Metrics for the win!

Using Micrometer to Understand Application Behavior

Erin Schnabel
@ebullientworks

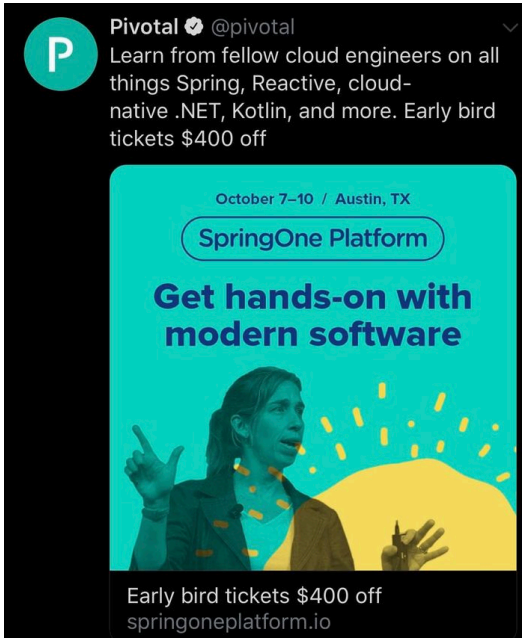
IBM Developer

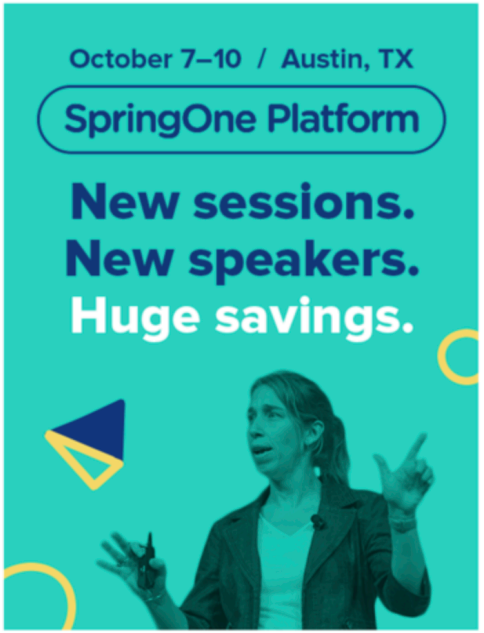
IBM

You may have seen me around..

Main stage with Cornelia in 2017!

Motivator to attend in 2019?

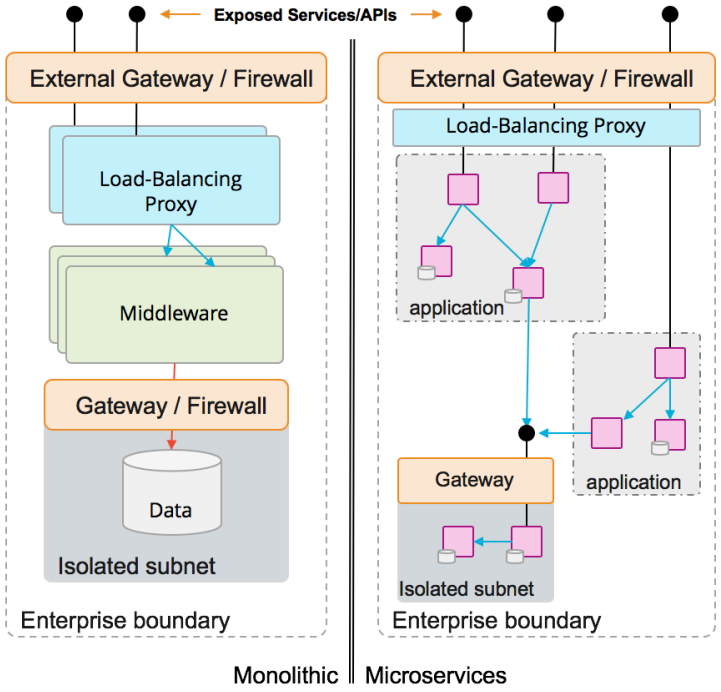


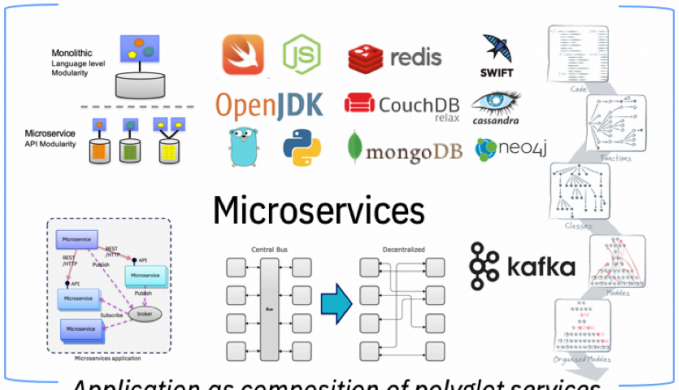


<backstory>

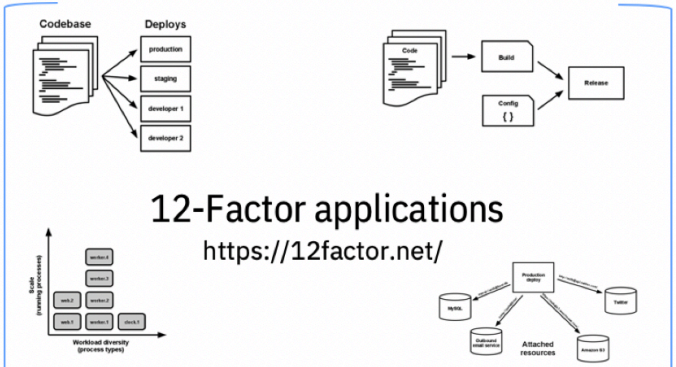
Microservices & Orchestration

Monolith -> Microservices





Application as composition of polyglot services



Independent, autonomous, stateless processes

Orchestrating all the things...

