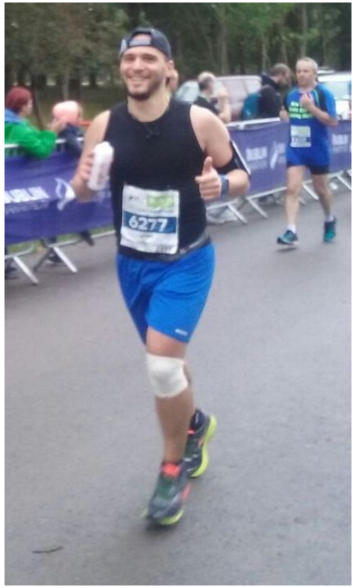
# Play with Prometheus

Journey to make “testing in production” more reliable

**Giovanni Gargiulo - HBC Digital - Promcon @ Munich 2017**

## About me...



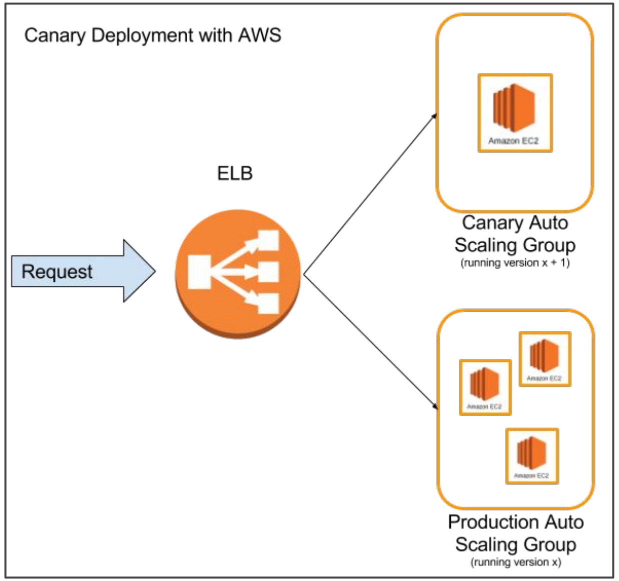
- Software Engineer
- 12 years on JVM languages
- Gilt Personalization team since 2015
- @giannigar
- On github: nemo83

## Brief history of Gilt.com

- Gilt is a high end fashion online retailer
- Business model: flash sales
- Launched in 2007 as monolithic Rails app
- In 2010 journey to break the monolith: ~10 Java services
- Today 350+ (mostly scala) micro services
- Gilt joined HBC in early 2016

# Development process

- Short iterations and CD/CI
- No testers
- ~~Integration~~ Testing in production
- Canary and Production deployment



## Release checklist

*"... it works in dev (i.e. Dark Canary), but will it work live?..."*

- ❏ Smoke test
- ❏ RPM
- ❏ Response time
- ❏ Errors

## Operations in Personalization 2016

**Monitoring:**

- Vanilla New Relic
- Cloudwatch (CPU usage)
- Custom AWS Lambda functions (deployment notifications)

**Alerting:**

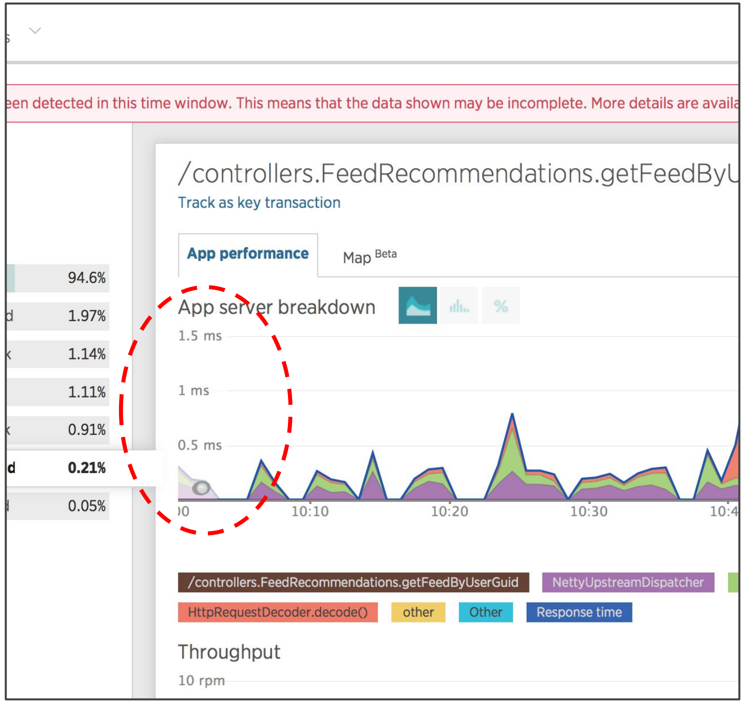
- PagerDuty via New Relic + Cloudwatch

## Some limitations

With the tools at hand:

