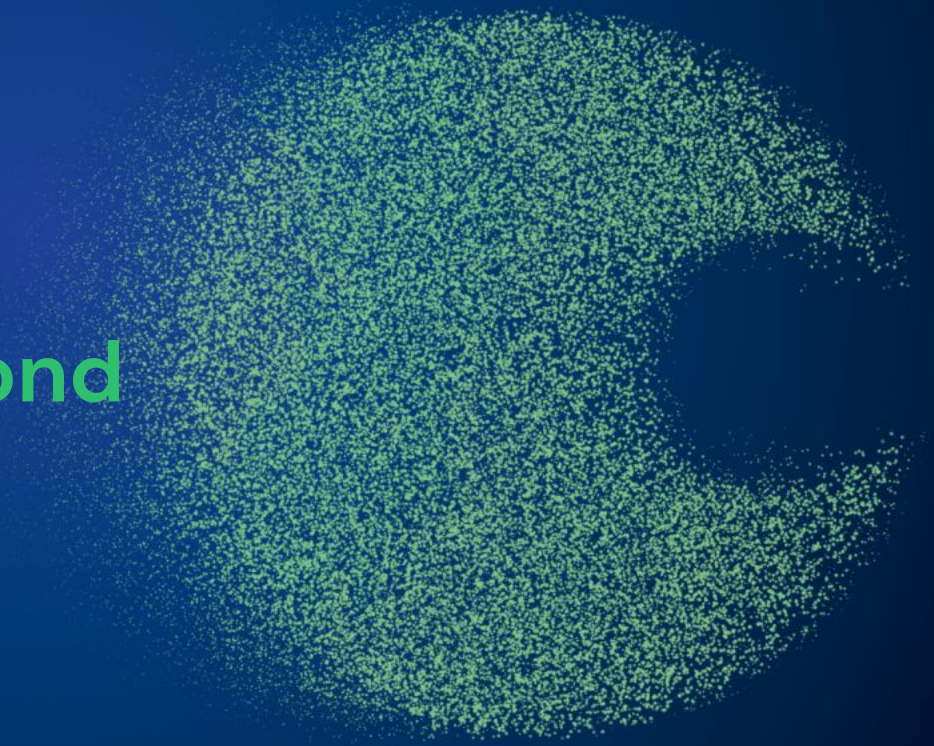
chronosphere DOORDASH

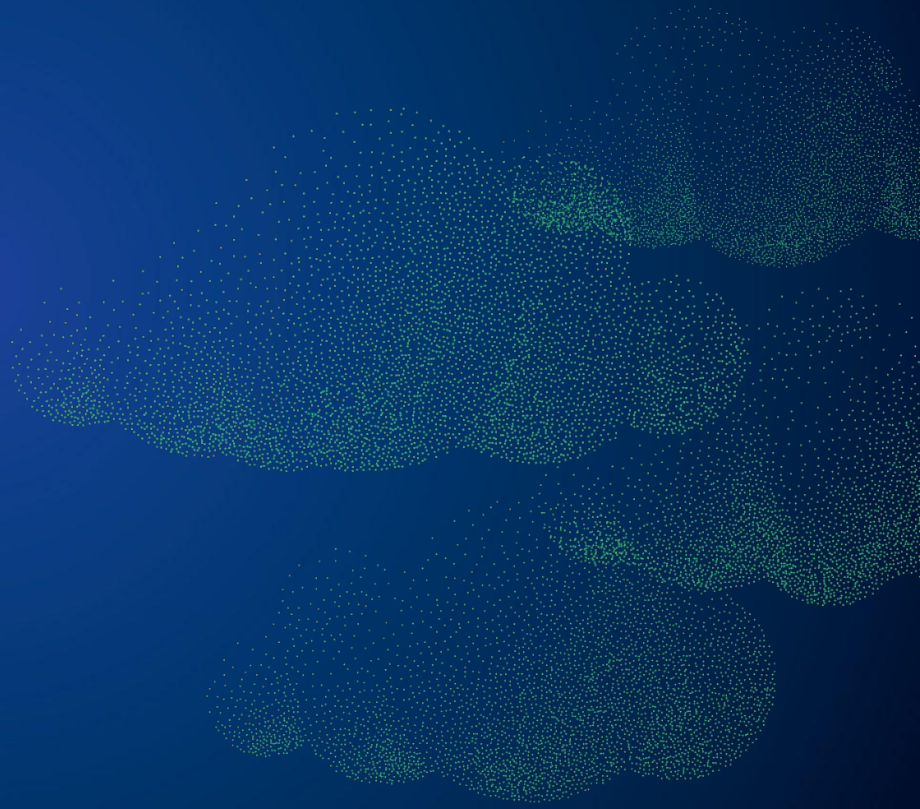
DoorDash's Journey From StatsD To Prometheus With 10 Million Metrics/Second