Cloud Environment

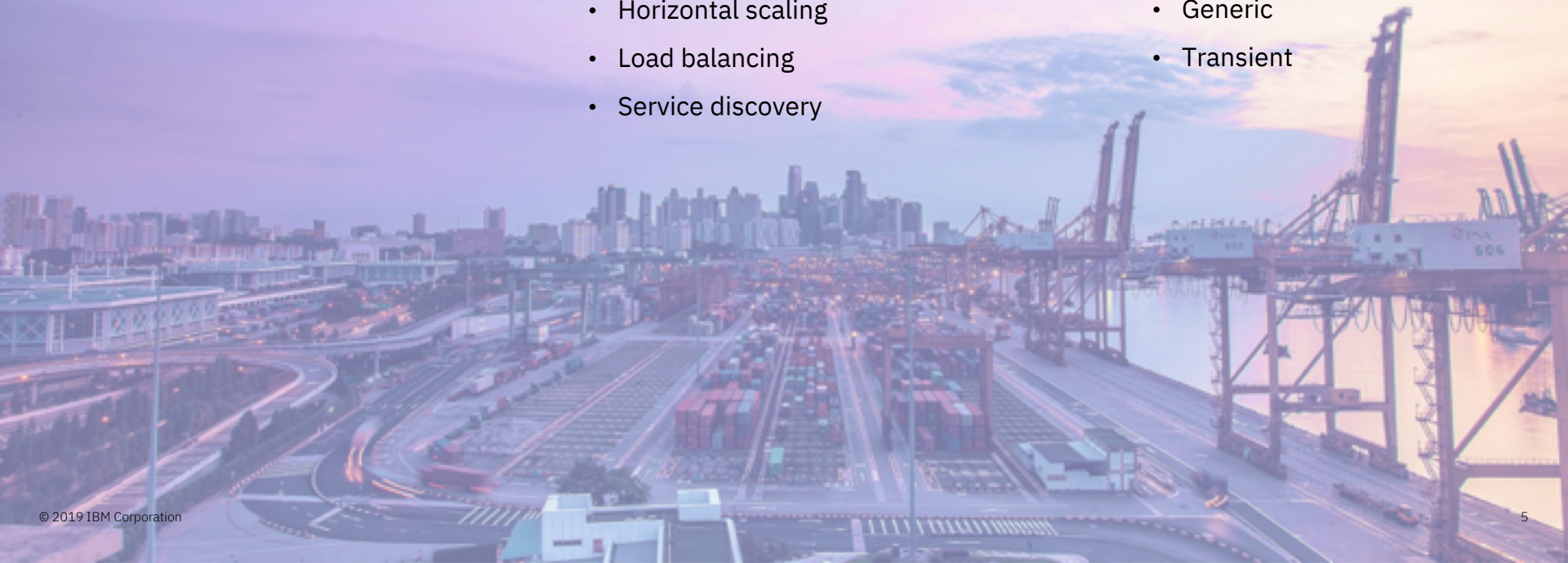
- Dynamic infrastructure
- Flexible capacity

Container orchestration:

- System health (self-healing)
- Workload scheduling
- Horizontal scaling
- Load balancing
- Service discovery

Containerized workloads

- *Simplified operations*
- Resource isolation
- Generic
- Transient



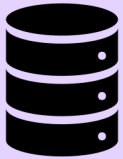
Orchestration requires Observability

A system is observable if you can figure out WTF is going on without redeploying to add more trace.

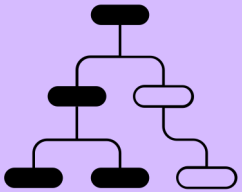
Health Checks

Metrics

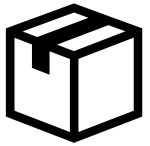
Log Entries



Distributed Trace



Infrastructure



Default probe behaviors

Metrics collection
Resource utilization metrics

Log collection

Span creation & propagation

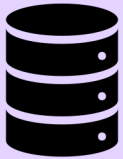
Health Checks

Specialized probes

Metrics

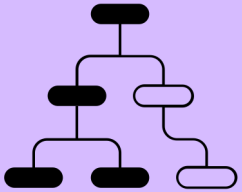
Application-centric statistical data

Log Entries



Structured or unstructured text capturing discrete events

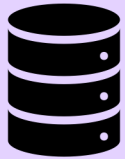
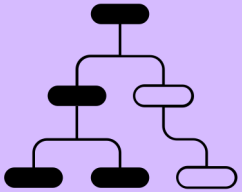
Distributed Trace



Describe relationships between spans

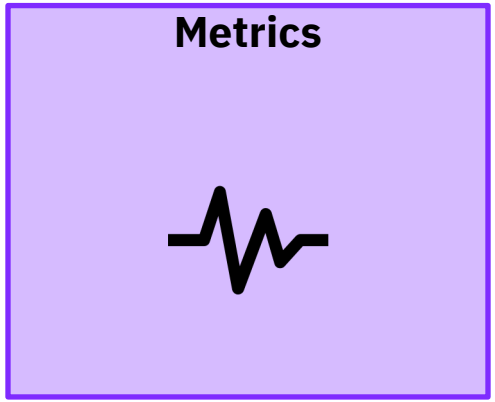
Application

Which for what?

