
- ✅ More experience with metrics (prometheus)
- ❑ Preparing for tomorrow

We will get a bit more down and dirty tomorrow

Generating an ssh key

In the `terminal`

```
$ ssh-keygen -t rsa -b 4096 -C "email@host.com"
```

Defaults will create a file at `/home/<you>/.ssh/id_rsa` without a passphrase

```
$ cat /home/<you>/.ssh/id_rsa.pub
```

Copy this text and send it to us. We will add it to `~/.ssh/authorized_keys`

Day Break

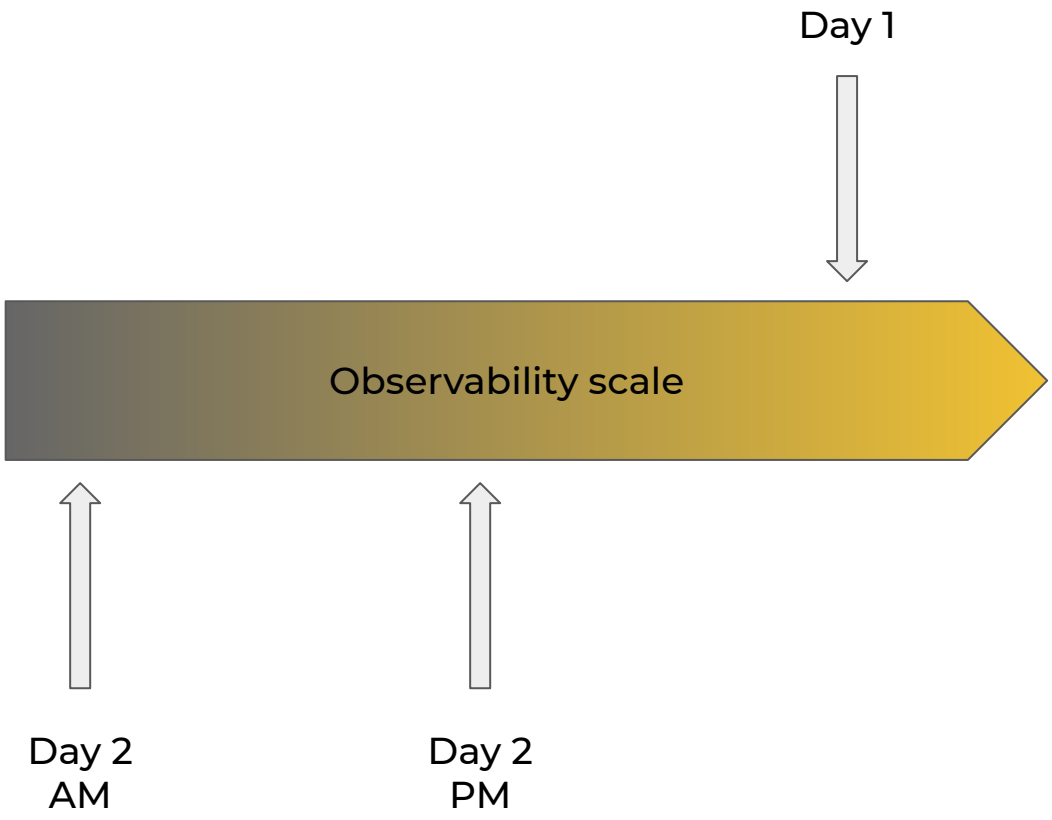
Our two days together

Goal:

To introduce the power of arbitrarily wide, context rich events and how to leverage them to explore a distributed environment

Takeaways:

- ➔ Identify what can impact levels of observability
- ➔ Exposure to tools and techniques that are used when exploring observable systems
- ➔ Understand how the industry has moved towards observability



Our day today

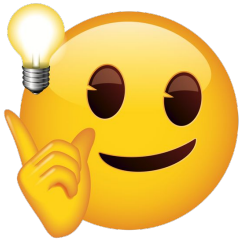
Goal:

