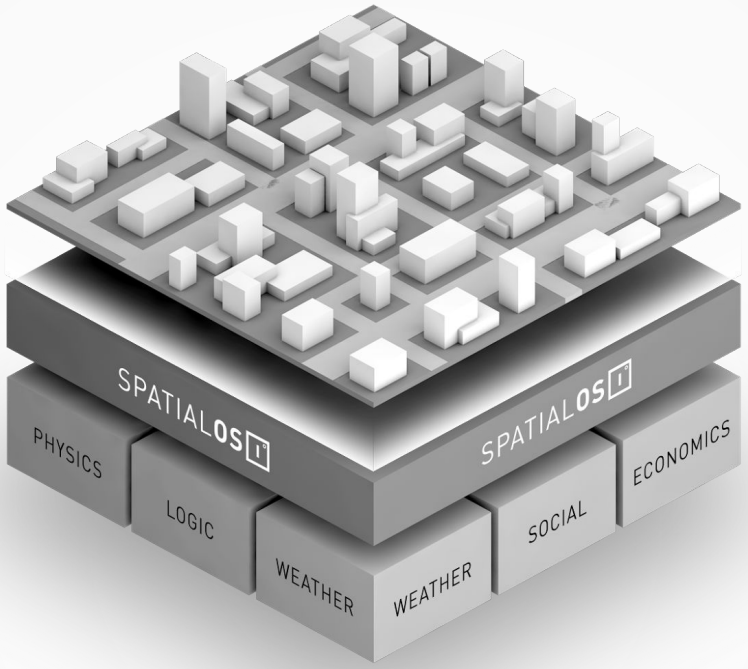
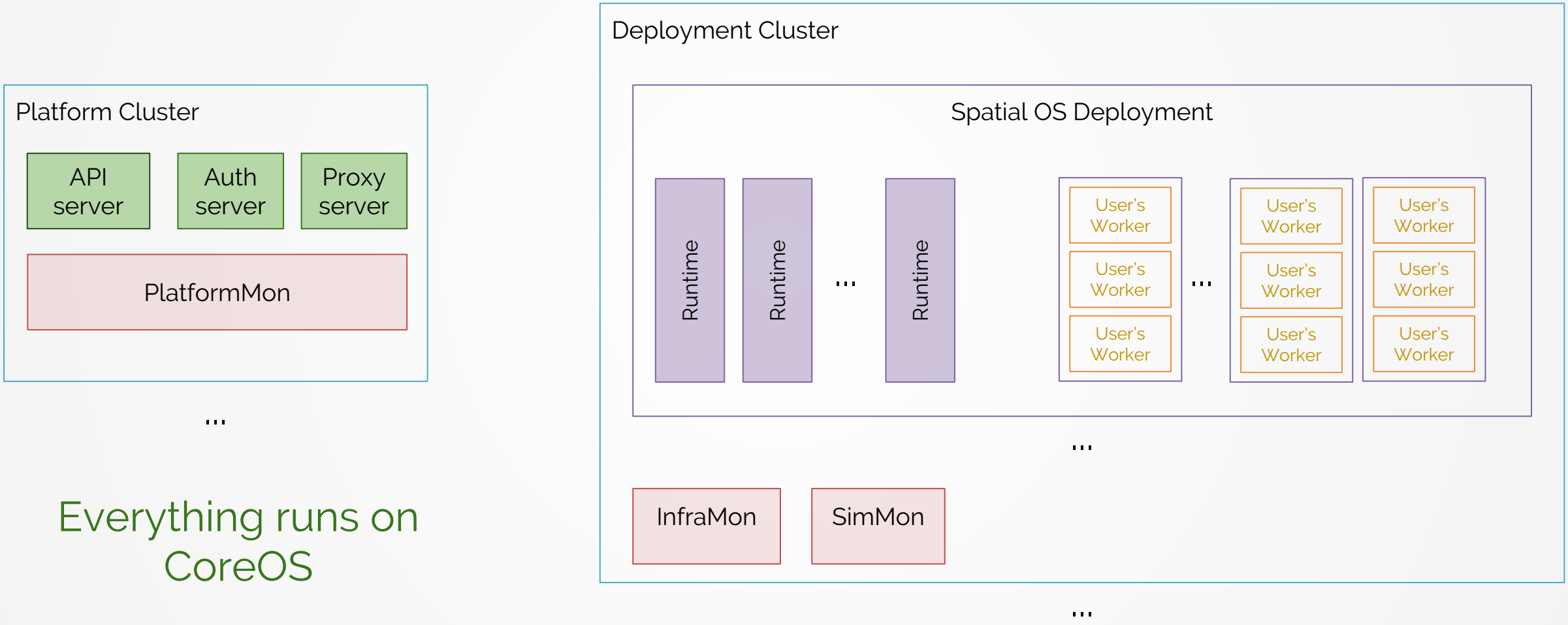
Prometheus