Health Checks	Metrics	Log Entries	Distributed Trace
Service is ready Service is not a zombie	How many times was method x called?	What happened when method x was called	Method x was called
			
Workload routing System health	Statistics & trends Analytics	Service-centric problem determination	Context + relationships for end–to-end analysis

</backstory>

Monsters #FTW!



Prometheus for time series data

Grafana for visualization

Micrometer for app-centric instrumentation

MONSTERS!

```
Added Minotaur Living Crystal Statue (large construct), AC: 15, HP:
Added Minotaur Skeleton (large undead), AC: 12, HP: 67(9d10+18)
Added Miraj Vizann (medium humanoid), AC: 10, HP: 82(11d8+33)
Added Miros Xelbrin (medium humanoid), AC: 10, HP: 22(4d8+4)
Added Mirt (medium humanoid), AC: 16, HP: 153(18d8 + 72)
Added Moloch (large fiend), AC: 19, HP: 253(22d10+132)
Added Molydeus (huge fiend), AC: 19, HP: 216(16d12+112)
Added Mongrelfolk (medium humanoid), AC: 11, HP: 26(4d8+8)
Added Monodrone (medium construct), AC: 15, HP: 5(1d8+1)
Added Monstrous Peryton (large monstrosity), AC: 14, HP: 144(17d10 +
Added Morkoth (medium aberration), AC: 17, HP: 130(20d8+40)
Added Mormesk the Wraith (medium undead), AC: 13, HP: 45(6d8+18)
Added Mouth of Grolantor (huge giant), AC: 14, HP: 105(10d12+40)
Added Mr. Dory (medium aberration), AC: 18, HP: 170(20d8 + 80)
Added Mud Mephit (small elemental), AC: 11, HP: 27(6d6+6)
Added Muiral (large monstrosity), AC: 16, HP: 195(23d10 + 69)
Added Mule (medium beast), AC: 10, HP: 11(2d8+2)
Added Mummy (medium undead), AC: 11, HP: 58(9d8+18)
Added Mummy Lord (medium undead), AC: 17, HP: 97(13d8+39)
Added Mwaxanaré (medium humanoid), AC: 10, HP: 13(3d8)
Added Myconid Adult (medium plant), AC: 12, HP: 22(4d8+4)
Added Myconid Sovereign (large plant), AC: 13, HP: 60(8d10+16)
Added Myconid Sprout (small plant), AC: 10, HP: 7(2d6)
Added NImblewright (medium construct), AC: 18, HP: 45(6d8 + 18)
Added Nabassu (medium fiend), AC: 18, HP: 190(20d8+100)
Added Naergoth Bladelord (medium undead), AC: 18, HP: 135(18d8+54)
Added Nagpa (medium humanoid), AC: 19, HP: 187(34d8+34)
Added Nalfeshnee (large fiend), AC: 18, HP: 184(16d10+96)
Added Narrak (small humanoid), AC: 12, HP: 40(9d6+9)
Added Narth Tezrin (medium humanoid), AC: 12, HP: 18(4d8)
Added Narzugon (medium fiend), AC: 20, HP: 112(15d8+45)
Added Naxene Drathkala (medium humanoid), AC: 10, HP: 27(6d8)
Added Necromancer (medium humanoid), AC: 12, HP: 66(12d8+12)
Added Needle Blight (medium plant), AC: 12, HP: 11(2d8+2)
Added Neogi (small aberration), AC: 15, HP: 33(6d6+12)
Added Neogi Hatchling (tiny aberration), AC: 11, HP: 7(3d4)
Added Neogi Master (medium aberration), AC: 15, HP: 71(13d6+26)
Added Neothelid (gargantuan aberration), AC: 16, HP: 325(21d20+105)
Added Nereid (medium fey), AC: 13, HP: 44(8d8+8)
Added Neronvain (medium humanoid), AC: 17, HP: 117(18d8+36)
Added Nezznar the Black Spider (medium humanoid), AC: 11, HP: 27(6d8
Added Night Hag (medium fiend), AC: 17, HP: 112(15d8+45)
Added Nightmare (large fiend), AC: 13, HP: 68(8d10+24)
Added Nightveil Specter (medium undead), AC: 17, HP: 105(14d8 + 42)
```

Monsters & Highlander (there can be only one)

Dungeons & Dragons inspired combat

1. Ingest XML document → **1063 usable monsters**

2. **Monsters** engage in **battles**:
   - /battle/**faceoff** – 2 opponents
   - /battle/**melee** – 3-5 opponents

3. Battle has **stages**:
   1. Check for **surprise**: Dexterity vs. Perception
   2. Roll for **initiative**
   3. Repeat **rounds** (in initiative order) until only one left

Monster has **Type**, **Size**, **Armor Class**, **Hit Points**:

Star Spawn Grue (**small aberration**), AC: 11, HP: 17(5d6)
Star Spawn Hulk (large aberration), **AC: 16**, HP: 136(13d10+65)
Star Spawn Larva Mage (medium aberration), AC: 16, HP: 168(16d8+96)
Star Spawn Mangler (medium aberration), AC: 14, **HP: 71(13d8+13)**
Star Spawn Seer (medium aberration), AC: 17, HP: 153(18d8+72)