To experience the journey of building more observability into a system step by step

Takeaways:

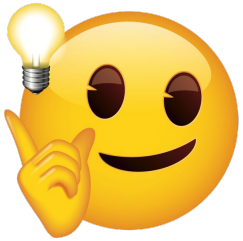
- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

- ❑ Recap
- ❑ Take a journey through log experiences
  - ❑ No aggregation, no structure
  - ❑ Aggregation without structure
  - ❑ Aggregation and structure
  - ❑ Events
- ❑ Take a journey through metrics
  - ❑ Up
  - ❑ Server metrics
  - ❑ Business metrics
- ❑ Take a journey through traces
  - ❑ APM
  - ❑ Contextual tracing
- ❑ Discussion about observability and testing



- Tooling
  - Kibana APM
  - Prometheus
- Power of data in ELK

what do you think you can apply when you get back home next week?



- Tooling
  - Kibana APM
  - Prometheus
- Power of data in ELK

what do you think you can apply when you get back home next week?

What questions have popped up overnight?

Our day today

Goal:

To experience the journey of building more observability into a system step by step

Takeaways:

- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

✅ Recap

- ❑ Take a journey through log experiences
  - ❑ No aggregation, no structure
  - ❑ Aggregation without structure
  - ❑ Aggregation and structure
  - ❑ Events
- ❑ Take a journey through metrics
  - ❑ Up
  - ❑ Server metrics
  - ❑ Business metrics
- ❑ Take a journey through traces
  - ❑ APM
  - ❑ Contextual tracing
- ❑ Discussion about observability and testing

Let’s look at ***how*** to debug stuff…

Logs “on disk”

Finding the right log file

Get onto the computer:

```
$ ssh atd@46.101.170.74
```

Figure out what is running:

Opening a log file for a service:

Logs “on disk”

Finding the right log file

Get onto the computer:

```
$ ssh atd@46.101.170.74
```

Figure out what is running:

```
$ docker ps
```

Opening a log file for a service:

Logs “on disk”

Finding the right log file

Get onto the computer:

```
$ ssh atd@46.101.170.74
```

Figure out what is running:

```
$ docker ps
```

Opening a log file for a service:

```
$ docker logs <service_name>

(e.g. dima_imageflip_1)
```

Logs “on disk”

Finding the right log file

Get onto the computer:

```
$ ssh atd@46.101.170.74
```

Figure out what is running:

```
$ docker ps
```

Opening a log file for a service:

```
$ docker logs <service_name>
```

```
(e.g. dima_imageflip_1)
```

Finding the right log details

Watch the logs as they come in:

Open your logs to search:

Jump to the bottom of the file:

Search for a string:

Next result: *Previous result:*

Count search results:

Logs “on disk”

Finding the right log file

Get onto the computer:

```
$ ssh atd@46.101.170.74
```

Figure out what is running:

```
$ docker ps
```

Opening a log file for a service:

```
$ docker logs <service_name>
```

```
(e.g. dima_imageflip_1)
```

Finding the right log details

Watch the logs as they come in:

```
$ docker logs -f <service_name>
```

Open your logs to search:

Jump to the bottom of the file:

Search for a string:

Next result: *Previous result:*

Count search results:

Logs “on disk”

Finding the right log file

Get onto the computer:

```
$ ssh atd@46.101.170.74
```

Figure out what is running:

```
$ docker ps
```

Opening a log file for a service:

```
$ docker logs <service_name>
```

```
(e.g. dima_imageflip_1)
```

Finding the right log details

Watch the logs as they come in:

```
$ docker logs -f <service_name>
```

Open your logs to search:

```
$ docker logs <service_name> | less
```

Jump to the bottom of the file:

Search for a string:

Next result: *Previous result:*

Count search results:

Logs “on disk”