Benjamin Raskin
Emma Wang

28 October 2022

Introductions



chronosphere

Who are we?

Ben Raskin

Solutions Architect

chronosphere

Emma Wang

Infrastructure Software Engineer

DOORDASH

chronosphere

Agenda

1. StatsD vs Prometheus
2. Client Side Migration
3. Enablement
4. Learnings/Results

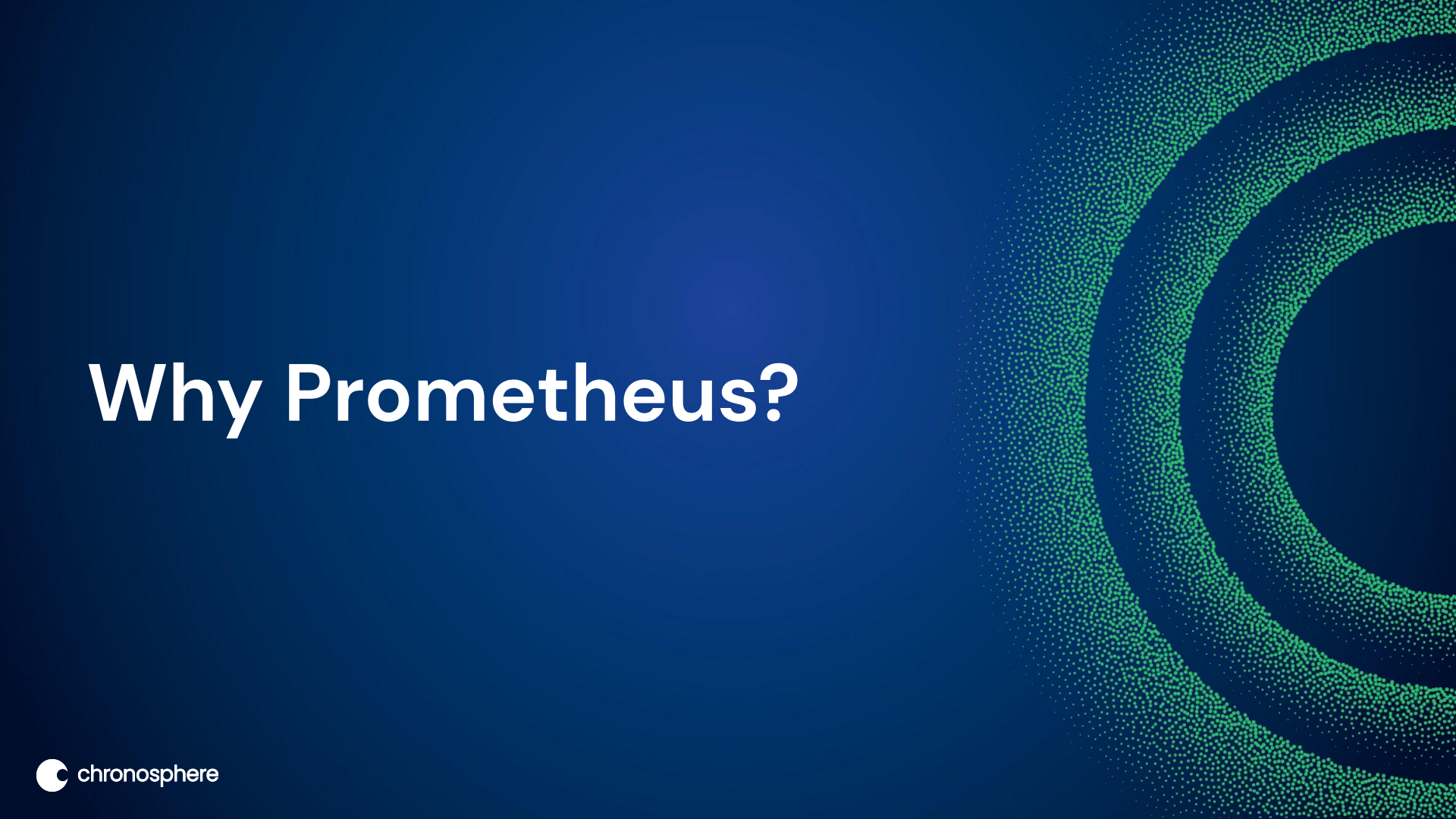
Challenges with StatsD

chronosphere

Challenges with StatsD

- **Lack of naming standardization**/limited support for tags
  - Service A: service_a.page_views.us-east-1
  - Service B: service_b.us-east-1.page_views
- Number of metrics **scales with user traffic**
  - Requires an aggregation tier; otherwise metrics can grow exponentially
- **Sent over UDP** (potential for packet loss)