Weapons inflict variable **damage**:

bite:	Hit: 3,	Damage: 2d6+3+1d6
claw:	Hit: 3,	Damage: 3d6+5
tail:	Hit: 3,	Damage: 2d4+3
acid spray:	Hit: 0,	Damage: 3d6

NUMBERS!

```
# HELP monster_rounds_survived_total
# TYPE monster_rounds_survived_total counter
monster_rounds_survived_total{size="huge",type="construct",} 9.0
monster_rounds_survived_total{size="small",type="plant",} 1.0
monster_rounds_survived_total{size="small",type="elemental",} 8.0
monster_rounds_survived_total{size="medium",type="fiend",} 17.0
monster_rounds_survived_total{size="tiny",type="undead",} 1.0
monster_rounds_survived_total{size="tiny",type="beast",} 3.0
monster_rounds_survived_total{size="gargantuan",type="ooze",} 7.0
monster_rounds_survived_total{size="huge",type="beast",} 6.0
monster_rounds_survived_total{size="medium",type="undead",} 32.0
monster_rounds_survived_total{size="large",type="giant",} 10.0
monster_rounds_survived_total{size="tiny",type="aberration",} 3.0
monster_rounds_survived_total{size="large",type="elemental",} 4.0
monster_rounds_survived_total{size="large",type="aberration",} 8.0
monster_rounds_survived_total{size="huge",type="undead",} 7.0
monster_rounds_survived_total{size="large",type="monstrosity",} 5.0
monster_rounds_survived_total{size="large",type="beast",} 4.0
monster_rounds_survived_total{size="medium",type="construct",} 10.0
monster_rounds_survived_total{size="medium",type="beast",} 4.0
monster_rounds_survived_total{size="medium",type="aberration",} 4.0
monster_rounds_survived_total{size="medium",type="monstrosity",} 16.0
monster_rounds_survived_total{size="small",type="construct",} 3.0
monster_rounds_survived_total{size="medium",type="humanoid",} 30.0
monster_rounds_survived_total{size="large",type="fiend",} 11.0
monster_rounds_survived_total{size="small",type="humanoid",} 17.0
# HELP monster_surprised_total
# TYPE monster_surprised_total counter
monster_surprised_total{size="huge",type="construct",} 9.0
monster_surprised_total{size="small",type="plant",} 2.0
monster_surprised_total{size="large",type="plant",} 1.0
monster_surprised_total{size="medium",type="fiend",} 21.0
monster_surprised_total{size="large",type="aberration",} 9.0
monster_surprised_total{size="large",type="monstrosity",} 6.0
monster_surprised_total{size="huge",type="undead",} 5.0
monster_surprised_total{size="tiny",type="beast",} 6.0
monster_surprised_total{size="large",type="beast",} 5.0
monster_surprised_total{size="gargantuan",type="ooze",} 7.0
```

Time-series data

String key with ***ONE**** **numeric value**

Value is ***observed*** at collection time

Periodic collection over time → series
Each value ***appended*** as new entry

Time is a primary axis (x)

Useful for spotting:

- Trends
- Variability
- Rate of change

* “Univariate”. Keeps things from getting out of hand.

Dimensional data

Consider **hierarchical** data (our old friend):

```
jvm.memory.used
```

Add datacenter, and instance id:

```
datacenter.instance_id.jvm.memory.used
*.instance_id.jvm.memory.used
*.*.jvm.memory.used
```

What if you want to add

- environment (dev, test, prod)
- service name

Tags / Labels add **dimensions** for analysis:

```
jvm.memory.used {
    datacenter=us1
    instance_id=pod1234
    env=prod
    service=serviceName
}
```

Filtered aggregation of **single numeric value**

Not constrained by wildcards / hierarchy

Cardinality explosion

New series for each unique combination

Label **A (10 values),** Label **B (100 values)**,

Add **pod (p)**, **instance (i)**, **service (s)**

(10 * 100 * p * i * s) unique time series!

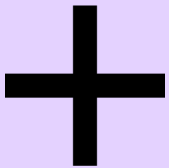
Real (bad) examples:

- 404 URLs
- request ids
- user ids

→ Normalize and bound dimensions

```
attack="claw”
attack="claws”
attack="claw/bite”

attack="dagger"
attack="infernal dagger”
attack="mind-poison dagger"
```

Meter types

Incrementing value	Observed value	Distribution of values (buckets)	Bucket x Timer
Counter	**Gauge**	**Histogram**	**Summary**
			 
`Counter`	`Gauge`	`DistributionSummary`	`Timer`
		count, total, max Optional: percentiles, SLAs, buckets	Duration (total time) AND Distribution summary

Micrometer

EXAMPLES

From here on out: Focused on *custom* application metrics

- Examples to show what values are produced
- Looking for information that is interesting *to me*

Also, Winning!