Finding the right log file

Get onto the computer:

```
$ ssh atd@46.101.170.74
```

Figure out what is running:

```
$ docker ps
```

Opening a log file for a service:

```
$ docker logs <service_name>
```

```
(e.g. dima_imageflip_1)
```

Finding the right log details

Watch the logs as they come in:

```
$ docker logs -f <service_name>
```

Open your logs to search:

```
$ docker logs <service_name> | less
```

Jump to the bottom of the file: `shift + G`

Search for a string:

Next result: *Previous result:*

Count search results:

Logs “on disk”

Finding the right log file

Get onto the computer:

```
$ ssh atd@46.101.170.74
```

Figure out what is running:

```
$ docker ps
```

Opening a log file for a service:

```
$ docker logs <service_name>
```

```
(e.g. dima_imageflip_1)
```

Finding the right log details

Watch the logs as they come in:

```
$ docker logs -f <service_name>
```

Open your logs to search:

```
$ docker logs <service_name> | less
```

Jump to the bottom of the file: `shift + G`

Search for a string: `/<search_term>`

Next result: *Previous result:*

Count search results:

Logs “on disk”

Finding the right log file

Get onto the computer:

```
$ ssh atd@46.101.170.74
```

Figure out what is running:

```
$ docker ps
```

Opening a log file for a service:

```
$ docker logs <service_name>
```

```
(e.g. dima_imageflip_1)
```

Finding the right log details

Watch the logs as they come in:

```
$ docker logs -f <service_name>
```

Open your logs to search:

```
$ docker logs <service_name> | less
```

Jump to the bottom of the file: `shift + G`

Search for a string: `/<search_term>`

Next result: `n` *Previous result:* `N`

Count search results:

Logs “on disk”

Finding the right log file

Get onto the computer:

```
$ ssh atd@46.101.170.74
```

Figure out what is running:

```
$ docker ps
```

Opening a log file for a service:

```
$ docker logs <service_name>
```

(e.g. dima_imageflip_1)

Finding the right log details

Watch the logs as they come in:

```
$ docker logs -f <service_name>
```

Open your logs to search:

```
$ docker logs <service_name> | less
```

Jump to the bottom of the file: `shift + G`

Search for a string: `/<search_term>`

Next result: `n` *Previous result:* `N`

Count search results:

```
$ docker logs <service_name |
  grep -o '<search_term>' | wc -l
```

Our day today

Goal:

To experience the journey of building more observability into a system step by step

Takeaways:

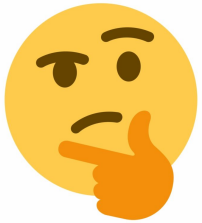
- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

- ✅ Recap
- ❑ Take a journey through log experiences
  - ✅ No aggregation, no structure
  - ❑ Aggregation without structure
  - ❑ Aggregation and structure
  - ❑ Events
- ❑ Take a journey through metrics
  - ❑ Up
  - ❑ Server metrics
  - ❑ Business metrics
- ❑ Take a journey through traces
  - ❑ APM
  - ❑ Contextual tracing
- ❑ Discussion about observability and testing

Whew, we have Kibana back!

➔ Look at some of the tools you used yesterday…

- How does include/exclude work now?  
- How valuable are columns now? add 



How would we search for logs of a certain image type now?

Our day today

Goal:

To experience the journey of building more observability into a system step by step

Takeaways:

- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

- ✅ Recap
- ❑ Take a journey through log experiences
  - ✅ No aggregation, no structure
  - ✅ Aggregation without structure
  - ❑ Aggregation and structure
  - ❑ Events
- ❑ Take a journey through metrics
  - ❑ Up
  - ❑ Server metrics
  - ❑ Business metrics
- ❑ Take a journey through traces
  - ❑ APM
  - ❑ Contextual tracing
- ❑ Discussion about observability and testing

Coffee break

Our day today

Goal:

To experience the journey of building more observability into a system step by step