- Custom metrics and dashboards not user friendly
- Unreliable alerting (false positive / negatives)
- No Single Place for all alerts
- Copy and paste same alerts everywhere: DRY
- Straw that broke the camel’s back: NR’s fails to trace Scala Futures

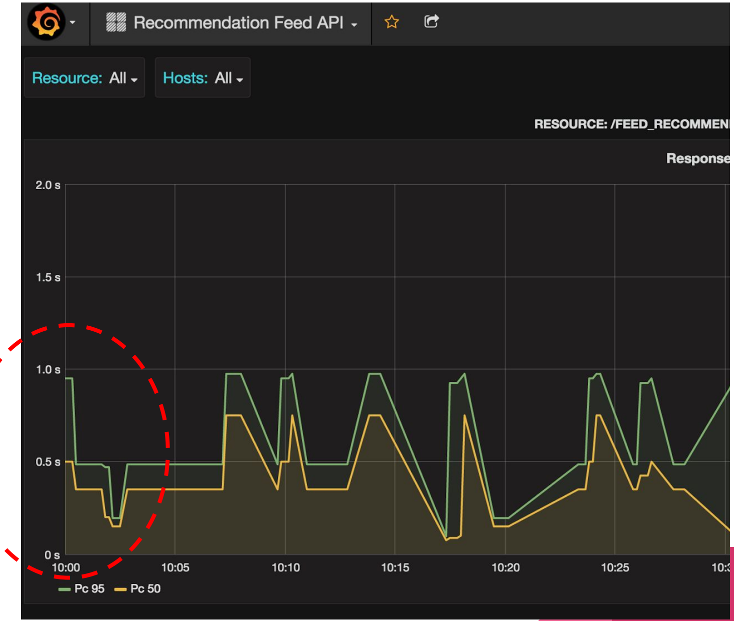
# New Relic async reporting issue



een detected in this time window. This means that the data shown may be incomplete. More details are availa

/controllers.FeedRecommendations.getFeedByU

Track as key transaction



## We needed something new!

Key things that drove our decision:

- Designed for Time Series
- Scalable (thousands of hosts)
- Percentiles and derived metrics
- User friendly, reusable and customisable dashboards

## Solution

**Prometheus + Grafana**

Prometheus: is an open-source systems monitoring and alerting toolkit originally built at SoundCloud.

Grafana: provides a powerful and elegant way to create, explore, and share dashboards and data with your team and the world.

## The plan

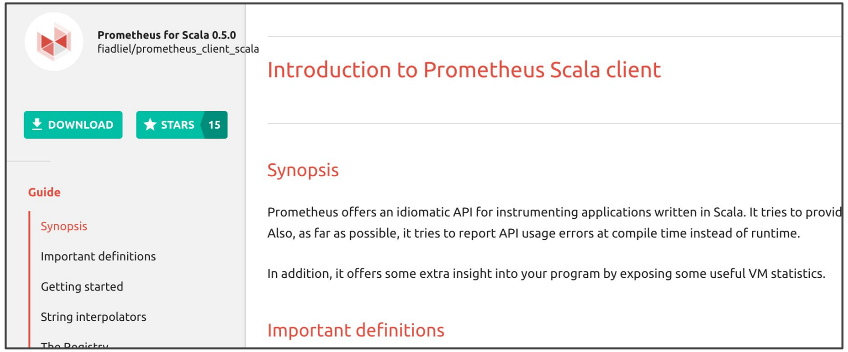
1. Evaluate the Prometheus suite and Grafana in the Personalization team
2. Create reusable templates
3. Other teams to adopt
4. Create Prometheus Hierarchical Federation + centralised Grafana

## Code instrumentation

- No official Prometheus Scala client
- Awkward to use the Java lib to instrument Scala code
- Pimp my library pattern

# The Prometheus Scala client

- Open Source
- Github: https://github.com/fiadliel/prometheus_client_scala
- Extended guide: https://www.lyranthe.org/prometheus_client_scala/guide/



Introduction to Prometheus Scala client

Synopsis

Prometheus offers an idiomatic API for instrumenting applications written in Scala. It tries to provid
Also, as far as possible, it tries to report API usage errors at compile time instead of runtime.