- *My goals:*
  - Understand how D&D battles work
  - Get a feeling for the scale factor behind the stats

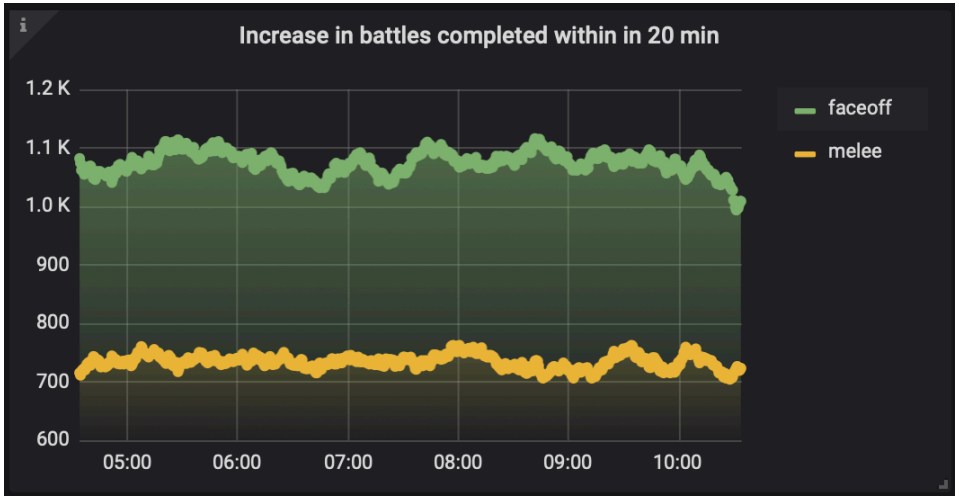
Counter: Activity

How many battles have been fought?

```
registry.counter("battles.completed", "type", "faceoff")
    .increment();
registry.counter("battles.completed", "type", ”melee")
    .increment();
```

Prometheus

- `rate(battles_completed_total[20m])`
  average rate of increase within a 20 minute window
- `increase(battles_completed_total[20m])`
  human readable: rate * seconds_in_interval

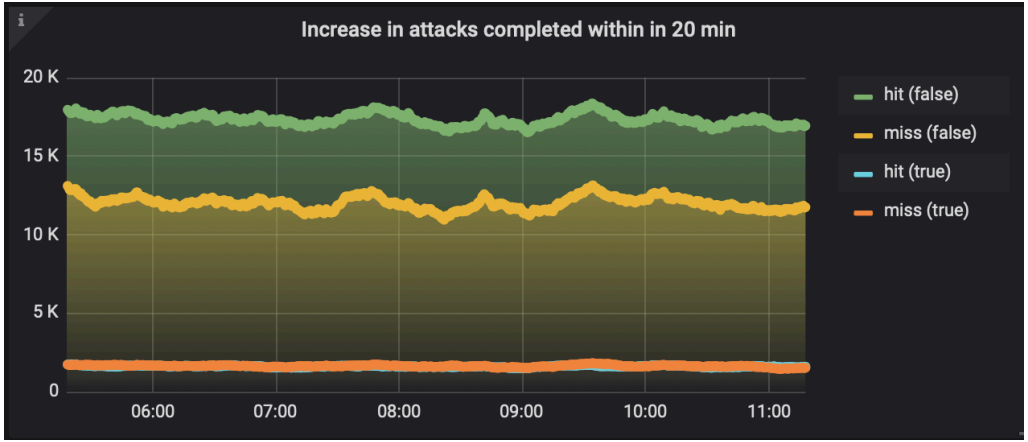



```
# HELP battles_completed_total
# TYPE battles_completed_total counter
battles_completed_total{type="faceoff",} 218.0
battles_completed_total{type="melee",} 153.0
```

Counter: Attack!

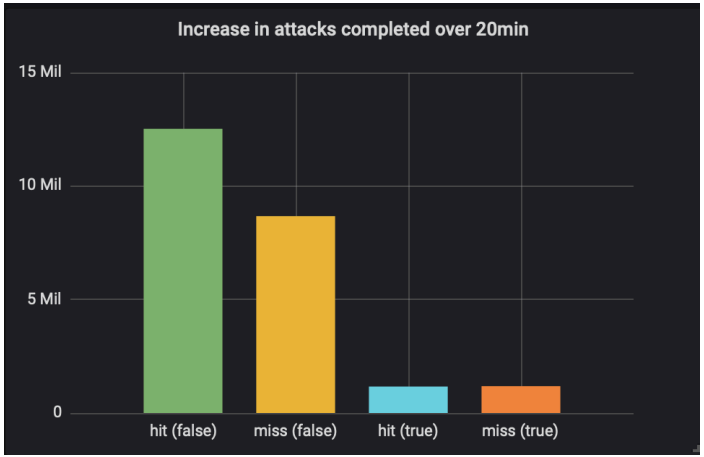
How many monster attacks?

```
registry.counter("monster.attacks",
        "type", r.hitOrMiss(),
        "critical", r.critical())
    .increment();
```



Prometheus

```
increase(battles_completed_total[20m])
```




```
# HELP monster_attacks_total
# TYPE monster_attacks_total counter
monster_attacks_total{critical="false",type="miss",} 2284.0
monster_attacks_total{critical="true",type="miss",} 306.0
monster_attacks_total{critical="false",type="hit",} 3254.0
monster_attacks_total{critical="true",type="hit",} 312.0
```

Counter: Survival

How many rounds did monsters survive?

```
registry.counter("monster.rounds.survived",
        "type", p.getType(),
        "size", p.getSize())
    .increment();
```

```
registry.counter("monster.rounds.survived.individual",
        "name", p.getName())
    .increment();
```

Top 5 surviving types/size

Metric	Total
medium humanoid	408.12 K
large monstrosity	72.69 K
medium undead	72.31 K
medium fiend	55.86 K
medium monstrosity	48.35 K

```
topk(5, sum(monster_rounds_survived_total) by (type, size))
```