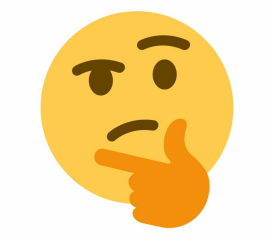
Takeaways:

- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

- ✅ Recap
- ❑ Take a journey through log experiences
  - ✅ No aggregation, no structure
  - ✅ Aggregation without structure
  - ❑ Aggregation and structure
  - ❑ Events
- ❑ Take a journey through metrics
  - ❑ Up
  - ❑ Server metrics
  - ❑ Business metrics
- ❑ Take a journey through traces
  - ❑ APM
  - ❑ Contextual tracing
- ❑ Discussion about observability and testing

Structures, how much does this help?

➔ How would we search for logs of a certain image type now?



How many times did we persist a flipped image?

Our day today

Goal:

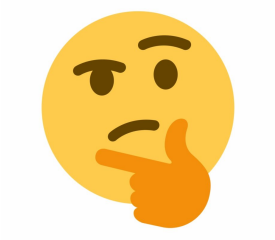
To experience the journey of building more observability into a system step by step

Takeaways:

- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

- ✅ Recap
- ❑ Take a journey through log experiences
  - ✅ No aggregation, no structure
  - ✅ Aggregation without structure
  - ✅ Aggregation and structure
  - ❑ Events
- ❑ Take a journey through metrics
  - ❑ Up
  - ❑ Server metrics
  - ❑ Business metrics
- ❑ Take a journey through traces
  - ❑ APM
  - ❑ Contextual tracing
- ❑ Discussion about observability and testing

Events...a world beyond logs



How many times did we persist a flipped image?

So we know ***how*** to debug stuff…

… how do we know ***when*** to start debugging??

Our day today

Goal:

To experience the journey of building more observability into a system step by step

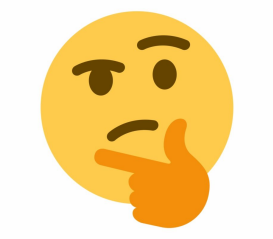
Takeaways:

- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

- ✅ Recap
- ✅ Take a journey through log experiences
  - ✅ No aggregation, no structure
  - ✅ Aggregation without structure
  - ✅ Aggregation and structure
  - ✅ Events
- ❑ Take a journey through metrics
  - ❑ Up
  - ❑ Server metrics
  - ❑ Business metrics
- ❑ Take a journey through traces
  - ❑ APM
  - ❑ Contextual tracing
- ❑ Discussion about observability and testing

What does “up” really mean?

➔ Review what is currently “up” to make sure we are in a healthy state



Well then, why is “Manipulate” acting funny?

Our day today

Goal:

To experience the journey of building more observability into a system step by step

Takeaways:

- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

- ✅ Recap
- ✅ Take a journey through log experiences
  - ✅ No aggregation, no structure
  - ✅ Aggregation without structure
  - ✅ Aggregation and structure
  - ✅ Events
- ❑ Take a journey through metrics
  - ✅ Up
  - ❑ Server metrics
  - ❑ Business metrics
- ❑ Take a journey through traces
  - ❑ APM
  - ❑ Contextual tracing
- ❑ Discussion about observability and testing

Lunch time!

Our day today

Goal:

To experience the journey of building more observability into a system step by step

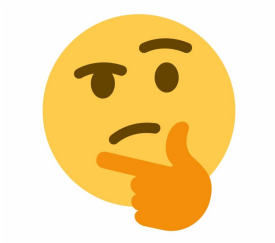
Takeaways:

- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

- ✅ Recap
- ✅ Take a journey through log experiences
  - ✅ No aggregation, no structure
  - ✅ Aggregation without structure
  - ✅ Aggregation and structure
  - ✅ Events
- ❑ Take a journey through metrics
  - ✅ Up
  - ❑ Server metrics
  - ❑ Business metrics
- ❑ Take a journey through traces
  - ❑ APM
  - ❑ Contextual tracing
- ❑ Discussion about observability and testing

“Technical” monitoring to the rescue!