dima@improbable.io



SpatialOS is a distributed operating system enabling massive, real-time, simulations of spatial problems running on thousands of machines in the cloud.

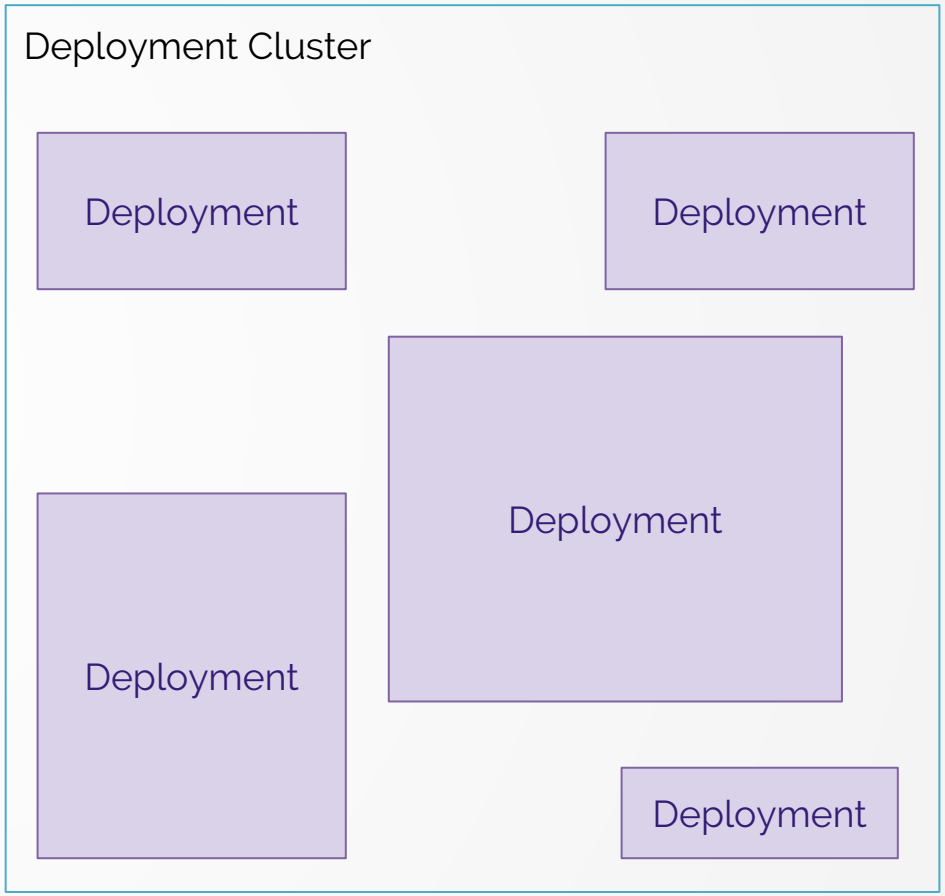
Spatial OS From 10,000 feet



Everything runs on CoreOS

SpatialOS Deployment are not usual jobs

- Short-lived
  - Runtime Experiments
  - User Development
  - Testbeds
- Long-Lived
  - Running games
  - Live city simulations
- Small (1 machine), Huge (~hundreds)
- Dynamically named



What does Prometheus do for us?