- **Delta counters**
  - No way to interpolate missing data points
- **Inability to aggregate percentiles**/lack of histograms

Why Prometheus?



DoorDash Observability Requirements

Use Open Source

- Cost efficiency
- Integration
  - Open source data formats and open source query languages.
- No vendor lock-in

Governance and Control

- Standard Conventions
  - Standards for common tags, metrics naming conventions.
- Cost accounting
  - Ability to breakdown usage by team, service, etc.

Self-Service

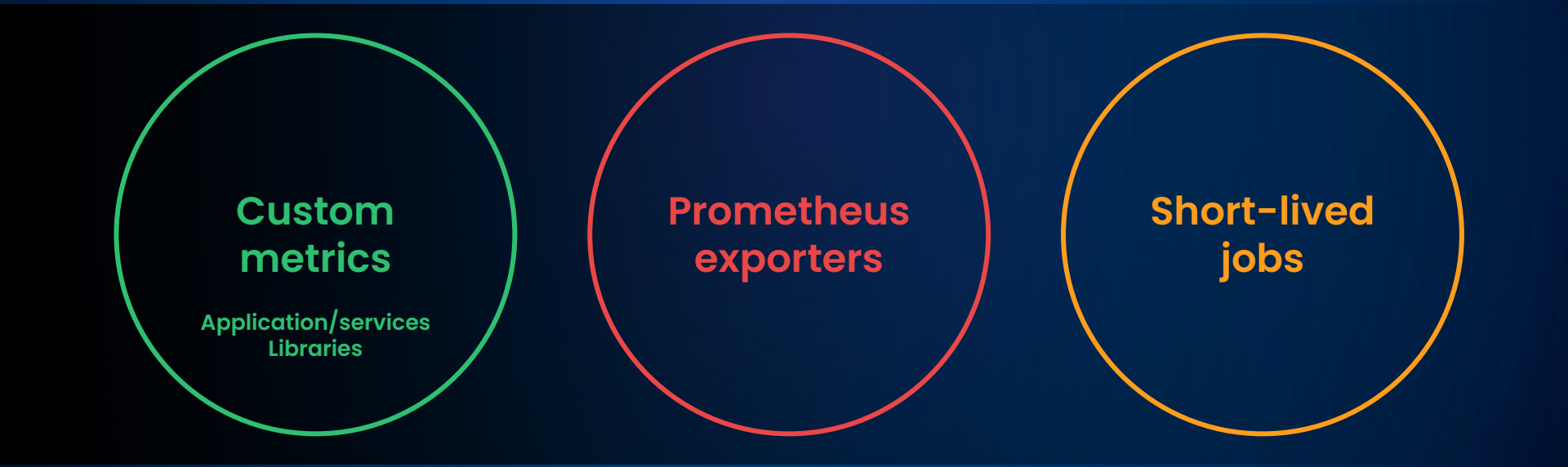
- Productivity Empowerment
  - For newly registered services, the metrics can be automatically discovered and collected.
  - Automate the process of generating basic dashboards and alerts.