➔ Review prometheus (`46.101.170.74:9090`) for newly available data to help us be alerted to that weird grayscale behaviour?



While this is great, what more could we want?

Our day today

Goal:

To experience the journey of building more observability into a system step by step

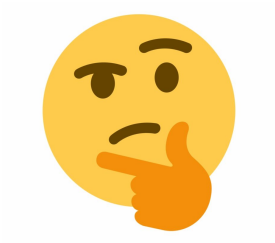
Takeaways:

- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

- ✅ Recap
- ✅ Take a journey through log experiences
  - ✅ No aggregation, no structure
  - ✅ Aggregation without structure
  - ✅ Aggregation and structure
  - ✅ Events
- ❑ Take a journey through metrics
  - ✅ Up
  - ✅ Server metrics
  - ❑ Business metrics
- ❑ Take a journey through traces
  - ❑ APM
  - ❑ Contextual tracing
- ❑ Discussion about observability and testing

Context...we always want more context!

➔ Now we have custom metrics, we can be more business context aware



What is a question we can answer with the `application_` metrics that we can not without them?

Now we know ***how*** and ***when*** to debug…

...But ***where*** should we start??

Our day today

Goal:

To experience the journey of building more observability into a system step by step

Takeaways:

- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

- ✅ Recap
- ✅ Take a journey through log experiences
  - ✅ No aggregation, no structure
  - ✅ Aggregation without structure
  - ✅ Aggregation and structure
  - ✅ Events
- ✅ Take a journey through metrics
  - ✅ Up
  - ✅ Server metrics
  - ✅ Business metrics
- ❑ Take a journey through traces
  - ❑ Basic tracing
  - ❑ Contextual tracing
- ❑ Discussion about observability and testing

The developers got antsy and gave themselves tracing

ZIPKIN

- Open zipkin by going to: `46.101.170.74:9411` and give it an explore how long a request takes in each service

Find your specific image manipulation

Our day today

Goal:

To experience the journey of building more observability into a system step by step

Takeaways:

- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

- ✅ Recap
- ✅ Take a journey through log experiences
  - ✅ No aggregation, no structure
  - ✅ Aggregation without structure
  - ✅ Aggregation and structure
  - ✅ Events
- ✅ Take a journey through metrics
  - ✅ Up
  - ✅ Server metrics
  - ✅ Business metrics
- ❑ Take a journey through traces
  - ✅ Basic tracing
  - ❑ Contextual tracing
- ❑ Discussion about observability and testing

And eventually,
they wanted the benefits of their logs WITH their tracing

ZIPKIN

➔ Retry...can you find your specific image manipulation

We still value logs, they are helpful, how can I see the logs from this trace?

Our day today

Goal:

To experience the journey of building more observability into a system step by step

Takeaways:

- ➔ Observability is a scale
- ➔ Investment in observability can be expensive, but also impactful
- ➔ We can find value in our system outputs even at low levels of observability

- ✅ Recap
- ✅ Take a journey through log experiences
  - ✅ No aggregation, no structure
  - ✅ Aggregation without structure
  - ✅ Aggregation and structure
  - ✅ Events
- ✅ Take a journey through metrics
  - ✅ Up
  - ✅ Server metrics
  - ✅ Business metrics
- ❑ Take a journey through traces
  - ✅ Basic tracing
  - ✅ Contextual tracing
  - ❑ Tracing as a part of our other data
- ❑ Discussion about observability and testing

Nirvana state realised… *(sort of)*

ZIPKIN

➔ Join us back in Kibana and explore the logs from the apm traces

What more could we ask for?

(a lot...so seriously...what more?)

Let’s chat

Thanks for your attention!

Feedback welcome!

Go to **agiletestingdays.com/session-ratings**
and give your rating!