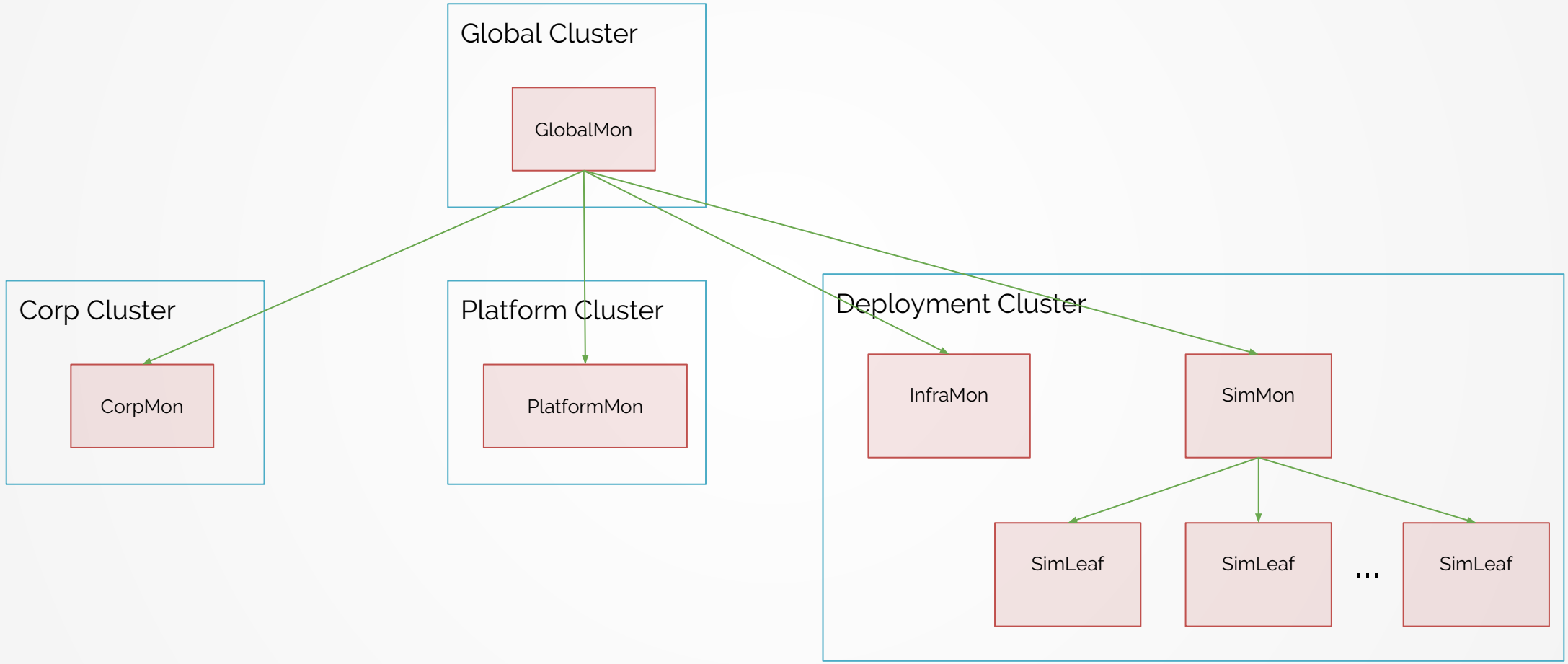
- Monitoring of production services
- Monitoring of corp services
- Alerting

What does Prometheus do for our customers?

