
PCP ❤ Java
...
Ryan Doyle

What we'll cover

- Black-box instrumentation
- In-JVM instrumentation
- Integrations
  - Codahale metrics
  - Graphite/Grafana
- Code available at https://github.com/ryandoyle/pcpconf18-java-tooling

$ whoami

- Software/Systems engineer - mostly web applications
- Heavy *user* of PCP
- Occasional contributor
  - Java tooling
  - Golang & Ruby language bindings
  - Wireshark protocol dissector
  - PMDAs

Building on others contributions

- Most of what I'll cover others have written
  - parfait-agent
  - parfait-dropwizard
  - pcp2graphite
  - graphite/grafana
- Thanks to all that contribute


Parfait

Parfait

- Java instrumentation toolkit
- Pluggable
  - Export to PCP or JMX
  - Supports Dropwizard metrics
- https://github.com/performancecopilot/parfait


GBPUSD M15
1.4505
1.00
1.4508
SL/TP
pcpcoin

Example application - pcpcoin

- https://github.com/ryandoyle/pcpconf18-java-tooling
  - Spring Boot (https://spring.io/projects/spring-boot)
- `POST /customer -d {“name”: “Ryan Doyle”}`
  - Creates new customer
- `POST /transaction -d {“from”, 123, “to”: 456, “amount”: 500}`
  - Creates a new transaction
  - Randomly fails for invalid IDs & not enough funds
- Simulated load via Gatling (https://gatling.io/)

DEMO

```
$ ./generate_users.sh
{"resource":"/customer/39568"}
{"resource":"/customer/39569"}
{"resource":"/customer/39570"}

$ ./generate_transactions.sh
{"resource":"/transaction/1921272b-50c5-43ee-a7f7-8ea7cd7446d9"}
{"message":"There are not enough funds in your account","path":"/transaction"}
{"resource":"/transaction/8942dbe0-e0a4-4e7a-a86f-1b37965b0768"}
{"resource":"/transaction/ce460b4f-46f1-4dcd-a22e-088fd4ba2f4c"}
{"user ID is invalid","path":"/transaction"}
{"resource":"/transaction/aefc483b-ca88-4b70-bd8e-763927c05aeb"}
{"resource":"/transaction/bba71714-b6b4-498d-a850-3eef2e9b4532"}
```

parfait-agent

parfait-agent

- Monitors via `-javaagent` or standalone proxy-mode
- Basic metrics/anything exposed via JMX
  - Configuration file for additional metrics

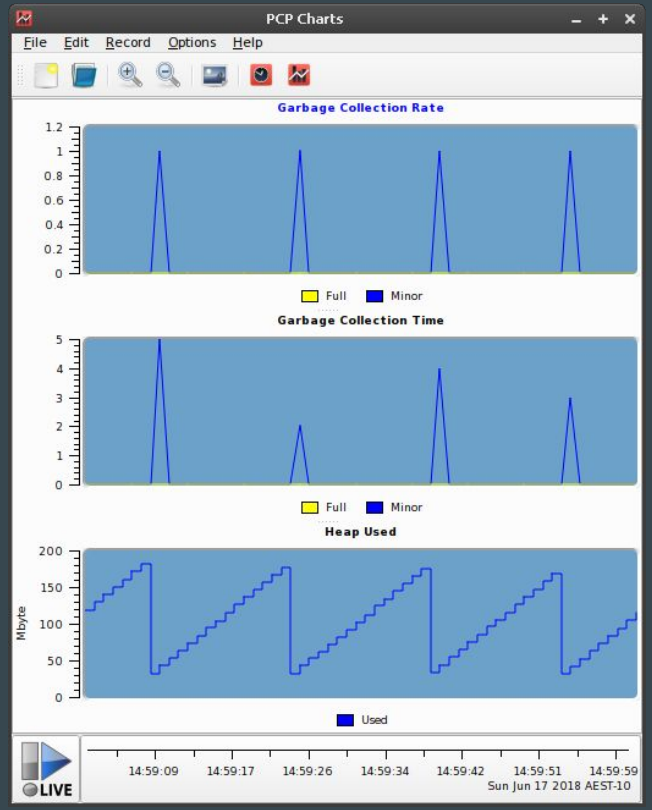
```
{
    "metrics": [
     {
         "name": "java.memory.heap[init]",
         "description": "Initial Java heap memory configuration size",
         "units": "bytes",
         "mBeanName": "java.lang:type=Memory",
         "mBeanAttributeName": "HeapMemoryUsage",
         "mBeanCompositeDataItem": "init"
     },
     ...
}
```