In addition, it offers some extra insight into your program by exposing some useful VM statistics.

Important definitions

## Take away #1

Instrumenting your code is powerful but:

- It could lead to tons of boilerplate and repeated code
- It's frustrating and error prone

**Solution**: provide out of the box instrumentation to most common scala frameworks. E.g: Playframework, akka-http, http4s

## Play instrumentation #1

Instrumenting the JVM in a Scala Play application

PrometheusJmxInstrumentation.scala

```
import com.google.inject.{Inject, Singleton}
import org.lyranthe.prometheus.client._

@Singleton
class PrometheusJmxInstrumentation @Inject()()(implicit registry: Registry) {
  jmx.register()
}
```

# Play instrumentation #2

Instrumenting ReST endpoints in a Scala Play application

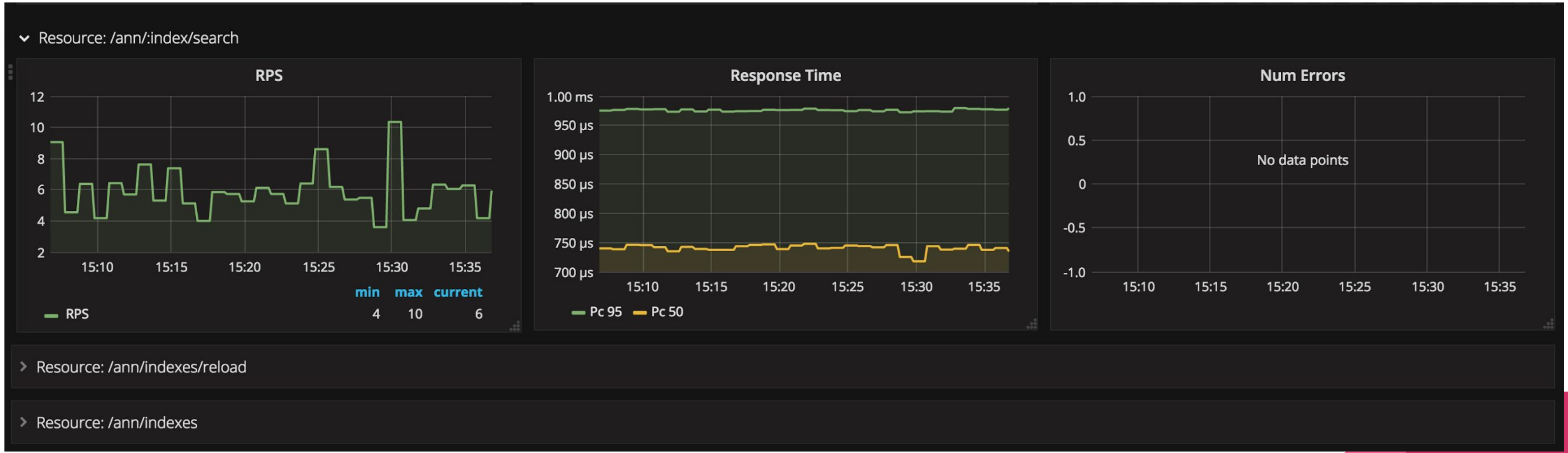
Filters.scala

```
import com.google.inject.{Inject, Singleton}
import org.lyranthe.prometheus.client._

class Filters @Inject()(prometheusFilter: PrometheusFilter) extends HttpFilters {
   val filters = Seq(prometheusFilter)
}
```