- Metrics for users to debug/QA/measure performance of their deployments
- Used for game design
- Alerting

Infra/Global/Platform/SimMons

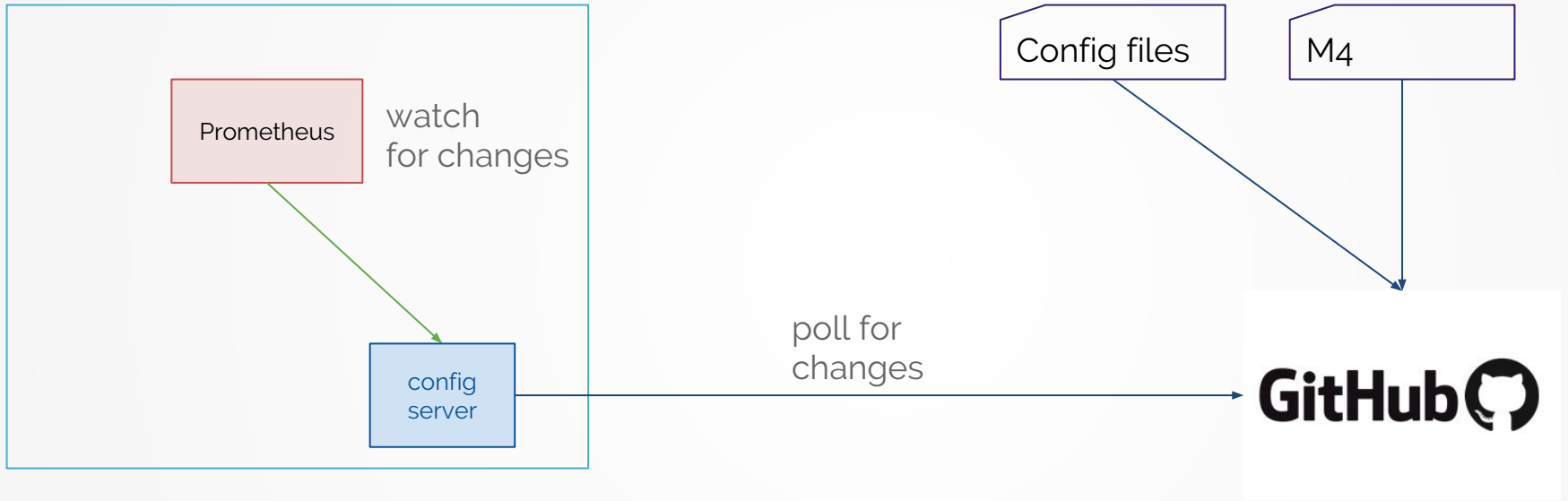
Monitoring from 10,000ft



How do mons actually work?

- All configs are stored in remote Git repos and pulled on start-up
  - Monitoring targets
  - Rules
  - Alerts
- Separate task monitors the repo for changes and reloads Prometheus
- Recording rules and M4 magic to be covered later

Configuration loading for Prometheus



Recording rules

- Precompute frequently used or expensive expressions
- E.g.
  - `job:http_requests:sum = sum(http_requests) by (job)`
- Also scraped during federation

Federation

- Scrapes a subset of time series data from another prometheus server
- Done via making a request to **`/federate`** with some **`match[]`**
- We have convention to have `::` for federated metrics