Top 5 Survivors (Individual)

Metric	Current
Meloon Wardragon	3.36 K
The Angry	3.29 K
Manshoon	3.21 K
Winter Eladrin	3.14 K
Ultroloth	2.90 K

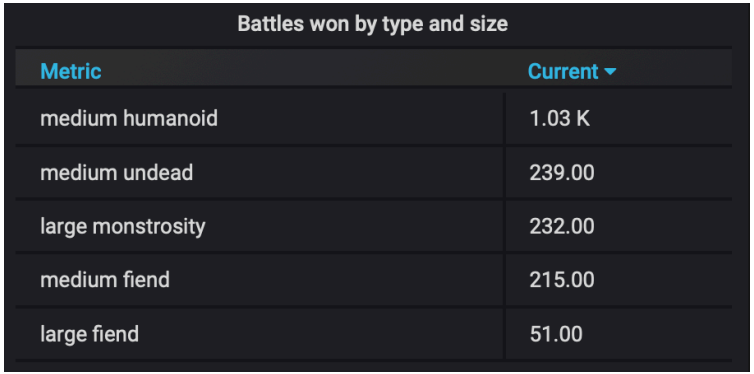
```
topk(5, monster_rounds_survived_individual_total)
```

Counter: Winning!

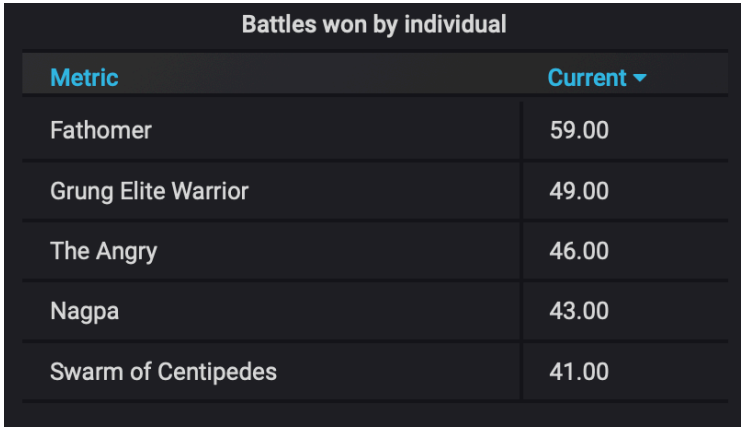
How often did Monsters win?

```
registry.counter("monster.rounds.won",
        "type", p.getType(),
        "size", p.getSize())
    .increment();
```

```
registry.counter("monster.rounds.won.individual",
        "name", p.getName())
    .increment();
```



`topk(4, sum(monster_rounds_won_total) by (type, size))`

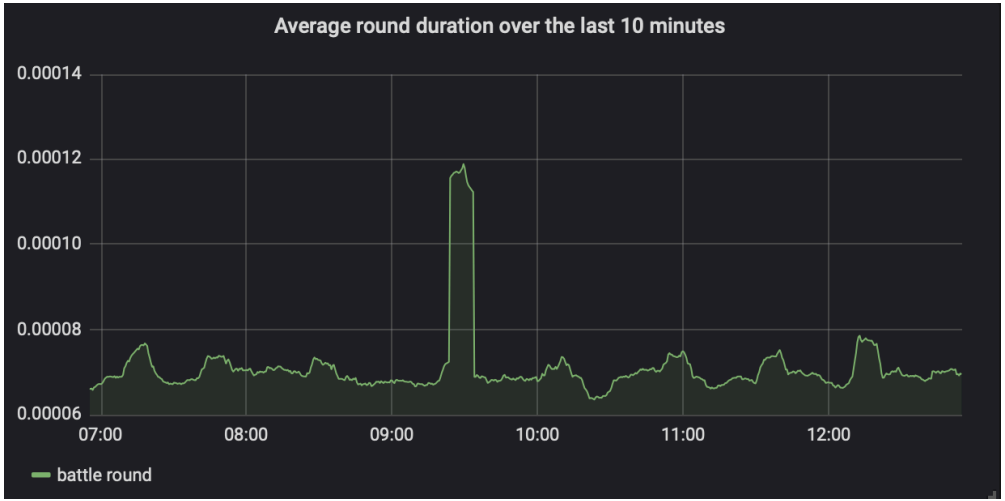


`topk(5, monster_rounds_survived_individual_total)`

Timers: Rounds

How long does a round take?

```
roundDuration =
  Timer.builder("battle.rounds.duration")
    .minimumExpectedValue(Duration.ofMillis(1))
    .maximumExpectedValue(Duration.ofSeconds(3))
    .register(registry);
```



Will vary by number of monsters in the round,

AND..

There may be anomalies!

