Prometheus

ContribFest

Instrument Go Application

Prometheus team members

Agenda

1. How Prometheus Works & Concepts
2. Prometheus Data Model
3. Task: Instrument Go Application

What is Prometheus?

Metrics-based monitoring & alerting stack.

- Instrumentation for applications and systems
- Metrics collection and storage
- Querying, alerting, dashboarding
- For all levels of the stack!

Made for dynamic cloud environments.

Architecture

Architecture

Targets

web app

API server

web app

mysqld

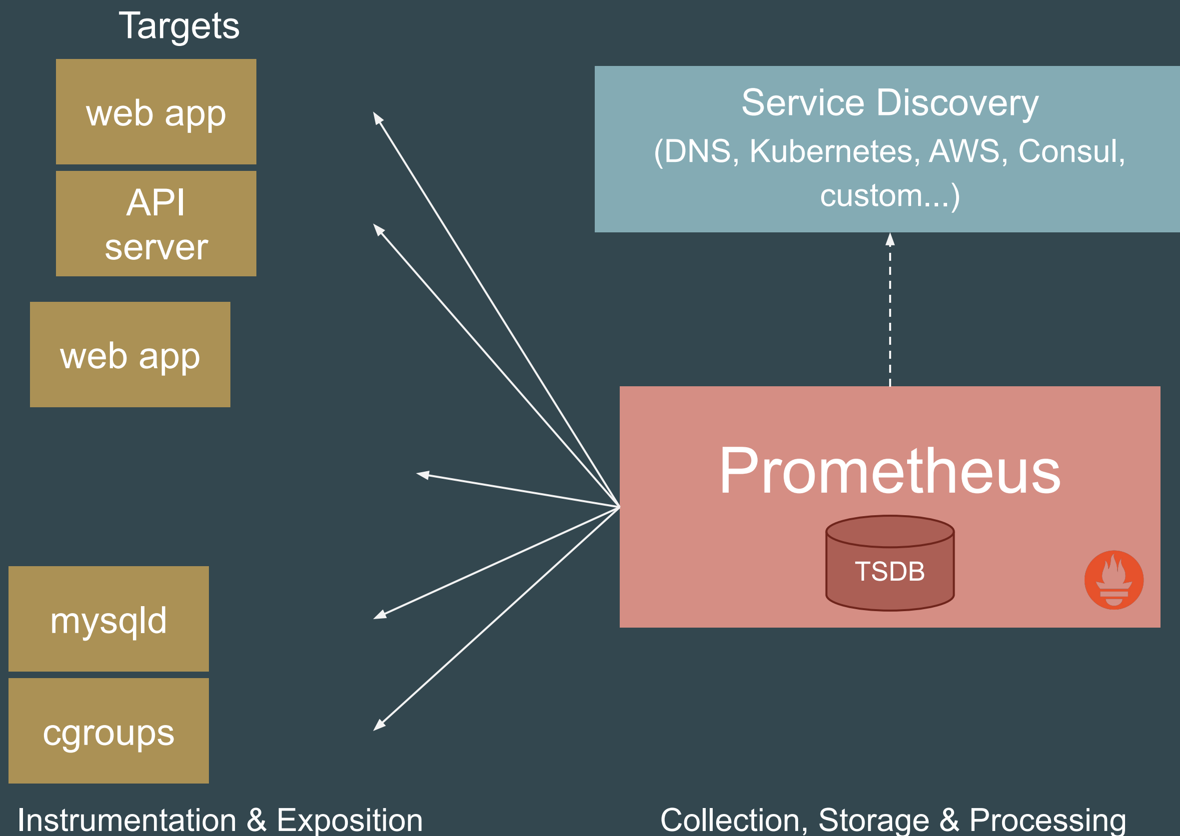
cgroups

Instrumentation & Exposition