Why we need federation

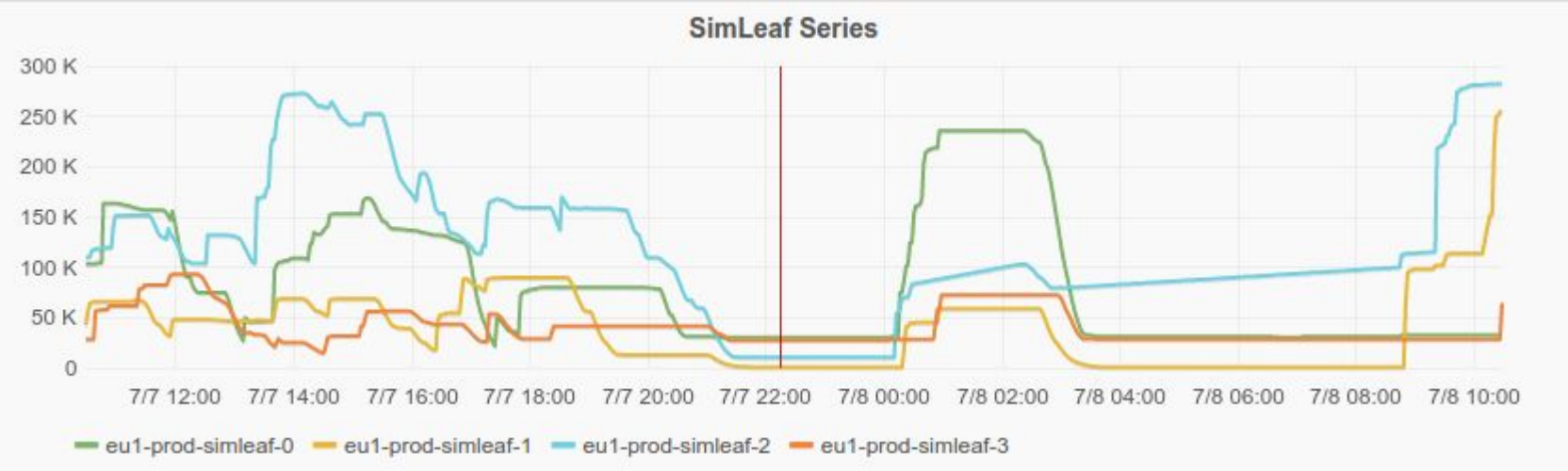
- About 4m unique time series
- **`player_latency{deploymentName, projectName, deploymentTag, nodeId}`**

SimMons

- Dynamic simleaf discovery via dns
- Naming convention to select federated metrics
- Data retention 14 days

SimLeaves

- Scrape deployment metrics
- Data sharded based on the deployment name
- Short data retention



Example of sharding

- How we shard config

```
relabel_configs:
- source_labels: [dpl,prj]
  modulus:       4
  target_label:  __hash
  action:        hashmod
- source_labels: [__hash]
  regex:         ^1$
  action:        keep
```

Can not rely on DNS

- DNS for service discovery
- Extract from a ticket tracking metrics outage
  - We ran out of DNS response size today with 500 machines
- Prometheus has a nice blog post describing how to solve this
  - http://prometheus.io/blog/2015/06/01/advanced-service-discovery/#custom-service-discovery
- We read relevant etcd entries and export it into a file that is read by Prometheus

