Instrumentalização de logs

Este projeto é uma poc, que tenta demonstrar e explicar pontos importantes, para instrumentalizar logs.

Vamos lá.

Defina o que se quer obter dos logs

Primeiro de tudo, é necessário definirmos o que desejamos conseguir extrair dos logs.

Como:

- Descrobir qual processo falhou;
- Quanto tempo um determinado processo durou;
- Por quais processos ou aplicações o dado passou (Onde, quando e por que);
- Quais e quantos dados não alcançaram o final do processo;
- Se o dado faz parte de um grupo específico, tais como uma campanha, se pertence à uma conta ou um usuário específico.

Essa parte é importante, pois é a partir dela, que é possível definir quando, onde e como gerar o log, evitando gerar uma quantidade massiva de dados desnecessários ou então gerando dados que não dão visibilidade nenhuma do que queremos.

Faça os logs acessíveis, padronize

Padrões são importantes para facilitar a execução e análise do que foi feito.

Escolha um padrão pra data e hora

É sempre importante ter o tempo de forma concisa nas informações que desejamos analisar, para isso é importante ter um formato definido, tal como:

- O padrão de escrita (ISO, unix, etc)
- O fuso horário
    - Seja a informação do fuso horário como UTC + offset ou em casos menos críticos apenas um fuso horário definido

Alguns servidores de logs possuem a funcionalidade de registrar o horário de chegada do log, enquanto outras vezes, é necessário fazer esse controle a partir do log ou da ferramenta que coleta o mesmo (o que pode ser útil para casos com servidores e apps em fuso horários diferentes).

Defina Níveis para os logs e os configure adequadamente

Níveis servem para identificar a criticidade do log. E podem ser informação, um erro, um aviso, uma falha crítica, debug, etc, representados às vezes por números.

Hoje já existem alguns padrões de Níveis de alguns sistemas, tais como Syslog.

Dê contexto aos logs quando possível

É importante sempre que possível se ter em mente a importância do contexto, sempre sabendo do que o processo/dado se trata e à quem pertence ou faz parte.

E.g.

Suponhamos que temos um sistema que é capaz de:

- receber um vídeo
- gerar qualidades diferentes do vídeo
- separar o áudio
- gerar legendas
- disponibilizar o vídeo.

Porém, isso se trata apenas de algumas, das inúmeras funcionalidades que o sistema possui.

Um dia, surge a informação de que alguns vídeos não são encontrados após o upload e, nenhuma informação a mais é passada. Agora precisamos tentar olhar os logs de todos os precessos para encontrar alguma anormalidade.

O quão simples isso se tornaria se, os logs tivessem uma chave informando qual log é de um processo relacionado a vídeos (algo como uma label)?

E não só nesse caso mas em processos mais isolados é sempre importante ser capaz de compreender sobre o que o processo se trata.

Enforce um padrão

Para evitar nos perdermos entre formatos diferentes, ferramentas diferentes e dados diferentes com a mesma finalidade, devemos escolher uma ferramenta e apresentar regras de uso.

- Que ferramenta iremos utilizar para logar?
  - Uma lib, a própria ferramenta da linguagem? Simplesmente o stdout?
- Que formato iremos escrever as chaves dos logs?
  - camelCase? snake_case? (Lembrando que sempre que possível cabe evitar usar chaves compostas)
- Que chaves serão usadas para representar determinados dados?
  - Que chave será usada nomear o contexto por exemplo (podendo ter indicações de contexto em níveis diferentes, tais como app, processo, etc, mas evitando coisas como app, application, service que representam o mesmo nível)

- Quais serão os momentos em que iremos logar?
  - Geralmente definido pela meta estabelecida

Cuidados a se tomar ao logar

- Evitar informações sensíveis;
- Seguir os padrões definidos:
  - CamelCase ou snake_case;
  - Utilizar uma lib universal para tal;
  - Etc...;
- Garantir que a infraestrutura possui capacidade para coleta e armazenagem dos logs;
- Evitar informação em excesso, saber/aprender o que acontece e o que queremos;
- Evitar informações vagas, É bom saber que algo aconteceu, mas também o que é esse algo;
  - Logs should contain the “why” while avoiding unnecessary noise.
- VALIDAR OS LOGS. Sempre que possível testar localmente se os logs que foram posicionados satisfazem o desejado e também se não são apenas uma poluição.

Exemplo

Para este exemplo, iremos propor uma situação onde desejamos rastrear o que acontece com o dado do começo ao fim.

O sistema, é responsável por:

1. Receber um número de telefone e uma mensagem de texto | returnar id da requisição;
2. Validar se número está bloqueado;
3. Validar se número existe;
4. Normalizar mensagem;
5. Capitalizar primeira letra da mensagem 50% das vezes ou capitalizar todas as letras;
6. Salvar para um csv.