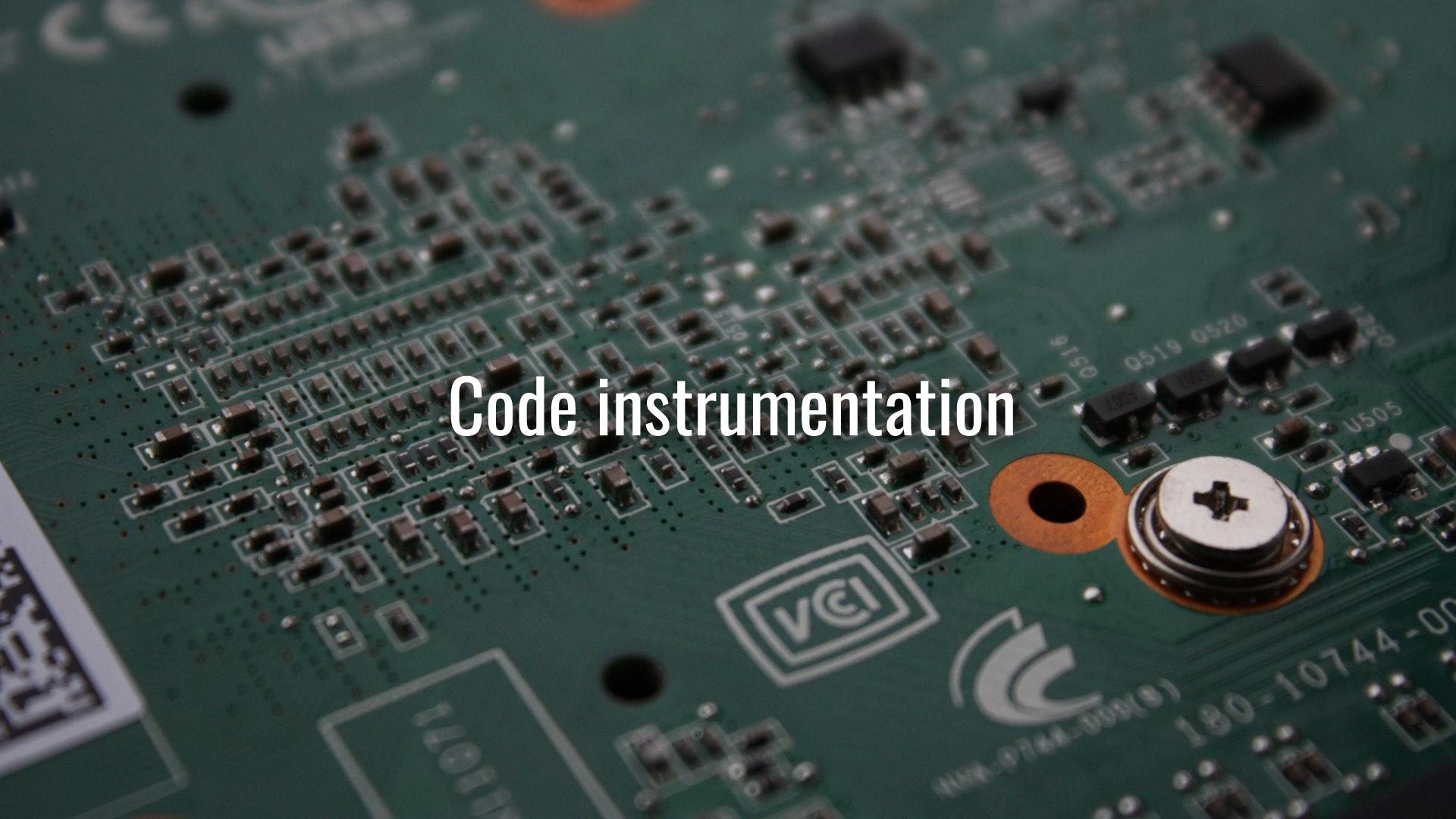
DEMO

```
java
   -javaagent:parfait-agent.jar
   -Dparfait.name pcpcoin
   -jar myApp.jar
```

