Key performance indicators

Breno Gomes
Senior Solution Engineer

January 2022

Why it is important

Modern software has a large volume of interconnected components. In an ideal world the number and types of failures grow proportionally. Strong adoption of cloud functions and containers increases complexity.

Distributed systems are updated regularly. Every change can create a new type of failure, instability or more “unknown unknowns”.

Key performance indicators are more than numbers you report during the software development lifecycle.

Metrics provide targets for teams to set goals, milestones to gauge progress and insights that help people across the organization make informed decisions.

A good KPI, by definition, should be measurable and trackable.

KPI in the context of feature flags and experimentation

Existing and prospective customers normally go through a proof of value (PoV) before adopting or subscribing new products.

It is important to define a pilot project to evaluate the feasibility of an implementation before it is widely incorporated in the software development lifecycle. It defines the processes to simplify a “minimum viable product”. A proof of value targets the financial, technical and organisational benefits of such implementation.

Key performance indicators are fundamental to define the success criteria. They are used to compare metrics before, during and after the PoV.

Engineers are often caught off guard when they are asked to share the KPI for a PoV. That is perfectly natural unless they were briefed beforehand. This guide intends to share metrics frequently adopted during product evaluation. It is neither prescriptive nor exhaustive. Each company is unique, having freedom to choose what suits it best.

Use cases

Synchronously toggle features across platforms.

Moving from an old service to a new service or cloud using LaunchDarkly to incrementally accept traffic.

Migrating between databases by reading/writing to different datastores without multiple deployments.

Control the rollout of expensive operations during application updates.

Switching between different UI themes for given sets of client users, allowing controlled testing before release to the store.

Safely test features in production releases to gather feedback.

Flag application elements to test click-through rate.

DevOps

Builds

Commits

Deployment frequency

Approving a feature release

Lead time for changes

Change volume

Change failure rate

Defect escape rate

Mean time to detect

Mean time to identify

Mean time to restore service

Percentage of code covered by automated tests

Application usage and traffic

Application availability

Support tickets

Business

Marketing conversion funnel

Cart abandonment

Cart optimisation

Revenue per customer

Customer retention rate

Profit margin

A/B tests

Unique users

Number of sessions

Page views

Session duration

Geography

Technical

Perceived page load time or mobile interaction

Error rate: frontend, backend and infrastructure

Mobile crash rate

DOM processing

Page rendering

Web browser

Mobile device

API timeout

API error

API latency

Web and Infrastructure

Availability

Throughput

Application response time

Database execution time

Error rate

Memory footprint

CPU utilisation

Network throughput

DORA

Aspect of Software Delivery Performance*	Elite	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per day and once per week	Between once per week and once per month	Between once per month and once every six months
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one day	Between one day and one week	Between one week and one month	Between one month and six months
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day[a]	Less than one day[a]	Between one week and one month
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0-15%[b,c]	0-15%[b,d]	0-15%[c,d]	46-60%

Source: https://cloud.google.com/blog/products/devops-sre/using-the-four-keys-to-measure-your-devops-performance

Database migration

Open source and NoSQL are the primary choices when companies migrate from proprietary to another database system. The main benefits are:

- Cost reduction - open source offers free and commercial licences.
- Flexibility - on-premise, cloud and hybrid.
- Customizability - open source is a thriving model for extensions and add-ons.

Proprietary systems implement extended features that are not part of standard SQL such as views, stored procedures, indexes and triggers. Primary key, foreign key, atomicity features and indexing schemes such as bitmap, reverse key and function-based are neither widely available nor easy port port.

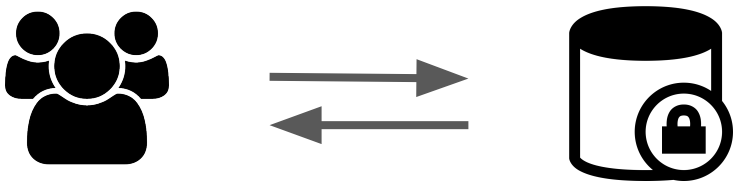
It is quick and simple to migrate code when there are only minor syntax differences or when equivalent functions exist in both databases. However, rewriting procedural logic requires extensive re-engineering.

The general considerations apply to most database systems.