Possui as seguintes aplicações:

> * significa que a responsabilidade é de chamar o processo que executa de fato a atividade

aplicações	responsabilidades
api	1,2,*3*
blacklist	1
telephone directory	2
process	4,5

aplicações	responsabilidades
csvWriter	6

Definindo o que queremos obter dos logs

Definiremos as seguintes necessidades que desejamos extrair dos logs.

1. Qual caminho o dado percorreu? Quais foram os processos pelo qual ele passou.
2. Caso ele não tenha chegado ao processo final, por que não o fez?
3. ~~Caso faça de um grupo de dados, tal como uma campanha, qual seria este grupo?~~
    - Removido, porém caso queira rastrear grupos de dados basta passar a propriedade agrupadora nos logs

Padronizando

Padrões e mais padrões.

Escolhendo Ferramentas

Agora que temos o que queremos, vejamos com o que vamos alcançar o resultado.

Escolhendo ferramenta para coleta e análise

Primeiro vamos escolher as ferramentas. Hoje em dia existem ferramentas que servem pra coletar e analisar logs, tais como fluentd (coleta), logstash (coleta), elasticsearch (armazenar), kibana (visualizar e analizar) e newrelic (visualizar e analisar), entre muitas outras ferramentas.

Para a coleta de logs no caso atual, iremos utilizar o fluentd e, para armazenar e analisar o newrelic.

Escolhendo ferramenta para geração de logs

Agora vamos escolher como vamos gerar os logs.

Felizmente no mundo em que vivemos existem inúmeras ferramentas de logs para várias linguagens, entre elas, no JavaScript temos, por exemplo, winston, log4js, signale, pino, etc.

Cada uma tem sua vantagem, seja por adaptalidade, velocidade ou simplicidade. E para o exemplo, vamos escolher o pino.

Benefícios `pino`:

- É simples então não há muito a se pensar
- A criação do timestamp vem em uma chave diferente da utilizada pelo servidor de logs
    - gosto pq me ajudaria a comparar se necessário, mas também pode trazer confusão
    - isso é configurável, veja aqui funções de tempo padrão e aqui como alterar para outras

- Possui um plugin chamado `@newrelic/pino-enricher` assim como também para o winston, que nada mais é do que um plugin que ajuda a manter um padrão, o newrelic em si mantém esses plugins para várias linguagens.

Embora não vamos usar o plugin aqui neste exemplo, seria uma boa ideia testar e usar em casos reais

Escolhendo o que logar

Após decidido o que desejamos alcançar, podemos definir algumas regras para decidir onde, quando e como logar.

Regras tais como:

- Sempre criar uma instância do logger quando em um escopo com informações do escopo
  - O id da request (requisição inicial) se disponível no formato `request.id`
  - O id da mensagem se disponível no formato `message.id`
  - A ação do escopo, para o indentificar, no caso definida de uma forma simples
    - Para funções `function.{FUNCTION_NAME}`
    - Para handlers http `http.{METHOD}:/{ROUTE_PATH}`
  - Informar aplicação anterior na propriedade `origination` quando disponível
- No primeiro processo pegar o id da requisição ou definir um
- Quando chamar outra aplicação informar de qual aplicação está vindo a requisição com o header `X-Origin-Application`
- Registrar erro, sucesso e finalização (quando não há um erro ou algo explicitamente que indique um sucesso, tais como subprocessos)
- Usar mensagens simples e diretas para indicar o estado da ação, podendo ter mais detalhes quando a necessidade se fizer
  - E.g. `start {ACTION}`, `end {ACTION}`, `fail {ACTION}`

Preparando o ambiente

Vamos executar o seguinte:

- Instalar o docker e docker-composer caso não tenhamos;
- Criar uma conta no newrelic caso não tenhamos;
- Copiar o .env.sample para o .env;
- Preencher o valor da variável `NEW_RELIC_LICENSE_KEY` no .env;
- Executar `docker-compose -f samples/fluentd/docker-compose.yaml up --build -d`;
- Verificar a inicialização com `docker-compose -f samples/fluentd/docker-compose.yaml`;
- Testar que está funcionado:
  - Executar algumas vezes `echo "hello" >> samples/logs/default.log` e ver na aba de logs do newrelic se está chegando (lembre de clicar em query logs, pode demorar uns segundos para computar).

> Não iremos entrar em detalhes, mas caso tenha, tente pesquisar um pouco sobre o que gostaria de ver mais a fundo.

O exemplo

Agora que nós temos o ambiente, podemos dar uma olhada por exemplo nas aplicações localizadas em:

> samples/projectsBefore/

Onde iremos encontrar as aplicações antes de adicionar os logs e comparar com:

> samples/projectsAfter/

Onde vamos encontrar as aplicações depois de adicionar os logs.