DEMO

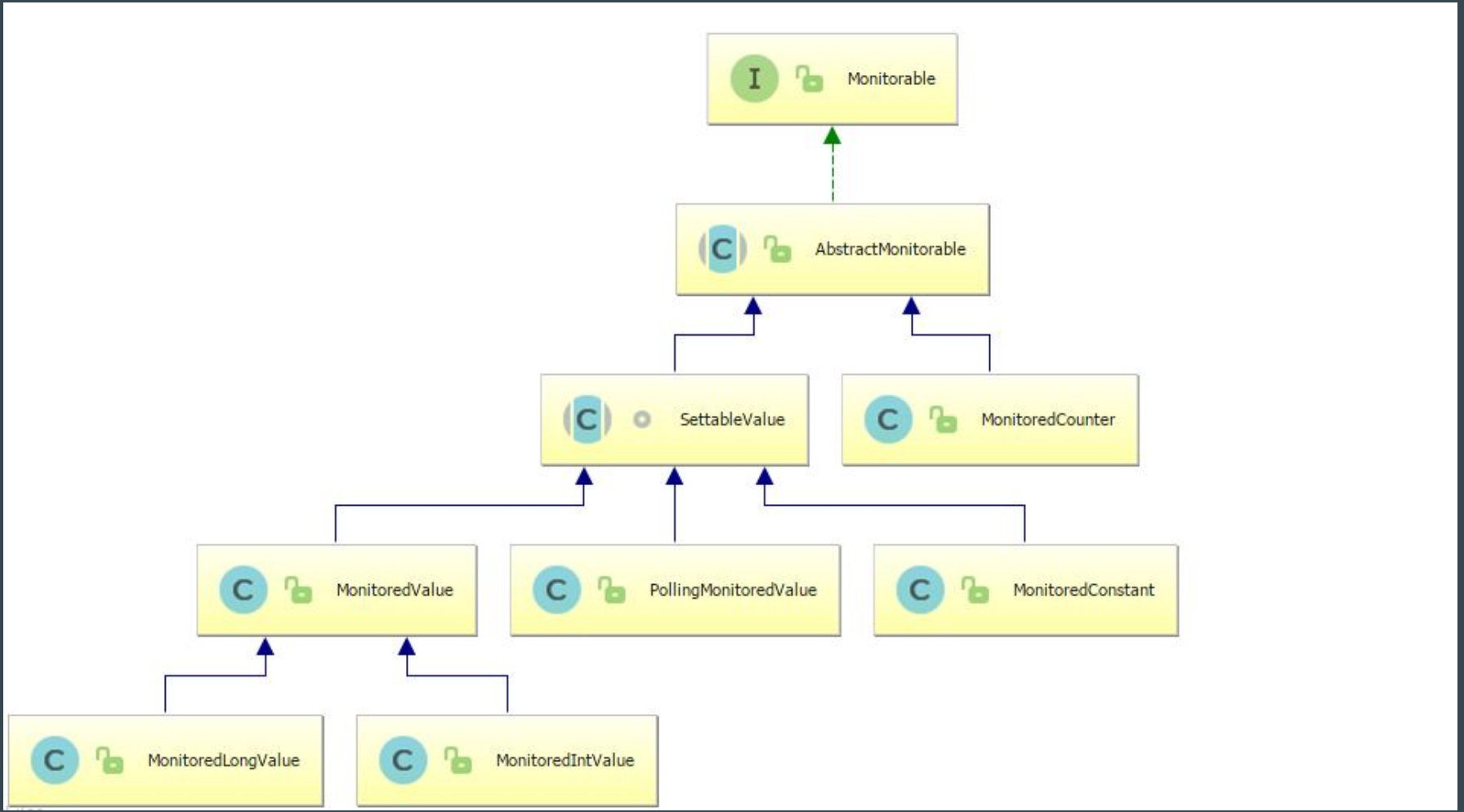
```
./gradlew startWithAgent

pminfo mmv.pcpcoin

pmchart -c pmchart-views/demo1 &

./gradlew gatlingRun
```




Code instrumentation

Basic code architecture

- **`Monitorable<T>`**: base interface for all
  - `MonitoredCounter, MonitoredValue, MonitoredConstant` etc... implementations
- **`TypeHandler<T>`**
  - Know how to write generic types to PCP's view of the world
- **`MonitorableRegistry`**: monitorables register themselves against this
- **`DynamicMonotoringView`**: glues `MonitorableRegistry` and `PCPMmvWriter` together
- **`PCPMmvWriter`**: knows how to write the on-disk format. Uses `TypeHandlers`


Monitorable
AbstractMonitorable
SettableValue
MonitoredCounter
MonitoredValue
PollingMonitoredValue
MonitoredConstant
MonitoredLongValue
MonitoredIntValue

