OBSERVABILITY AND THE DEV PROCESS

@cyen
@honeycombio
Hi.
OPS

"THE ONLY GOOD DIFF IS A RED DIFF"

DEV

"WORKS ON MY MACHINE"

"NINES DON’T MATTER IF USERS AREN’T HAPPY"
OPS

🤝

DEV

OBSERVABILITY
OPS
- e2e Checks
- Alert Thresholds
- Resource Allocation
- Networking
- CPU Utilization
- Hosts + Instance Types
  
DEV
- Builds (/ Build IDs)
- Customer IDs
- Endpoints
- Other repro-able characteristics
"Huh, CPU Utilization is increasing on that cluster. Time to increase capacity!"
"Request volume is increasing on that cluster... but it looks like it’s mostly one customer."
- Design documents
- Architecture review
- Test-driven development
- Integration tests
- Code review
- Continuous integration
- Continuous deployment

🎉🥂🍾🎊

(Wait for exception tracker to complain)
Not all interesting things are problems
Not all interesting things are known ahead of time
... or express themselves as anomalies
Not all problems manifest as exceptions
How to build those features / fix those bugs
How features and fixes are scoped
How to verify correctness or completion
How to roll out that feature or fix
How’s our load? Is it spread reasonably evenly across our Kafka partitions?

Did latency increase in our API server? Is our new /batch endpoint performing well?

How did those recent memory optimizations affect our query-serving capacity?
How’s our load? Are **high-volume customers** spread reasonably evenly across our Kafka partitions?

Did latency increase in our API server? **Which customers benefit most** from our new /batch endpoint?

How did those recent memory optimizations affect our query-serving capacity for **customers with string-heavy payloads**?
Tests aren’t enough

Benchmarks aren’t enough

Exceptions aren’t enough
Tests aren’t enough
Benchmarks aren’t enough
Exceptions aren’t enough
“WORKS ON MY MACHINE”
“WHERE (WHEN) DID IT COME FROM?”
ERROR VOLUME

ERROR VOLUME, BY BUILD
- Design documents
- Architecture review
- Test-driven development
- Integration tests
- Code review
- Continuous integration
- Continuous deployment
- Test in production (with feature flags)
- Identify outliers in dev terms, not ops terms
- Explore prod in realtime
- (Wait for exception tracker to complain)
OPS — DEV

- Form hypotheses about what their code will do in production
- Add/tweak instrumentation as necessary
- Query data to (in)validate hypotheses
- Take action (and repeat as necessary)
TAKING THE FIRST FEW STEPS

- Start at the edge with basic, common attributes (e.g. HTTP)
TAKING THE FIRST FEW STEPS

- Start at the edge with basic, common attributes (e.g. HTTP)
- Business-relevant or infrastructure-specific characteristics (e.g. customer ID, DB replica set)
TAKING THE FIRST FEW STEPS

- Start at the edge with basic, common attributes (e.g. HTTP)
- Business-relevant or infrastructure-specific characteristics (e.g. customer ID, DB replica set)
- Temporary additional fields for validating hypotheses
TAKING THE FIRST FEW STEPS

▸ Start at the edge with basic, common attributes (e.g. HTTP)

▸ Business-relevant or infrastructure-specific characteristics (e.g. customer ID, DB replica set)

▸ Temporary additional fields for validating hypotheses

▸ Prune stale fields (if necessary)
SOME BEST PRACTICES

- Contextual, structured data
SOME BEST PRACTICES

- Contextual, structured data
- Common set of nouns and consistent naming
SOME BEST PRACTICES

▸ Contextual, structured data
▸ Common set of nouns and consistent naming
▸ Don't be dogmatic; let the use case dictate the ingest pattern
SOME BEST PRACTICES

- Contextual, structured data
- Common set of nouns and consistent naming
- Don't be dogmatic; let the use case dictate the ingest pattern
  - e.g. instrumenting individual reads while batching writes
AN EXAMPLE SCHEMA EVOLUTION

first pass:

- server_hostname
- method
- url
- build_id
- remote_addr
- request_id
- status
- x_forwarded_for
- error
- event_time
- team_id
- payload_size
- sample_rate

then we added:

- dropped
- get_schema_dur_ms
- protobuf_encoding_dur_ms
- kafka_write_dur_ms
- request_dur_ms
- json_decoding_dur_ms +others

a couple of days later, we added:

- offset
- kafka_topic
- chosen_partition
AN EXAMPLE SCHEMA EVOLUTION

first pass:
- server_hostname
- method
- url
- build_id
- remote_addr
- request_id
- status
- x_forwarded_for
- error
- event_time
- team_id
- payload_size
- sample_rate

then we added:
- dropped
- get_schema_dur_ms
- protobuf_encoding_dur_ms
- kafka_write_dur_ms
- request_dur_ms
- json_decoding_dur_ms +others

a couple of days later, we added:
- offset
- kafka_topic
- chosen_partition

after that:
- memory_inuse
- num_goroutines

a week after that:
- warning
- drop_reason
- x_forwarded_for
- error
- event_time
- team_id
- payload_size
- sample_rate

and on and on, adding 2-3 fields every couple of weeks:
- user_agent
- unknown_columns
- dataset_partitions
- dataset_id
- dataset_name
- api_version
- create_marker_dur_ms
- offset
- kafka_topic
- nil_value_for_columns
- batch
- gzipped
- batch_datapoint_lens
- batch_num_datasets

a couple of days later, we added:

and on and on, adding 2-3 fields every couple of weeks:
- user_agent
- unknown_columns
- dataset_partitions
- dataset_id
- dataset_name
- api_version
- create_marker_dur_ms
- offset
- kafka_topic
- nil_value_for_columns
- batch
- gzipped
- batch_datapoint_lens
- batch_num_datasets
DEV'S, OUR MISSION:

- Stop writing software based on intuition, start backing it up with data
- Teach observability tools to speak more than "Ops"
- ??? (← ask lots of questions and validate hypotheses)
- Profit!
THANKS! 🤗

more stories:
https://honeycomb.io/blog/categories/dogfooding

icons:
https://thenounproject.com/daniele.catalanotto