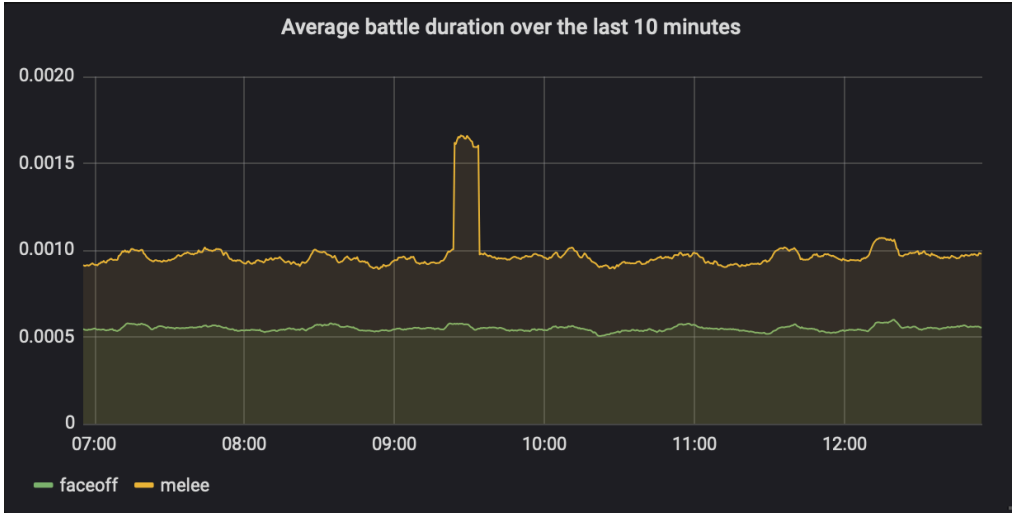
```
# HELP battle_rounds_duration_seconds_max
# TYPE battle_rounds_duration_seconds_max gauge
battle_rounds_duration_seconds_max 0.001736238

# HELP battle_rounds_duration_seconds
# TYPE battle_rounds_duration_seconds summary
battle_rounds_duration_seconds_count 3727.0
battle_rounds_duration_seconds_sum 1.519626711
```

```
rate(battle_rounds_duration_seconds_sum[10m]) / rate(battle_rounds_duration_seconds_count[10m])
```

Timers: Battles

How long does a battle take?

```
faceoffDuration =
  Timer.builder("battles.duration")
    .tag("type", "faceoff")
    .minimumExpectedValue(Duration.ofMillis(1))
    .maximumExpectedValue(Duration.ofSeconds(30))
    .register(registry);

meleeDuration =
  Timer.builder("battles.duration")
    .tag("type", ”melee")
    .minimumExpectedValue(Duration.ofMillis(1))
    .maximumExpectedValue(Duration.ofSeconds(30))
    .register(registry);
```



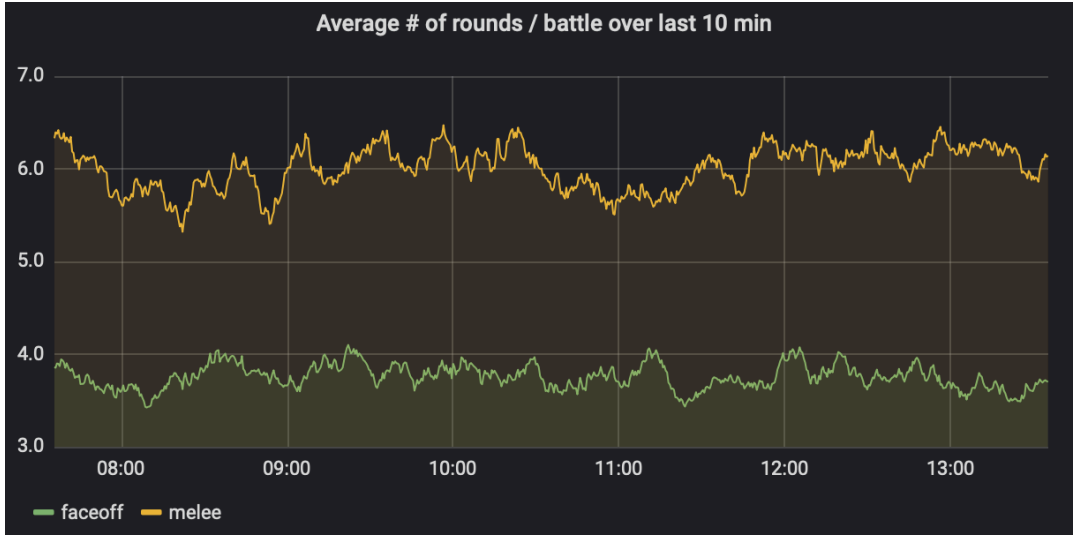

```
# HELP battles_duration_seconds
# TYPE battles_duration_seconds summary
battles_duration_seconds_count{type="faceoff",} 396.0
battles_duration_seconds_sum{type="faceoff",} 1.190387904
battles_duration_seconds_count{type="melee",} 274.0
battles_duration_seconds_sum{type="melee",} 1.040496664

# HELP battles_duration_seconds_max
# TYPE battles_duration_seconds_max gauge
battles_duration_seconds_max{type="faceoff",} 0.00839499
battles_duration_seconds_max{type="melee",} 0.004203886
```

```
rate(battles_duration_seconds_sum[10m]) / rate(battles_duration_seconds_count[10m])
```

Distribution Summary: Battle progress

How many rounds in each battle?

```
numberOfFaceOffRounds =
  DistributionSummary.builder("battle.rounds")
      .tag("type", "faceoff")
      .minimumExpectedValue((long) 1)
      .maximumExpectedValue((long) 15)
      .register(registry);
```




```
numberOfFaceOffRounds.record((double) finalRound.getNumber());
```