Life of a metric

- Metric on the simulation endpoint
  - `improbable_migrations_finished{project=”user”, dpl=”dpl”, ...} 0 2`
- Recording rule on simleaf
  - `improbable_migrations_finished::sum{} = sum(improbable_migrations_finished) BY (project, dpl)`
- Actual metric on the simleaf
  - `improbable_migrations_finished::sum{project="user",dpl="dpl",} 0 11`
- Simmon
  - ```
    metrics_path: '/federate'
    params:
      'match[]': ['{__name__=~".*::.*"}']
    ```


    ```

But you said it is user facing!

- Multitenancy
- Metrics are sensitive data
  - players_online
  - hours_played
  - frags
- Need data isolation
  - Users cannot see each other's metrics
  - Users cannot see internal platform metrics

AuthServer

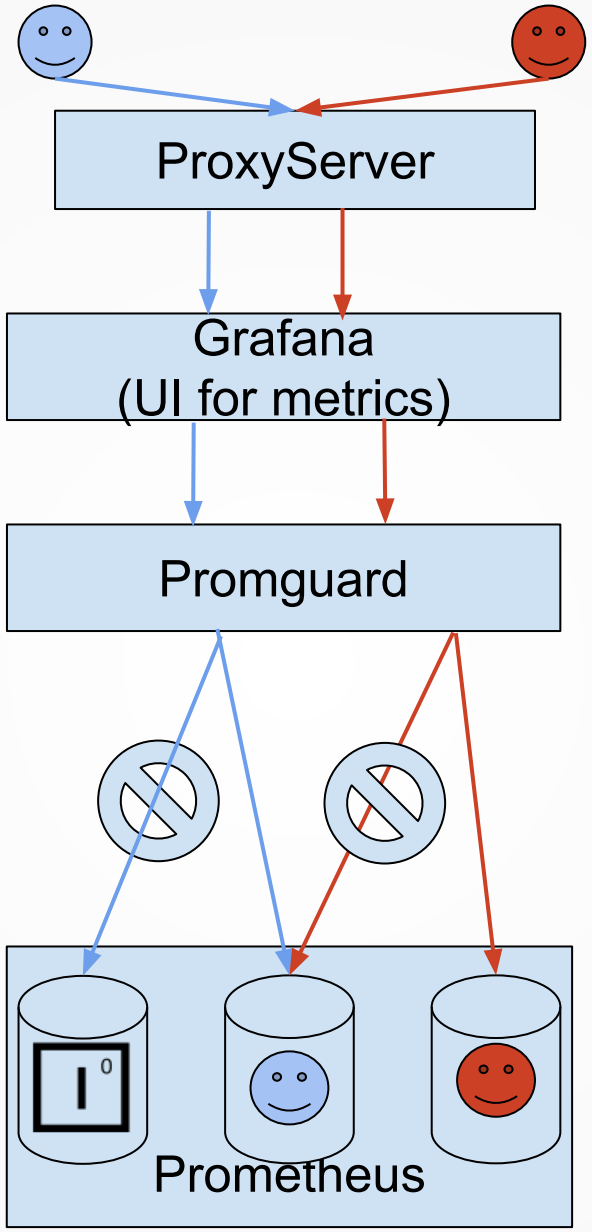
- ACL Server
- Google as identity provider
- Oauth2 flow
- JWT token contains user's permissions e.g.
  - `Read anything in project best_game_ever`

ProxyServer

- Proxy like no other
- Access to our internal services
  - Used to access Grafana for user's simulation metrics
- Uses AuthServer
- Throttling

PromGuard

- Another proxy
- Inspects the prometheus requests
- Understands the semantics of the request
- Checks permissions for this particular metric


ProxyServer
Grafana
(UI for metrics)
Promguard
Prometheus

Actual rewriting

- Prometheus endpoints
  - `/api/v1/query`
  - `/api/v1/query_range`
  - `/api/v1/series`
- Query
  - `players_lagging_total{prj=”secret_project”}`
- Parse with promql
- Based on project
- Check permissions

Example 1

- Request
  - `splines_reticulated{prj=”game”}`
- Permissions
  - `Read anything in project game`
- Result
  - `splines_reticualted{prj=”game”}`

Example 2

- Request
  - `splines_reticulated{prj=~”g.*”}`
- Permissions
  - `Read anything in project ga`
  - `Read anything in project gb`
  - `Read anything in project aa`
- Result
  - `splines_reticualted{prj=~”ga|gb”}`

Example 3

- Request
  - `splines_reticulated{prj=~”g.*”}`
- Permissions
  - `Read anything in project aa`
  - `Read anything in project ab`
- Result
  - permission denied!

Part 2

PROBLEM

- Thousands of recording rules needed to support federation
- Recording rule format isn't exactly intuitive for SpatialOS end-users
- Small set of patterns that almost all rules follow

SOLUTION

Some sort of templating system

SPEC (Gauges)

```
std_gauge(splines_reticulated, labels...) →

splines_reticulated::avg = avg (splines_reticulated) by (project, dpl, labels...)
splines_reticulated::sum = sum (splines_reticulated) by (project, dpl, labels...)
splines_reticulated::min = min (splines_reticulated) by (project, dpl, labels...)
splines_reticulated::max = max (splines_reticulated) by (project, dpl, labels...)
```

SPEC (Counters)