Why DoorDash Chose Prometheus

- Prometheus has emerged as the dominant standard for open source metrics and aligned well with the organization's strategy and requirements.
  - Open Source Standard
  - Query Language (PromQL)
  - Tag-Based Metric Ingestion Format
  - Pull-based system (allows for client-side aggregation)
  - Native support for histograms and cumulative counters
- Strong support for the more fundamental tools used in DoorDash. eg: Kubernetes

Client Side Migration

Metric generation



Custom Metrics

- Application and service metrics
  - microservices use prometheus native libraries to generate metrics.
  - pay attention to python clients with a multi-process pattern.
- Internal libraries
  - provide the internal library to generate HTTP/GRPC metrics automatically.
  - JVM prometheus exporter is included in the doordash docker base images to export JVM metrics.

Some Other Metrics

- Open Source Exporters
  - Wide-range of available exporters maintained by the open-source community
  - AWS Cloudwatch Metrics: official prometheus cloudwatch exporter VS YACE cloudwatch exporter
  - StatsD exporter
  - JVM exporter
  - Pgbouncer exporter
- Short-Lived Jobs
  - can not be scraped by metrics collectors
  - use the prom aggregation gateway with a small modification

Standard Tags For All Metrics

Common Tags:

- service
- app
- k8s_cluster
- environment
- sub_environment
- region
- zone
- cost_origin

Service Discovery (thru standardization)

- Service discovery done through Kubernetes annotations, which tell Prometheus what endpoints to scrape.
  - Central Observability Team (COT) creates gold-standard Kubernetes manifest templates, which services use.
- Default scrape options are also defined by the COT.
  - Scrape frequency
  - Standard labels (prev slide)

Reducing Cardinality & Improving Performance

Relabel rules

- Rewrite the label set of a target.
- Consolidate common labels and drop unused metrics or labels.

During the scrape event

Rollup rules using M3

- Eliminate labels that are not needed by the service owners.
- For example, the instance or pod id can be removed.

Server side aggregation

Mapping rules using M3

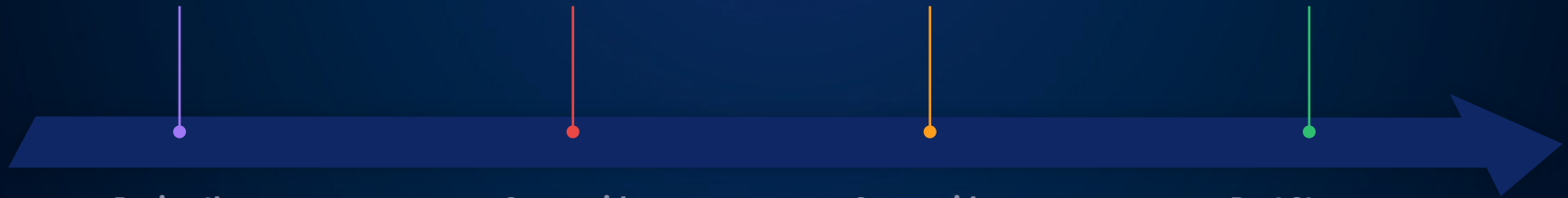
- Define the storage policy for each time series.
- Drop unnecessary metrics after rollups are applied.

Server side aggregation

Recording rules

- Pre-compute expensive expressions to speed up query time.