** repeat for melee

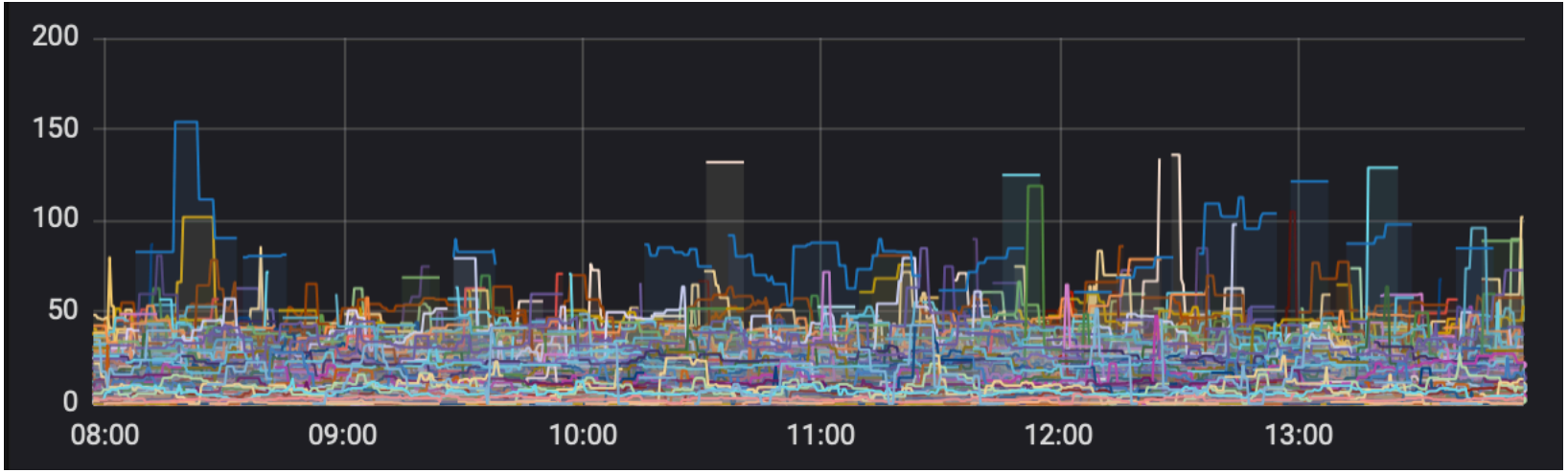

```
rate(battle_rounds_sum[10m]) / rate(battle_rounds_count[10m])
```

Distribution Summary: Lethal weapons

```
registry.summary("weapon.attack.damage",
    "type", r.getAttackName())
  .record((double) r.getDamage());
```

Most Lethal Weapons

Metric	Avg ▾
warleader's helix	84.81
quivering palm	47.66
horrid touch	42.77
extract brain	40.69
disruptive touch	39.14

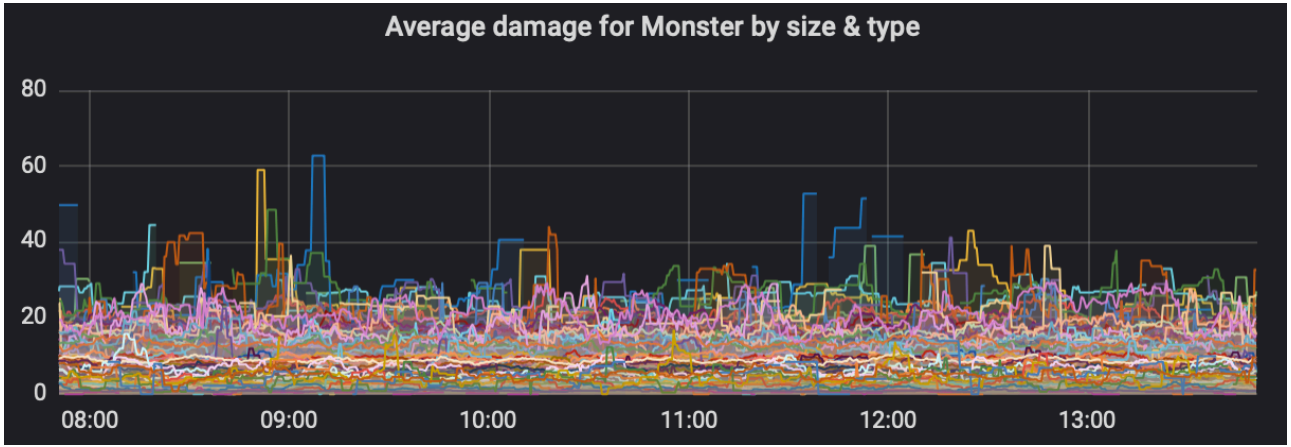


Distribution Summary: Lethal monsters

```
registry.summary("monster.attack.damage",
        "type", attacker.getType(),
        "size", attacker.getSize())
    .record((double) r.getDamage());
```

Most Lethal Monsters

Metric	Avg ▾
gargantuan fiend	26.22
gargantuan construct	24.60
huge aberration	23.86
gargantuan ooze	22.86
gargantuan beast	21.89




```
rate(monster_attack_damage_sum[10m]) / rate(monster_attack_damage_count[10m])
topk(5, monster_attack_damage_sum / monster_attack_damage_count )
```

Not much about micrometer, is there?

Micrometer makes it easy to collect the data you need

Actuator turns on endpoints:

/actuator/Prometheus

/actuator/metrics

All of the metrics gathered are visible using either endpoint

Asking the right questions is important.

What do you need to know?

Thank you!

@ebullientworks

https://github.com/ebullient/monster-combat