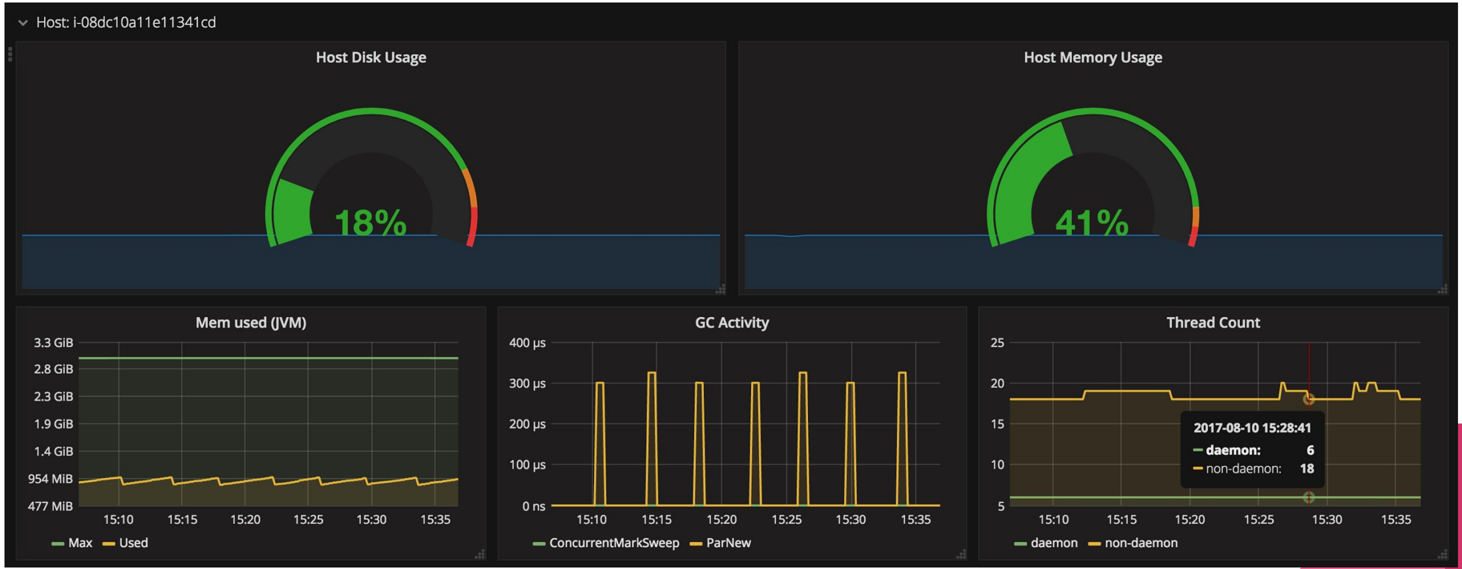
# Play instrumentation #3

Automatically create graphs leveraging Grafana template engine



# Play instrumentation #4

Automatically create graphs leveraging Grafana template engine



## Prometheus stack management

- Prometheus in AWS is not offered as-a-service
- We initially manually created the first stack
- The first time it crashed we lost data and configuration
- Difficult to be adopted by other teams

## Take away #2

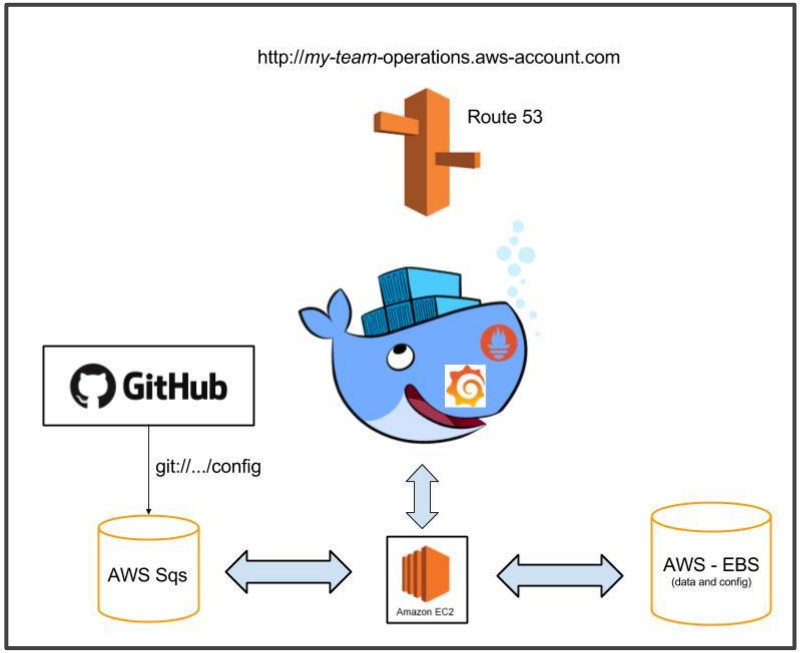
- In a DevOps team the *Ops* part needs to be simple and efficient
- Team to spend too much time supporting and maintaining Prometheus and Grafana

**Solution**: Create templates that are reusable, customizable and easy to maintain and upgrade

## Prometheus Cloudformation Template

- Monitor AWS resources
- AWS Cloudformation template
  - Describe service resources via templates
  - Can be created and destroyed quickly
- Github: https://github.com/nemo83/aws_prometheus_template

# Prom AWS Cloudformation Template



- Docker Compose to launch the Prometheus Suite
- Can be integrated with github to allow configuration versioning and automate the Prometheus configuration release
- External EBS Volume for decoupling EC2 instance lifecycle from data and configuration

## Prom AWS Cloudformation Template #3

The AWS Cloudformation template provides facility and documentation for:

