THE ZEN OF PROMETHEUS

Simple, Poetic, Illuminating.

The Zen of Prometheus is beginner friendly set of core values and guidelines for instrumenting your applications and writing idiomatic alerts using Prometheus. This is a document that's intended to maintained by the Prometheus community. Feel free to contribute.

Instrument first, ask questions later

During development you will never know what questions you need to ask later. Software needs good instrumentation, it's not optional. Metrics are cheap. Use them generously.

The First and the most important rule, if you have to remember only one thing remember this one. Instrument all the things!

Measure what users care

Do your users care if your database servers are down? Do they care about your CPU saturation? No, they care about what they experience. They care about whether they can access the page that they have requested and their results are fresh. Think in terms of latencies and availability. Let your SLOs guide your instrumentation.

RED, USE and The Four Golden Signals are known frameworks to get you started.

This doesn't necessarily mean that you should ignore causal metrics. Have them, spread them liberally.

> Use causal metrics to answer why something is broken.

Labels are the new hierarchies

THE ZEN OF PROMETHEUS

Simple, Poetic, Illuminating.

labels, one can group and aggregate measurements afterwards. Slice and dice using Labels. Remember Instrument first, ask questions later, provide much context as possible.

However, you have to *use labels with care*. The reasons will explain in subsequent rules.

Avoid missing metrics

Time series that are not present until something happens are difficult to deal with. To avoid this, export 0 (or NaN if 0 would be misleading) for any time series you know may exist in advance. You have to initialize your metrics with the zero values to prevent broken dashboards and misfiring alerts. For more detailed explanation check out Existential issues with metrics. And also remember, labels create timeseries, so same goes for your labels. Your client libraries can't know what labels you would have. Initialize your metrics with the labels that you would have.

Cardinality Matters

Every unique set of labels create a new timeseries. Use labels with care watch out what you put into your labels. Avoid cardinality explosion, unbounded labels will blow up Prometheus. And keep in mind that labels are multiplicative. You will have multiple labels, multiple target labels and targets.

> Prometheus performance almost always comes down to one thing: label cardinality.

Always remember that Cardinality is key.

Naming is hard

THE ZEN OF PROMETHEUS

Simple, Poetic, Illuminating.

you should always guard against metric name collisions among the jobs on your system.

Respect conventions over preferences. Conventions are no one's favorite, yet conventions are everyone's favorite. See the documentation on Naming conventions for the nitty-gritty details.

One other benefit of following the conventions is that you don't need to reinvent the wheel. You can benefit from the tools that's out there, such as Monitoring Mixins.

Counters rule and gauges suck

If you can express as a counter, use a counter and do everything else later. Counters are powerful, you can derive lots of things from them. Remember Instrument first, ask questions later. Especially, please don't add metrics for aggregations, PromQL can do it for you.

First the rate, then aggregate

Be aware of counter resets. As stated in Rate then sum, never sum then rate.

> As a rule of thumb, the only mathematical operations you can safely directly apply to a counter's values are `rate`, `irate`, `increase`, and `resets`. Anything else will cause you problems.

If you can log it, you can have a metric for it

> Logs and metrics complement each other: metrics give the insight that something isn't

THE ZEN OF PROMETHEUS

Simple, Poetic, Illuminating.

Whenever you handle an error (either by returning it or logging it), you should ask yourself whether you can add some metrics and be able to alert on it. Spread the metrics liberally. Remember metrics are cheap.

One does not simply use Histograms

Histograms are powerful.

Creating a correct bucket layout for your histograms is an art. To ensure usefulness of your observations and correctness of your alerts, you have come up with a meaningful bucket layout. And keep in mind that they are cumulative. This is also somewhat conflicting with Instrument first, ask questions later because you need to have idea about your latencies before you even measure. To circumvent this issue you can take and iterative approach or use an event system to obtain your latency distribution.

And as always, let your SLOs guide your bucket layout, create boundaries to match your SLO.

The histograms underneath are just counter with labels; where bucket boundaries used as labels. Be precautious while adding additional labels to your histograms. Remember *Labels are multiplicative* and Cardinality Matters.

If you can graph it, you can alert on it

You can't look at dashboards 24/7. Prometheus unified metrics, dashboarding, and alerting. PromQL is the core of every Prometheus alert, and a PromQL query is the source of any graph on a dashboard. That is very powerful. Use it.

THE ZEN OF PROMETHEUS

Simple, Poetic, Illuminating.

Always have an alert -*at least*- on presence and healthiness of your targets. You can't that rely on the things and take action that you can't observe.

Avoid missing targets and unhealthy targets. All of the client libraries provides `up` metric by default. Use it.

Alerts should be urgent, important, actionable, and real

Alerts should be urgent, important, actionable, and real. As plain as it goes.

And don't over alert, alert-fatigue is real.

Symptom-based alerts for paging, caused-based for troubleshooting

Similar to Measure what users care most about, alert on what *really* matters. It doesn't matter if you are CPU is saturated, as long as your users don't notice. Let your SLOs guide your alerting. For more context My Philosophy on Alerting.

Please five more minutes

Prometheus alerting rules let you to specify a time duration that determines after how long an alert should start firing according to given query, which is called `FOR`. If you don't specify it a single failed scrape could cause an alert to fire. You need more tolerance. Don't make it short and always specify it. And also don't make it too long. For more information check out Setting Thresholds on Alerts

Context is king

THE ZEN OF PROMETHEUS

Simple, Poetic, Illuminating.

The inspiration behind the idea comes from The Zen of Go, Go Proverbs and The Zen of Python.

Thank you very much all others that contributed with their invaluable ideas.

> The initial rules are gather from several sources from the community, such as Prometheus Proverbs by Björn Rabenstein, Best Practices and Beastly Pitfalls by Julius Volz, Patterns for Instrumenting Your Go Services by Bartek Plotka and Kemal Akkoyun, Instrumenting Applications and Alerting with Prometheus by Simon Pasquier and Robust Perception Blog by Brian Brazil.