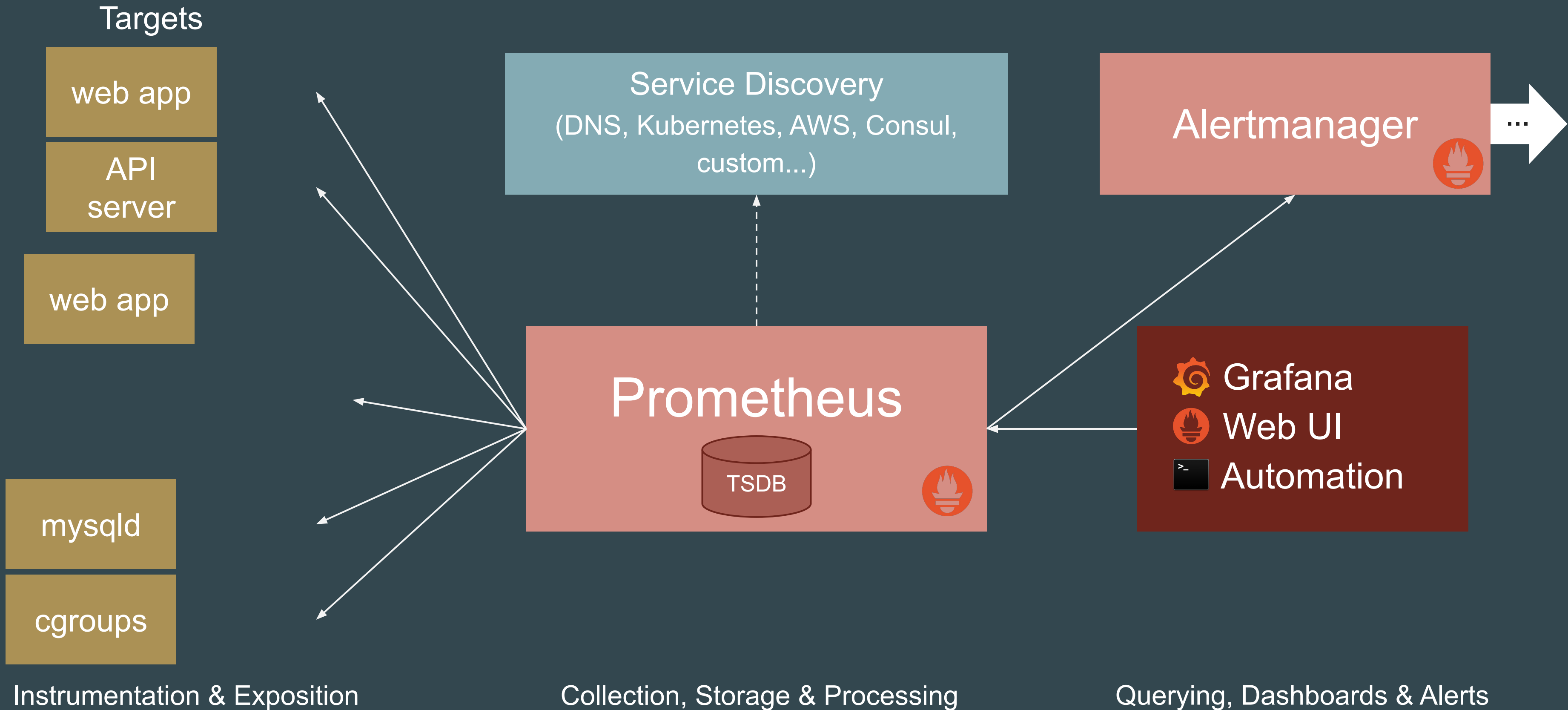
Collection, Storage & Processing

Querying, Dashboards & Alerts

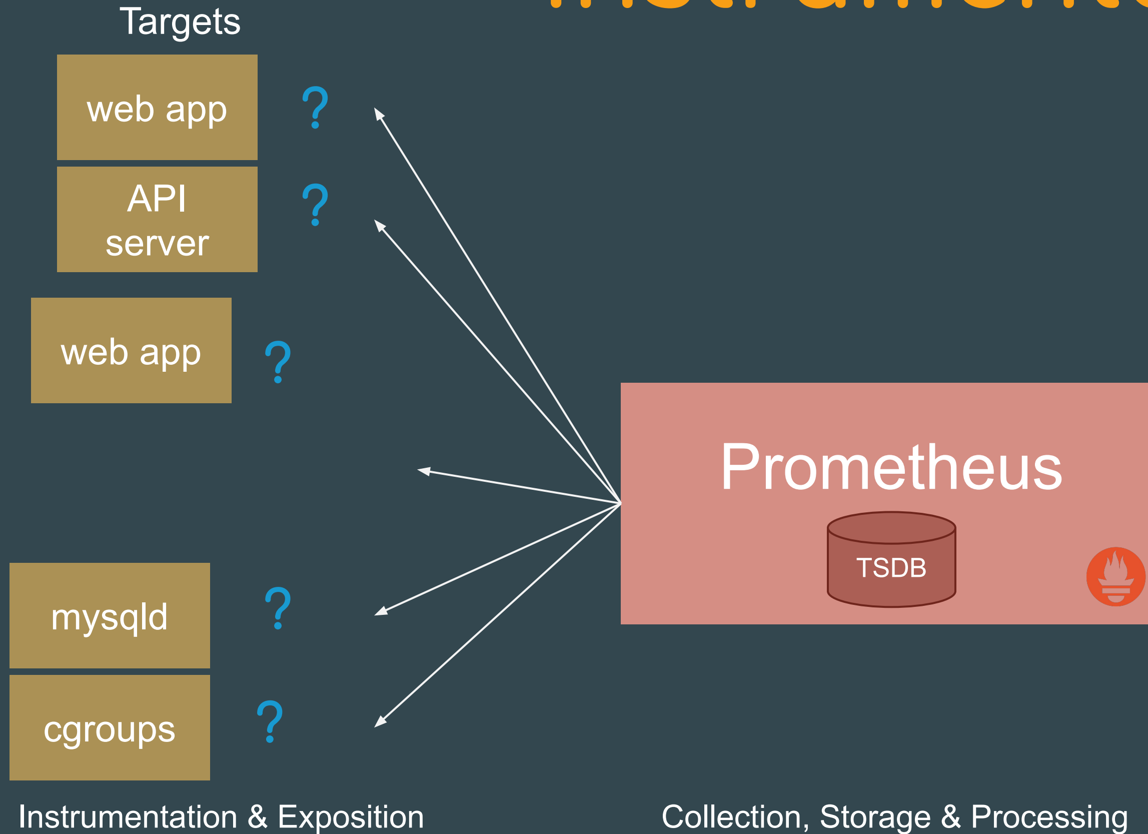
Architecture



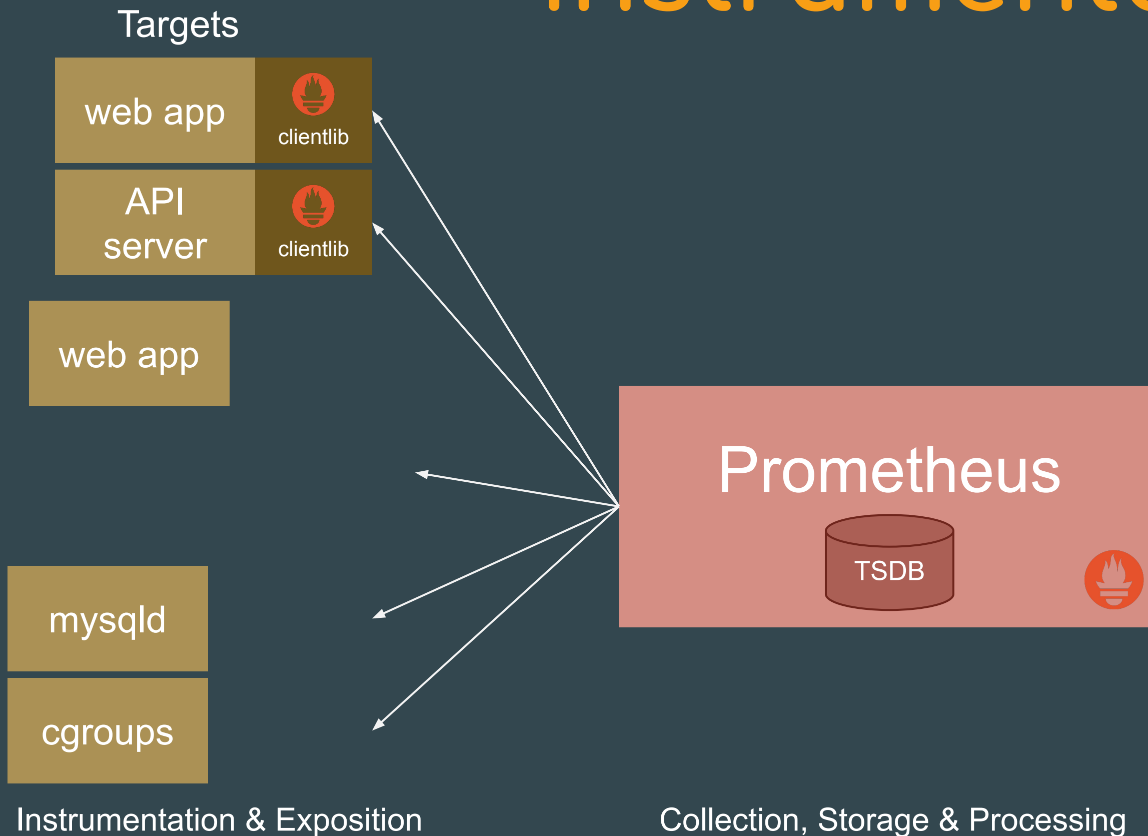
Architecture



Instrumentation



Instrumentation



Instrumentation

Targets

“Manual“

web app

API server

“Auto“

web app

service mesh / eBPF

mysqld

cgroups

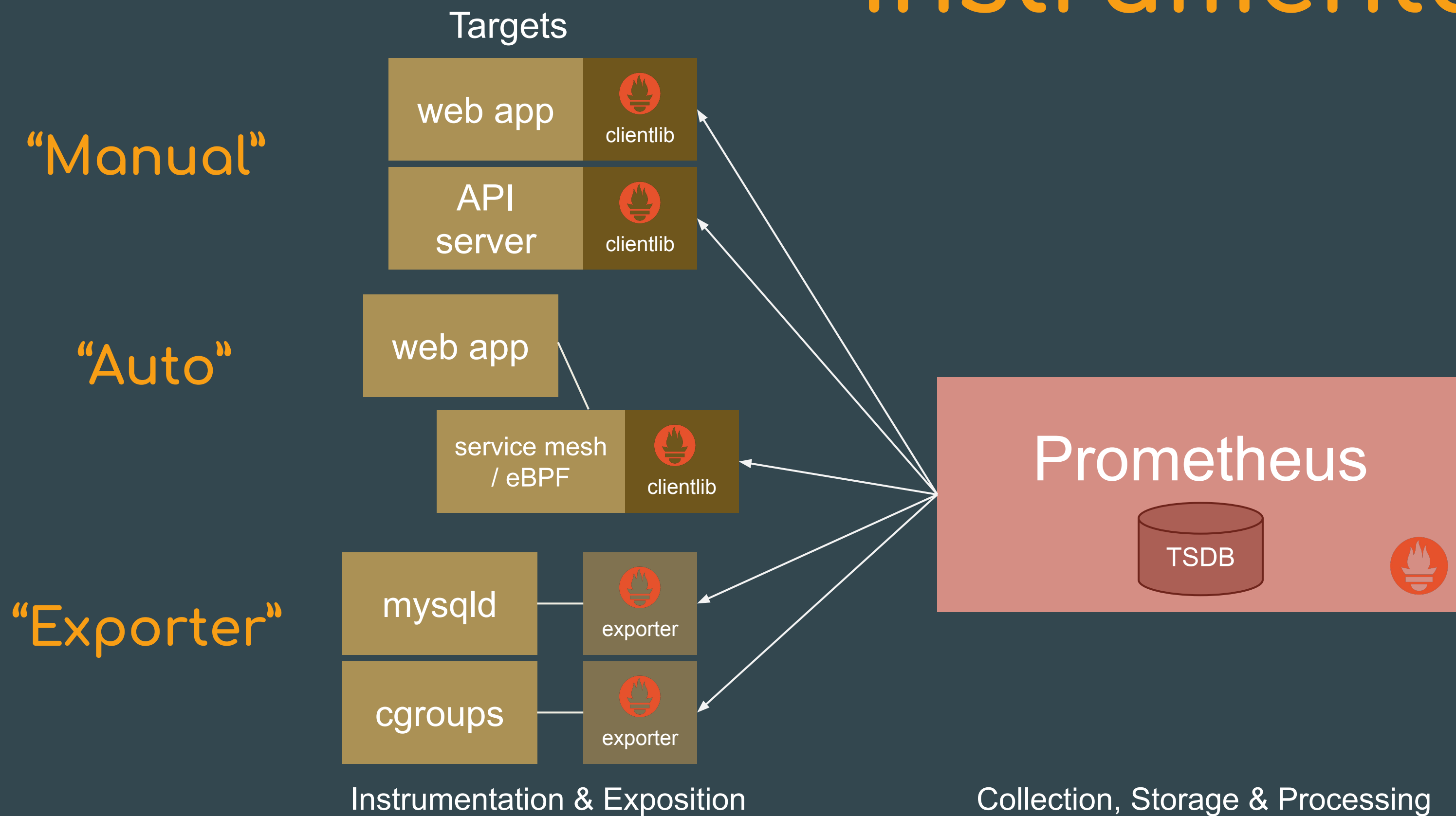
Prometheus

TSDB

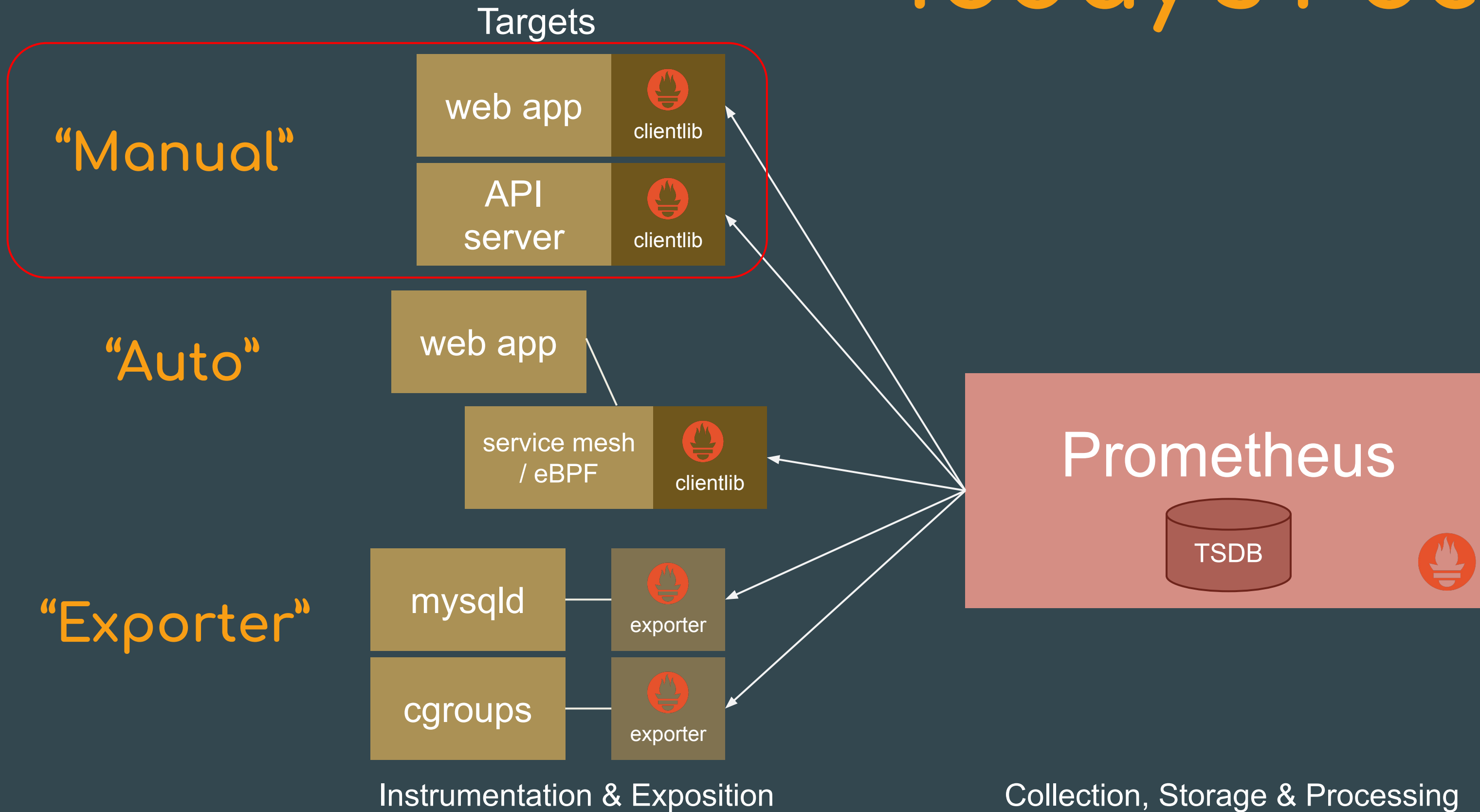
Instrumentation & Exposition

Collection, Storage & Processing

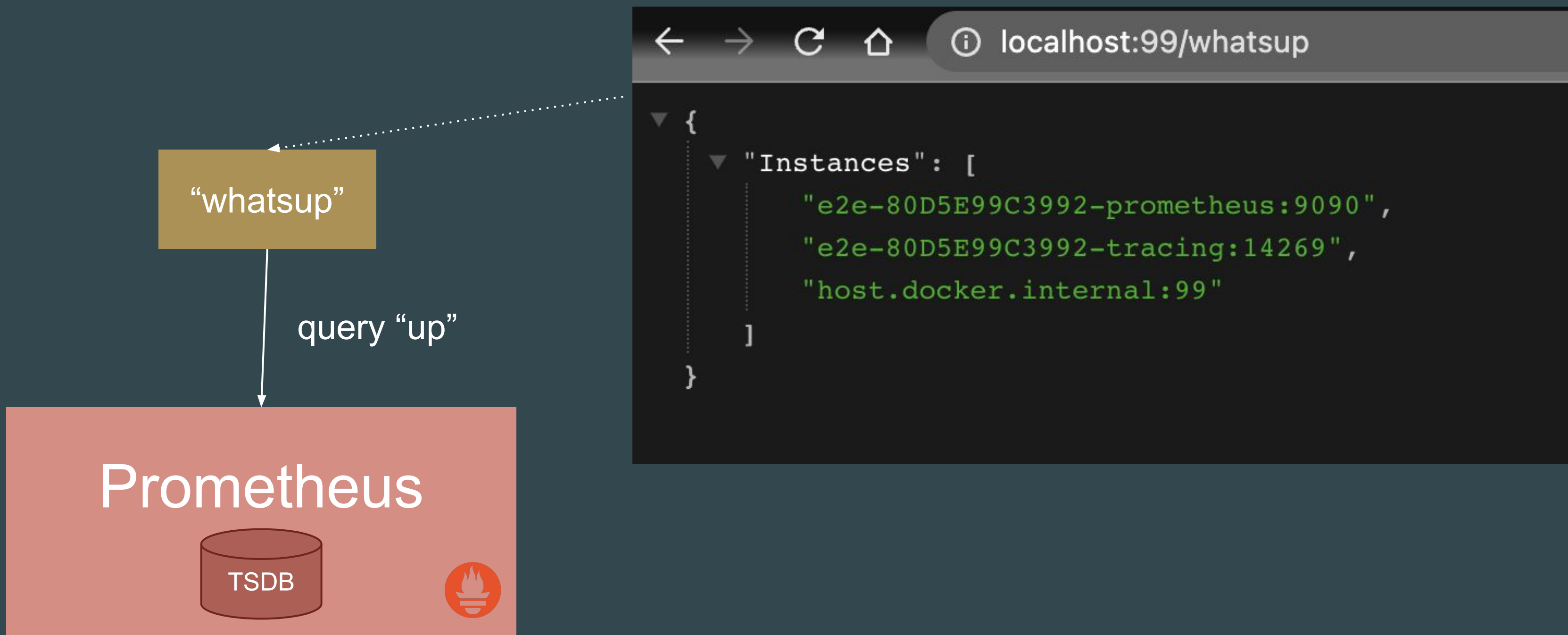