Database migration

Assess

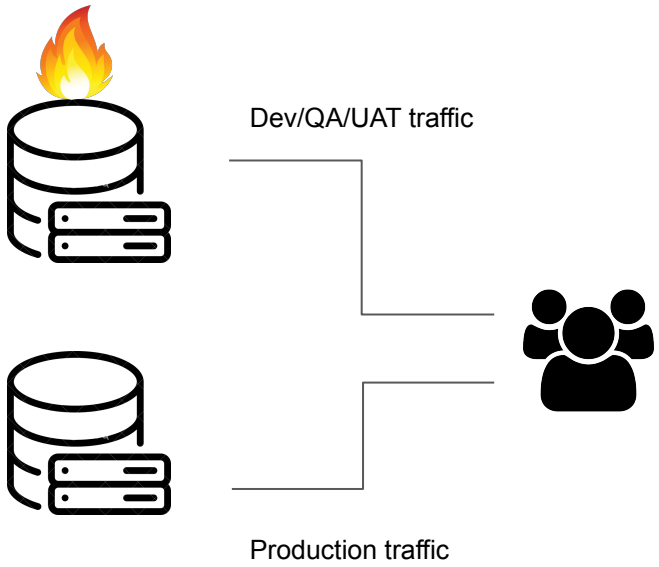
- Database schemas
- Relational and NoSQL
- What can be migrated directly
- What needs to be refactored
- Incompatible, unpractical or uneconomical
- Current performance levels



Database migration

Prototype

- database schemas
- validate migration
- limited rollout using feature flags
- measure against the baselines
- incremental rollout using feature flags
- feedback loop



Metrics

- percent CPU
- time used by the database process
- disk queue for waiting I/O
- throughput
- latency and number of errors

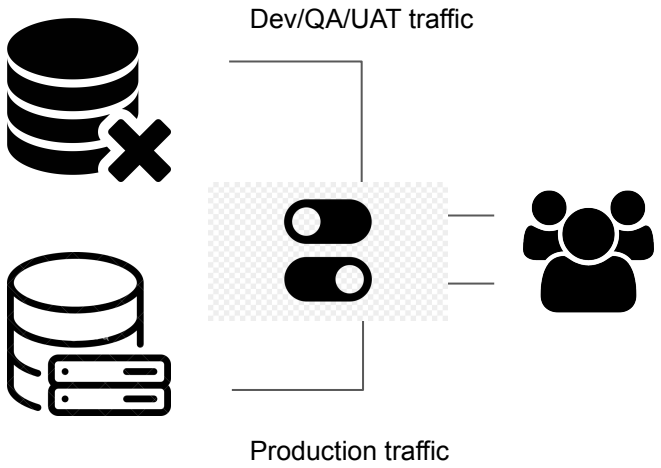
Database migration

Incremental rollout

- decreased risk
- cost optimisation
- compare against the baselines
- performance tuning
- feedback loop

Metrics

- percent CPU
- time used by the database process
- disk queue for waiting I/O
- throughput
- latency and number of errors



Database performance tuning

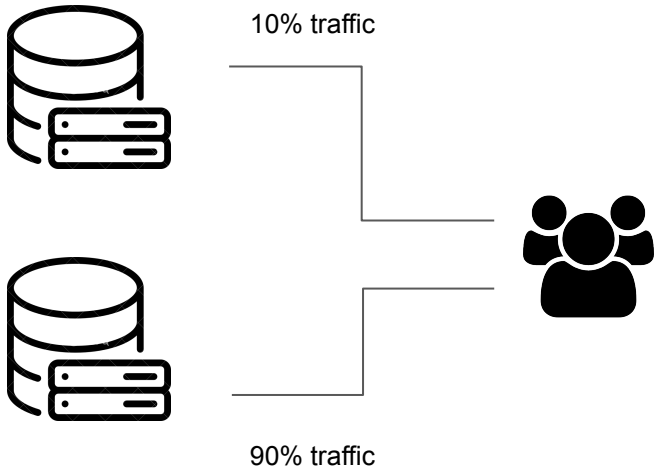
Feature flags are ideal to de-risk query optimisation.

```
SELECT
        invoice_amount
FROM
        order
WHERE
        TO_DAYS(CURRENT_DATE) - TO_DAYS(invoice_date) >= 30;
```

This query will not use any index because of the TO_DAYS() function.

```
SELECT
        invoice_amount
FROM
        order
WHERE
        invoice_date >= DATE_SUB(CURRENT_DATE, INTERVAL 30 DAY);
```

This query will use an index but will not use database caching schemes.
Replacing CURRENT_DATE with a literal value will use index and cache.



Multi variation flags add flexibility to test indexing strategies, targeting a reduced amount of users.

Optimising user experience

Web Vitals is an initiative by Google to provide unified guidance for quality signals that are essential to delivering a great user experience on the web.
Source: https://web.dev/vitals/

Image source: https://web.dev/vitals/

The performance of a site can vary dramatically based on a user's device capabilities, their network conditions, what other processes may be running on the device, and how they're interacting with the page.
Source: https://web.dev/vitals/

A/B tests and feature flags give teams more control over the user experience and de-risk optimisation.