```
std_counter(noobs_fragged, labels...) →

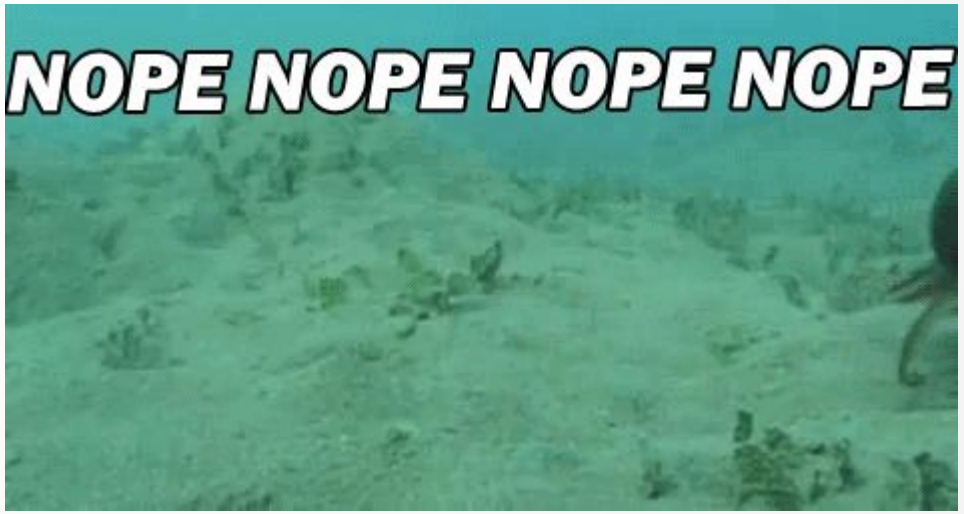
noobs_fragged:rate1m = rate (noobs_fragged[1m])
noobs_fragged::rate1m_avg = avg (noobs_fragged:rate1m) by (project, dpl, labels...)
noobs_fragged::rate1m_sum = sum (noobs_fragged:rate1m) by (project, dpl, labels...)
noobs_fragged::rate1m_min = min (noobs_fragged:rate1m) by (project, dpl, labels...)
noobs_fragged::rate1m_max = max (noobs_fragged:rate1m) by (project, dpl, labels...)
```

DISCLAIMER

- We implemented this in m4
- Rule configuration could be exposed to customers as a web service

IMPLEMENTATION

```
divert(-1)dnl
define(std_labels, `project,dpl')
define(strip, `patsubst(`$*', `,$')')
define(internal_aggregation, `$1 = $2 ($3) by (strip(std_labels,$4))')
define(internal_rate, $1 = rate($3[$2]))
define(internal_increase, $1 = increase($3[$2]))
define(internal_once,
`ifdef($1:internal_defined, `',
`$2'
)dnl
pushdef($1:internal_defined, `')')
define(internal_rate_once,
`internal_once($1:rate$2, internal_rate($1:rate$2, $2, $1))')
define(internal_increase_once,
`internal_once($1:increase$2, internal_increase($1:increase$2, $2, $1))')
define(std_gauge,
`internal_aggregation($1::avg, avg, $1, `shift($*)')
internal_aggregation($1::sum, sum, $1, `shift($*)')
internal_aggregation($1::min, min, $1, `shift($*)')
internal_aggregation($1::max, max, $1, `shift($*)')')
define(std_counter,
`internal_rate_once($1, 1m)dnl
internal_aggregation($1::rate1m_avg, avg, $1:rate1m, `shift($*)')
internal_aggregation($1::rate1m_sum, sum, $1:rate1m, `shift($*)')
internal_aggregation($1::rate1m_min, min, $1:rate1m, `shift($*)')
internal_aggregation($1::rate1m_max, max, $1:rate1m, `shift($*)')')
define(std_histogram,
`internal_increase_once($1_bucket, 5m)dnl
internal_increase_once($1_sum, 5m)dnl
internal_increase_once($1_count, 5m)dnl
internal_aggregation($1_bucket::increase5m_sum, sum, $1_bucket:increase5m, `le,shift($*)')
internal_aggregation($1_sum::increase5m_sum, sum, $1_sum:increase5m, `shift($*)')
internal_aggregation($1_count::increase5m_sum, sum, $1_count:increase5m, `shift($*)')')
divert(0)dnl
```



Questions?