Instrumentation



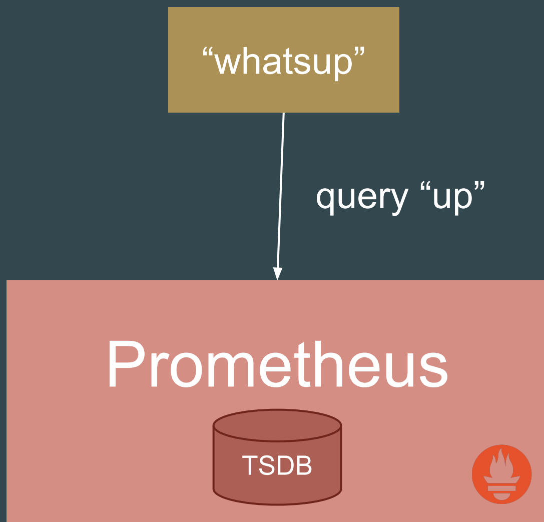
Today's Focus



Today's Challenge



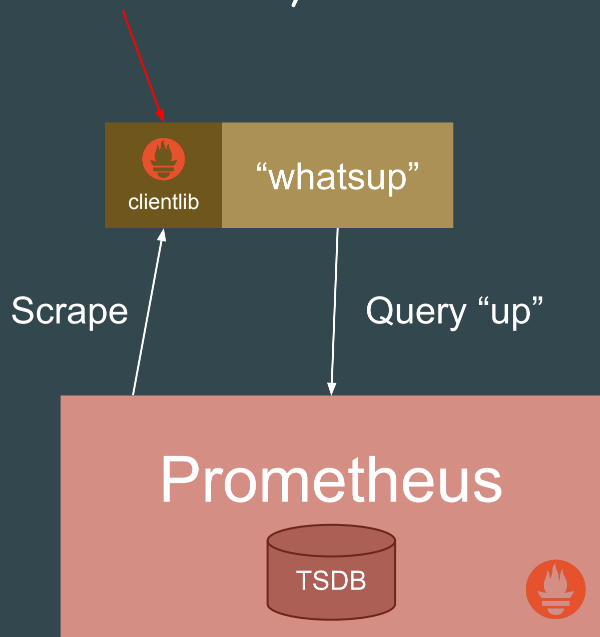
Today’s Challenge



When deployed:

1. Is the “whatsup” up?
2. How many HTTP calls it handled?
3. How fast it handled them?
4. What’s average response size?
5. What version it is running?
6. How much memory & CPU it’s using?

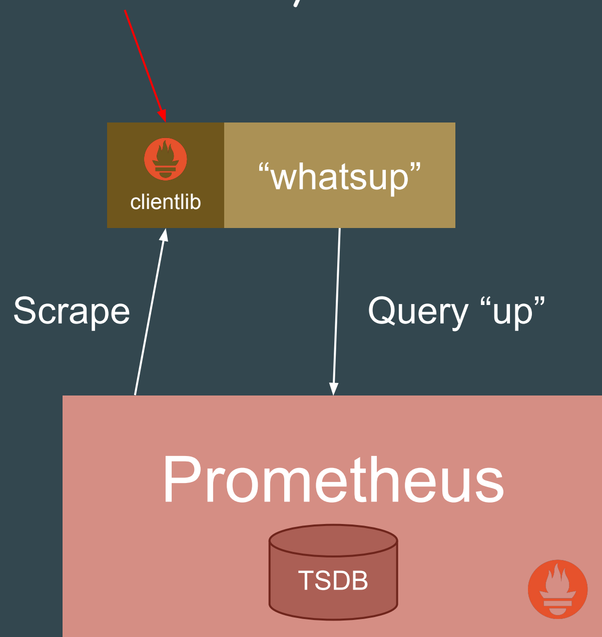
Today's Challenge



When deployed:

1. Is the "whatsup" up?
2. How many HTTP calls it handled?
3. How fast it handled them?
4. What's typical response size?