Post Storage

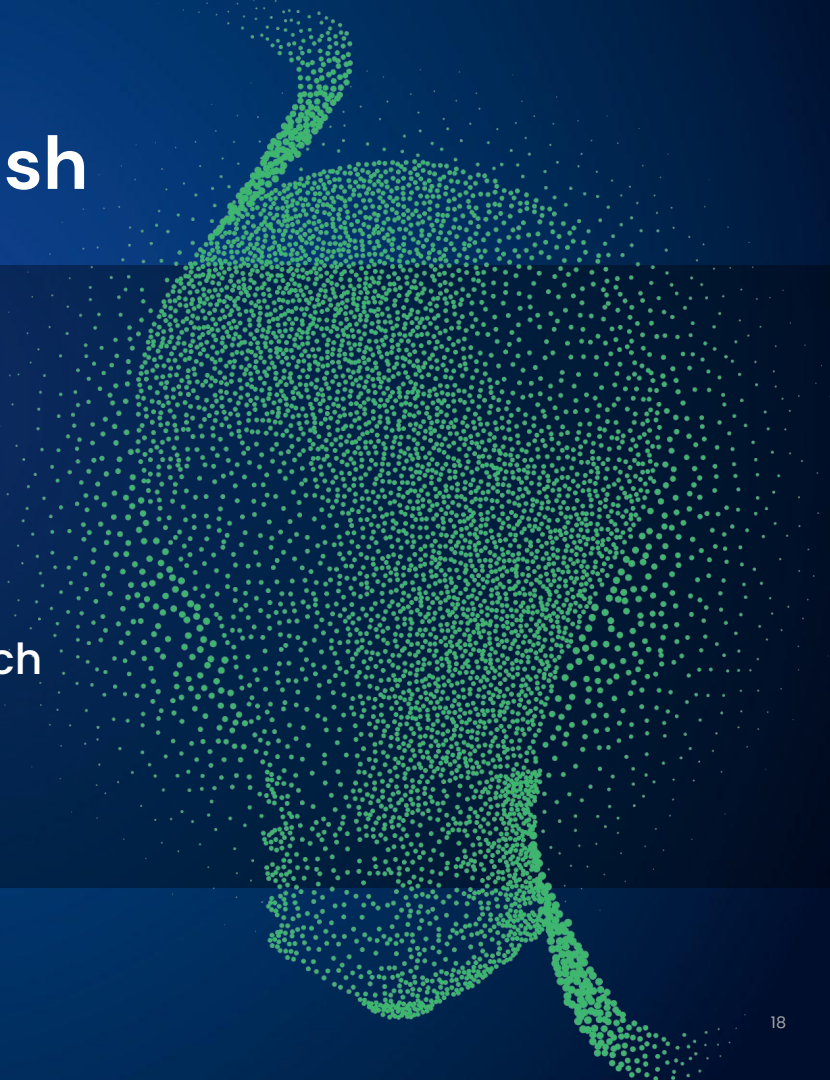


Enablement

chronosphere

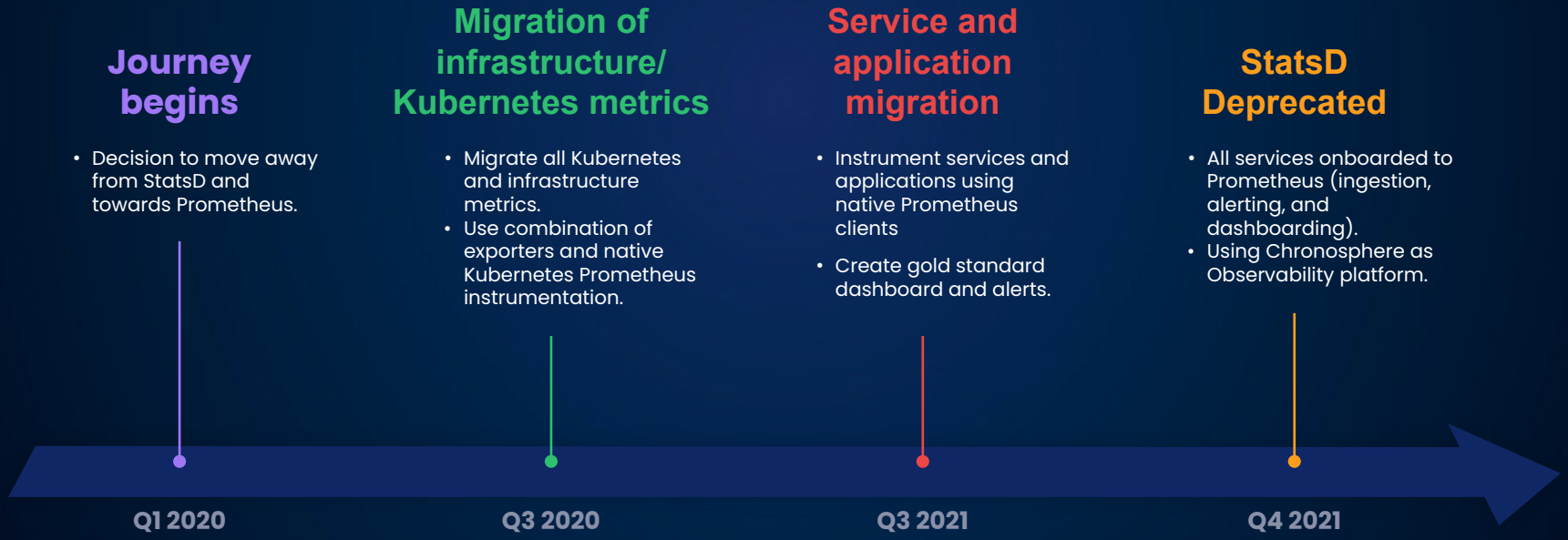
Learning (and teaching) to fish

- Prometheus data model
  - Best practices!
  - How to name things…
- PromQL
  - Counters (rate, rate, rate!)
  - Histograms vs. timers
- Not everything needs to be built from scratch
  - Open-source and gold standards
- Managing resources through Terraform

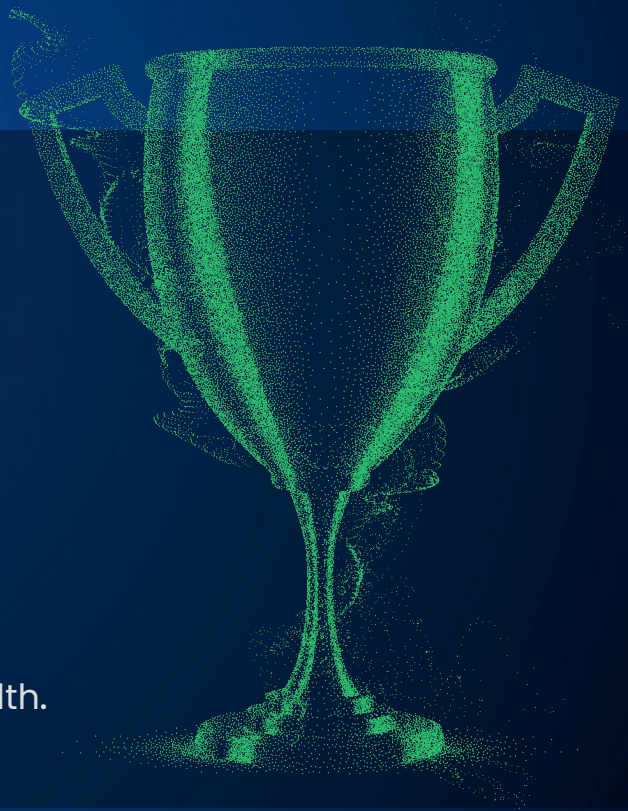


Timeline, Results, and Learnings

Timeline


Journey begins
• Decision to move away from StatsD and towards Prometheus.
Q1 2020
Migration of infrastructure/ Kubernetes metrics
• Migrate all Kubernetes and infrastructure metrics.
• Use combination of exporters and native Kubernetes Prometheus instrumentation.
Q3 2020
Service and application migration
• Instrument services and applications using native Prometheus clients
• Create gold standard dashboard and alerts.
Q3 2021
StatsD Deprecated
• All services onboarded to Prometheus (ingestion, alerting, and dashboarding).
• Using Chronosphere as Observability platform.
Q4 2021

Achievements and Wins



starting with more than 7000 alerts, 1500 dashboards and metrics for 130 services, this has been a massive collaboration across all teams and engineers.

One year post migration, we now have:

- ~27k alerts
- ~2.2k dashboards
- 15M metrics/s ingested
- 10M metrics/s persisted (after aggregation)

The Prometheus based monitoring system is stable, scalable, and flexible, providing a greatly improved view of DoorDash service health.

Learnings

Switching from a percentile to a histogram is tough

The instance label is an important concern for overall volume of metrics expected after the move

Prometheus is not friendly with short lifecycle jobs

Automate the monitoring onboarding process for teams

Come by Booth G15 to:

- Win the Ultimate Gaming Setup
- Test your gaming skills at O11Y Legend
- Win daily prizes at 4pm
- Talk to the best in observability

Trusted by the best in the industry

Abnormal Snap Inc. Robinhood DOORDASH

Scan the QR code

chronosphere

Thank You

chronosphere

Use Codification For Alerts

Choose Terraform based on:

- source control
- Locks on state files
- Alert templating

Some caveats:

- State management
- Performance at scale