- Creating and updating the cluster via cfn-init and cfn-hup
  - `make create-stack`
  - `make update-stack`
- A docker-compose file to launch the Prometheus suite and Grafana
- Automatically update the Prometheus configuration via Github and the AWS Simple Queue Service

## Prom AWS Cloudformation Template #4

It provides configuration templates and examples to get up and running quickly

prometheus.yaml

```
- job_name: unlabelled_job
  ec2_sd_configs:
    - region: us-east-1
      port: 9000
  relabel_configs:
    - source_labels: [__meta_ec2_tag_Name]
      regex: (my-cool-api)
      action: keep
    - source_labels: [__meta_ec2_instance_id]
      target_label: instance
    - source_labels: [__meta_ec2_tag_Name]
      target_label: job
    - source_labels: [__meta_ec2_tag_Environment]
      target_label: environment
```

## Nov - Dec 2016 Achievements

- Two teams adopted Prometheus and Grafana
- New beautiful user friendly dashboards
- Improved Alerting mechanism (warnings, critical)
- Scala client support for Play Framework 2.4 and 2.5
- First release of the Aws Prometheus CFN template
- $$$$ Cost savings: we were often overprovisioning

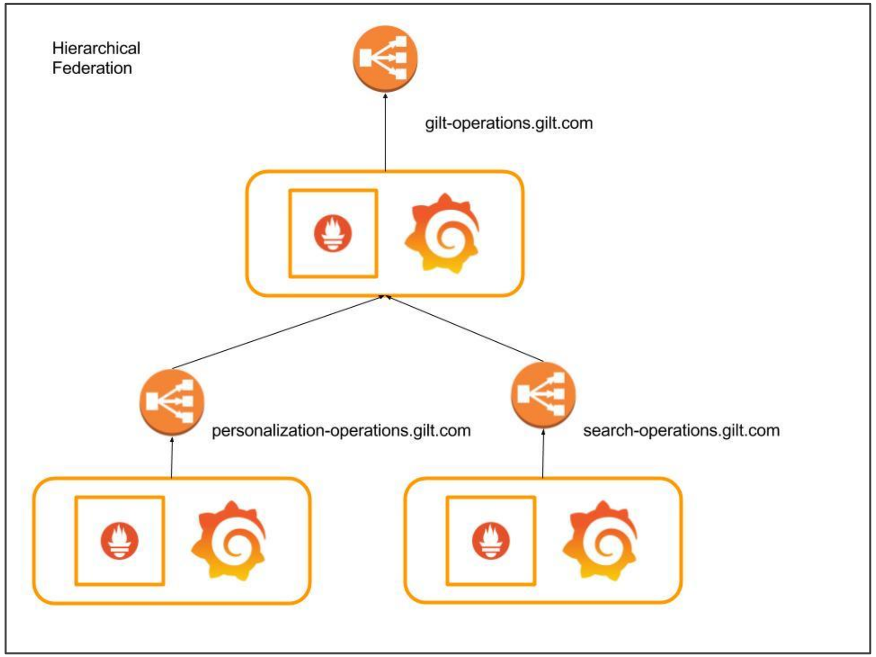
disk-space-alerts.yaml

```
# Slack Message if disk usage % greater than 80
ALERT disk_space_usage_pc_warning
 IF disk_space_usage_pc > 80
 FOR 5m
 LABELS {
   severity = "high"
 }
# Page if disk usage % greater than 90
ALERT disk_space_usage_pc_critical
 IF disk_space_usage_pc > 90
 FOR 5m
 LABELS {
   severity = "critical"
 }
```

## As of today

- Four teams have adopted Prometheus and Grafana
- 20+ Services have been migrated
- 60+ dashboards
- Scala client supports most common frameworks
- New Prometheus template and Federation

## Hierarchical Federation (take away #3)


Hierarchical Federation
gilt-operations.gilt.com
personalization-operations.gilt.com
search-operations.gilt.com

- Each team has it's own prometheus cluster
- Custom dashboards and alerts
- Subset of metrics are ingested by the generic gilt-operations cluster
- Templated dashboards are created for every service
- One stop shop to get at service health status at a glance

## What did we achieve?

- Custom dashboards give us a much more detailed picture about the health status of our services
- Optimise resource allocation
- Increased confidence during production releases
- Reliable alerting
- Overall improved customer experience

## What’s next

- Implement failover in the Cloudformation template
- Meta monitoring
- Validate Prometheus configuration with `promtool` when issuing a PR

# Thank you!

**Q&A**