5. What version it is running?
6. How much memory & CPU it's using?

Today's Challenge



All using client_golang!

https://github.com/prometheus/client_golang

```
✗ git clone git@github.com:prometheus/client_golang.gi
```

Metric Types

Metric Types

1. Counter
2. Gauge (also can represent Info Type)
3. Histogram (Classic and Native)

Bonus: Exemplars

https://prometheus.io/docs/concepts/metric_types/

Counter

"How many HTTP calls it handled?"

```
handled := promauto.With(reg).NewCounter(prometheus.CounterOpts{
    Name: "whatsup_queries_handled_total",
})
```

```
handled.Inc()
```

Counter + Exemplar

"How many HTTP calls it handled?"
"Also what's the traceID for example call that was handled?"

```go
handled := promauto.With(reg).NewCounter(prometheus.CounterOpts{
    Name: "whatsup_queries_handled_total",
})
```

```go
handled.(prometheus.ExemplarAdder).
    AddWithExemplar(1, getExemplarFn(ctx))
```

```go
func getExemplarFn(ctx context.Context) prometheus.Labels {  5 usages  bwplotka
    if spanCtx := tracing.GetSpan(ctx); spanCtx.Context().IsSampled() {
        return prometheus.Labels{"traceID": spanCtx.Context().TraceID()}
    }
    return nil
}
```

Gauge

"What's typical response size?"

```
lastNumElems := promauto.With(reg).NewGauge(prometheus.GaugeOpts{
    Name: "whatsup_last_response_elements",
})
```

```
    lastNumElems.Set(float64(len(resp.Instances)))
```

Gauge (Info)

"What version it is running?"

```
promauto.With(reg).NewGaugeFunc(prometheus.GaugeOpts{
    Name: "build_info",
    ConstLabels: map[string]string{
        "version":  "vYOLO",
        "language": "Go 1.20",
        "owner":    "@me",
    },
}, func() float64 {
    return 1
})
```