Para não nos extendermos muito e ter que apontar cada detalhe, vamos tentar passar apenas alguns dos pontos:

- Em cada app, iniciamos uma instância principal de logger, com ao menos o nome da app nos metadados;
  - Como pode ver em `api.js` nas primeiras linhas;
- Na primeira app utilizamos um header, para indicar o id da request e, passar para frente nos processos, assim como o id da mensagem que, é o que havia interesse em rastrear;
- Evitamos logar informações que podem ser sensíveis, como número de telefone ou a mensagem;
- Nos erros sempre foi passada a instância do erro e, depois uma mensagem;
  - Isso foi feito em parte por conta do funcionamento do `pino` que "trata" o erro caso esteja no primeiro campo do método.
- Usamos o header `X-Origin-Application` , para passar pra frente e, indicar nas apps quem requisitou;
  - Isso serve para caso múltiplos processos, sejam chamados por um processo ou, múltiplos processos chamem um processo e se queira identificar.
- Usamos debug, para informações que poderiamos não querer na maior parte do tempo, em que app estiver executando e, não necessariamente importavam para atingir o objetivo de rastreabilidade;
- Não passamos o id da requisição, para algumas partes no inicio do processo;
  - Isso pode ser desejado em alguns momentos, mas neste caso foi julgado que como a resposta seria "imediata" seria possível saber o que ocorre.

Que tal um teste? Vamos executar os ambientes e comparar.

O antes

Primeiro executamos `docker-compose -f samples/projectsBefore/docker-compose.yaml up --build -d` e, tentamos procurar no newrelic os logs (para facilitar tente adicionar no filtro

`separator:bpd`).

Bem, por enquanto, tudo que veremos, serão alguns logs indicando que as apps iniciaram.

Vamos executar alguns testes:

Requisição normal

```
curl --request POST \
  --url http://localhost:3141/v1/txt \
  --header 'content-type: application/json' \
  --header 'x-request-id: theidd' \
  --data '{
    "phone": "+5511912045674",
    "text": "Some text. Artoria, verto"
}'
```

Requisição com um erro

```
curl --request POST \
  --url http://localhost:3141/v1/txt \
  --header 'content-type: application/json' \
  --header 'x-request-id: theidd' \
  --data '{
    "phone": "+5511912045675",
    "text": "Lerto Some toria, verto"
}'
```

Requisição com um erro

```
curl --request POST \
  --url http://localhost:3141/v1/txt \
  --header 'content-type: application/json' \
  --header 'x-request-id: theidd' \
  --data '{
    "phone": "+5511912045675",
    "text": "Pseu dteu tori no met dta"
}'
```

E agora, como podemos identificar no newrelic, não há muita informação para entender o que ocorre (se é que há).

O depois

Agora vamos ver como os logs ficaram.

Primeiro derrubamos os processos anteriores:

> docker-compose -f samples/projectsBefore/docker-compose.yaml down -v

E agora subimos os novos:

> docker-compose -f samples/projectsAfter/docker-compose.yaml up --build -d

Executamos as mesmas requisições usadas anteriormente para gerar os logs.

E agora tentamos comparar, para facilitar podemos clicar no botão de adicionar colunas, com um símbolo de `+` e, adicionar as seguintes colunas `timestamp, time, application, message.id, request.id, origination, message` .

Eis Algumas coisas que podemos perceber:

- Alguns logs estão fora de ordem mesmo sendo da mesma mensagem. Isso ocorre quando um processo começa, antes do processo que o chamou terminar;
- Como dito anteriormente, alguns processos não indicam nenhum identificador;

Podemos também brincar um pouco, tentar parar algumas apps, editar o código para forçar alguns erros e ver como se comporta.

Como já se deve imaginar, este exemplo não é perfeito e nem mesmo definitivo, não atende a todas as necessidades, então não tenha medo de conversar com seu time e procurar se adaptar para atender as necessidades.

Algumas notas

- O `time` pode diferenciar de `timestamp` , pois enquanto o `time` foi gerado pelo `client` (`pino`), o `timestamp` foi gerado pelo servidor do newrelic
- Evite iniciar como no exemplo com o logger com níveis de log `trace` habilitado, é recomendado usar de `info` pra cima no dia a dia da app.
- É possível alterar o level do logger principal sem reiniciar a aplicação, basta adicionar na app uma funcionalide que edite logger.level
- O header que possui o id da requisição pode ser gerado por um reverse proxy (tente conversar com algum devop ou alguém de infra), porém pode ser algo gerado na própria aplicação sempre.

References

- https://www.dnsstuff.com/logging-monitoring-best-practices
- https://newrelic.com/resources/white-papers/log-management-best-practices
- https://docs.newrelic.com/docs/logs/ui-data/parsing/
- https://betterprogramming.pub/application-logging-best-practices-a-support-engineers-perspective-b17d0ef1c5df?gi=8e4024c7662f

- https://www.taraspan.com/blog/a-guide-to-log-management/
- https://dev.splunk.com/enterprise/docs/developapps/addsupport/logging/loggingbestpractices/
- https://docs.newrelic.com/docs/logs/logs-context/configure-logs-context-php/
- https://sodocumentation.net/http-headers/topic/10581/x-request-id