Histogram

"How fast it handled them?"

```go
handledDuration := promauto.With(reg).NewHistogram(
	prometheus.HistogramOpts{
		Name:    "whatsup_queries_duration_seconds",
		Help:    "Tracks the latencies for calls.",
		Buckets: []float64{0.1, 0.3, 0.6, 1, 3, 6, 9, 20},
	},
)
```

```go
handledDuration.Observe(time.Since(start).Seconds())
```

Histogram + Exemplars

"How fast it handled them?"
"Also what's traceID for example call that was faster than 100ms?"

```
handledDuration := promauto.With(reg).NewHistogram(
    prometheus.HistogramOpts{
        Name:    "whatsup_queries_duration_seconds",
        Help:    "Tracks the latencies for calls.",
        Buckets: []float64{0.1, 0.3, 0.6, 1, 3, 6, 9, 20},
    },
)
```

```
handledDuration.(prometheus.ExemplarObserver).
    ObserveWithExemplar(time.Since(start).Seconds(), getExemplarFn(ctx))
```

Group of Metrics: Collectors

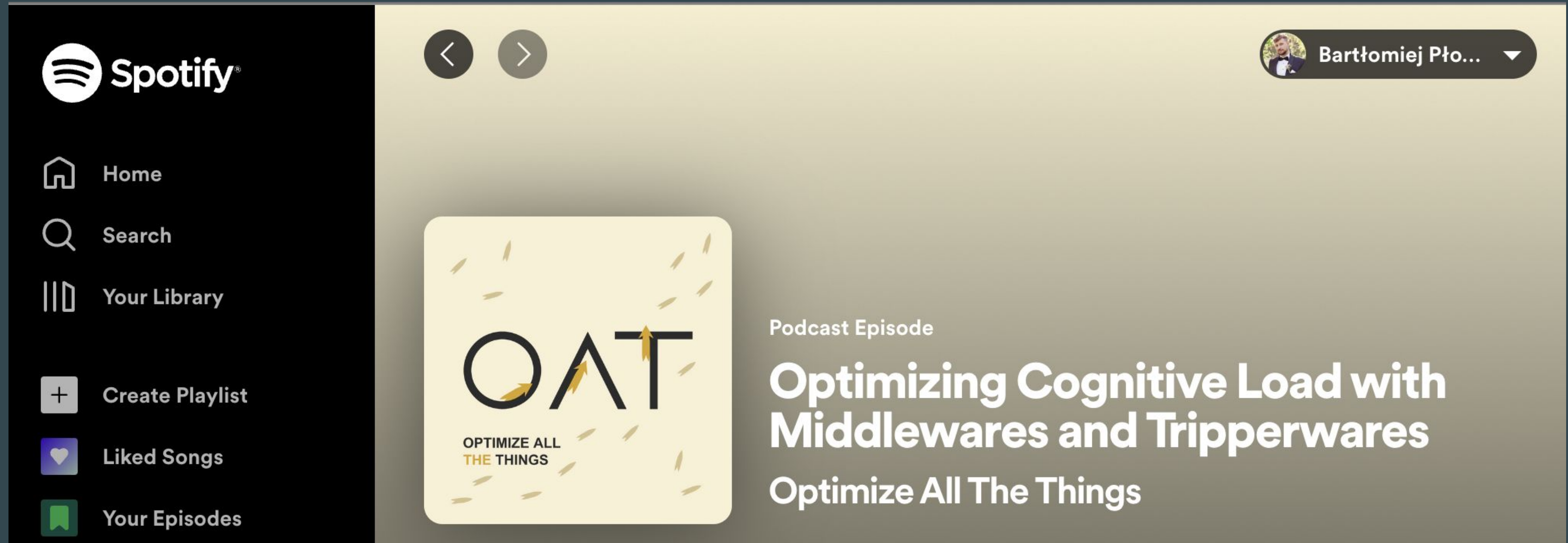
```go
// Create registry for Prometheus metrics.
reg := prometheus.NewRegistry()
reg.MustRegister(
	collectors.NewGoCollector(),                                       // Metrics from Go runtime.
	collectors.NewProcessCollector(collectors.ProcessCollectorOpts{}), // Metrics about the current UNIX process.
)
```

Middlewares/Tripperwares

```go
requestDuration := promauto.With(reg).NewHistogramVec(
	prometheus.HistogramOpts{
		Name:    "http_request_duration_seconds",
		Help:    "Tracks the latencies for HTTP requests."
		Buckets: []float64{0.1, 0.3, 0.6, 1, 3, 6, 9, 20},
	},
	[]string{"method", "code"},
)
requestSize := promauto.With(reg).NewSummaryVec(
	prometheus.SummaryOpts{
		Name: "http_request_size_bytes",
		Help: "Tracks the size of HTTP requests.",
	},
	[]string{"method", "code"},
)
requestsTotal := promauto.With(reg).NewCounterVec(
	prometheus.CounterOpts{
		Name: "http_requests_total",
		Help: "Tracks the number of HTTP requests.",
	}, []string{"method", "code"},
)
responseSize := promauto.With(reg).NewSummaryVec(
	prometheus.SummaryOpts{
		Name: "http_response_size_bytes",
		Help: "Tracks the size of HTTP responses.",
	},
	[]string{"method", "code"},
)
```

```go
base := promhttp.InstrumentHandlerRequestSize(
	requestSize,
	promhttp.InstrumentHandlerCounter(
		requestsTotal,
		promhttp.InstrumentHandlerResponseSize(
			responseSize,
			promhttp.InstrumentHandlerDuration(
				requestDuration,
				http.HandlerFunc(func(writer http.ResponseWriter, r *http.Request) {
					handler.ServeHTTP(writer, r)
				}),
				promhttp.WithExemplarFromContext(getExemplarFn),
			),
		),
		promhttp.WithExemplarFromContext(getExemplarFn),
	),
)
```

Middlewares/Tripperwares



https://rss.com/podcasts/oat

Scrape Endpoint

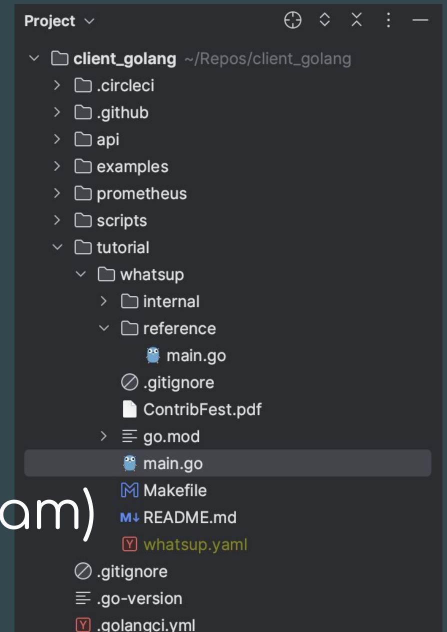
```go
// Create registry for Prometheus metrics.
reg := prometheus.NewRegistry()
```

```go
m := http.NewServeMux()
// Create HTTP handler for Prometheus metrics.
m.Handle("/metrics", promhttp.HandlerFor(
    reg,
    promhttp.HandlerOpts{
        // Opt into OpenMetrics e.g. to support exemplars.
        EnableOpenMetrics: true,
    },
))
```

Today's Task

1. Clone https://github.com/prometheus/client_golang
2. Go to **./tutorials/whatsup**
3. Modify **main.go** to have:

- Scrape Endpoint
- whatsup_queries_handled_total (counter)
- whatsup_last_response_elements (gauge)
- build_info (info gauge)
- whatsup_queries_duration_seconds (histogram)
- go_goroutines (gauge)



Testing & Verification

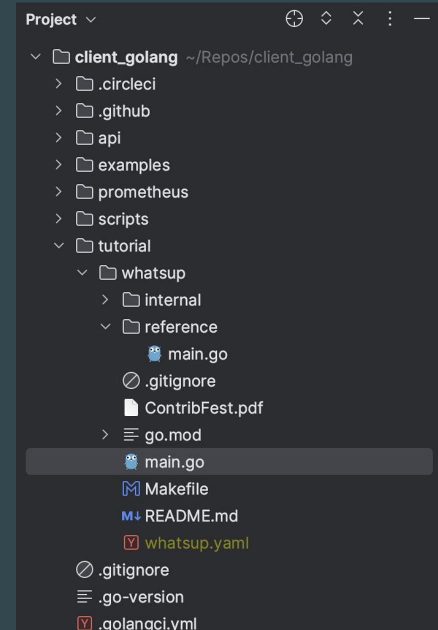
1. “**make init**” to run Prometheus and Jaeger in background
   (make stop to stop it)
2. “**make run**” to run main.go (whatsup)
   (Ctrl+C to stop it)

Now you can:

a. Run “make metrics” to get main.go metrics to explore
b. Open Prometheus UI and query metrics
c. Run “make test” to run acceptance test

Stuck? Check reference impl!

Check ./tutorials/whatsup/reference/main.go for example working solution!



Prerequisites for local runs:

1. Go 1.18+
2. git, make
3. docker

Engage with Promethus!

COMMUNITY

https://prometheus.io/community/

Prometheus is developed in the open. Here are some of the channels we use to communicate and contribute:

IRC: `#prometheus` on irc.libera.chat. This channel is bridged to the Matrix room below.

Matrix: `#prometheus:matrix.org`. This room is bridged to the IRC room above.

Community-maintained Slack channel: `#prometheus` on CNCF Slack.

User mailing lists:

- prometheus-announce (mirror) – low-traffic list for announcements like new releases.
- prometheus-users (mirror) – for discussions around Prometheus usage and community support. Announcements are *not* generally mirrored from prometheus-announce.

Discourse forum: Web-based discussion forum at discuss.prometheus.io hosted by Discourse.

Calendar for public events: We have a public calendar for events, which you can use to join us.

If you just want to get an overview, simply use our web view in your browser's time zone.

If you're using Google products, there's an automagic link to add it your own Google calendar.

If you're using a different calendar, there's an .ics to add to non-Google calendars.

Twitter: @PrometheusIO

GitHub: To file bugs and feature requests, use the GitHub issue tracker of the relevant Prometheus repository. For questions and discussions, many repositories offer GitHub discussions. Generally, the other community channels listed here are best suited to get support or discuss overarching topics.