Examples

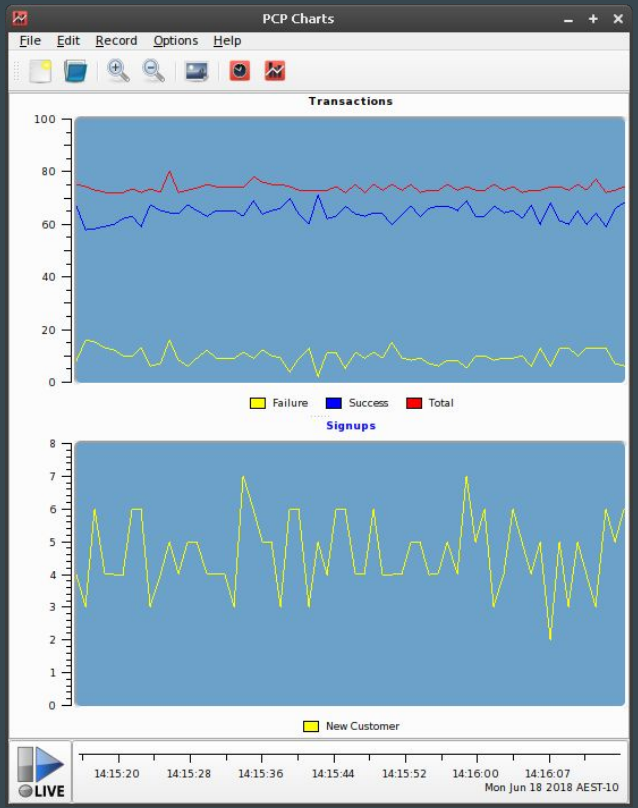
```
// Registers with static DEFAULT_REGISTRY unless given as an argument
Counter counter = new MonitoredCounter(“pcpcoin.customer.new”, “New customer
signups”);


// Thread-safe increment
counter.inc();
```

DEMO - Basic Metrics

See:
com.example.pcpconf18.pcpcoin.service.CustomerService#createNewCustomer
com.example.pcpconf18.pcpcoin.configuration.CustomerMetricsAspect#monitorNewCustomer
com.example.pcpconf18.pcpcoin.service.TransactionService#newTransaction
com.example.pcpconf18.pcpcoin.configuration.TransactionMetricsAspect

```
./gradlew startWithoutAgent
./gradlew gatlingRun
pminfo -f mmv.pcpcoin.customer.new
pmchart -c pmchart-views/demo2 &
```

DEMO - Basic Metrics

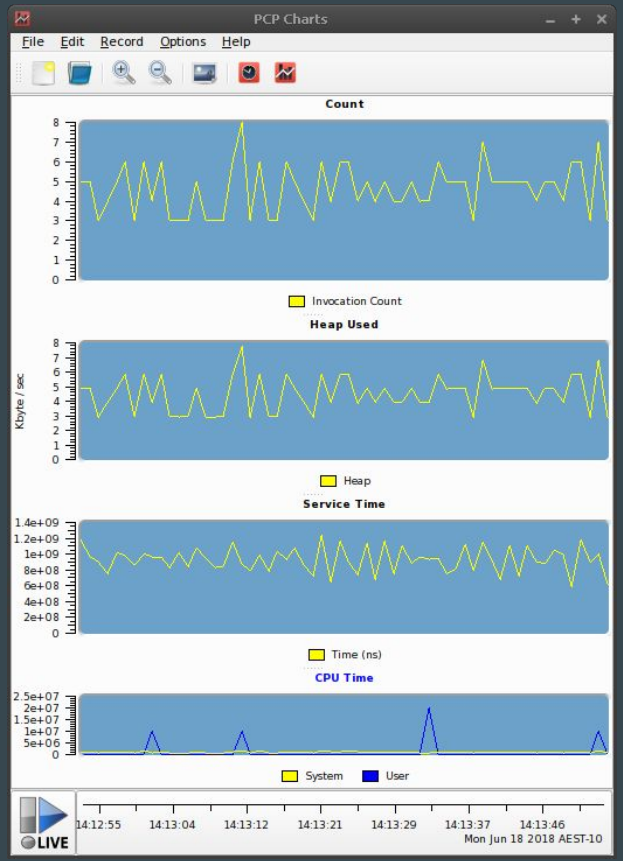


DEMO - Annotation-Based Profiling

See:
com.example.pcpconf18.pcpcoin.service.CustomerService#createNewCustomer

```
pminfo -f mmv.pcpcoin.profiled.customerService
pmchart -c pmchart-views/demo3 &
```

DEMO - Annotation-Based Profiling

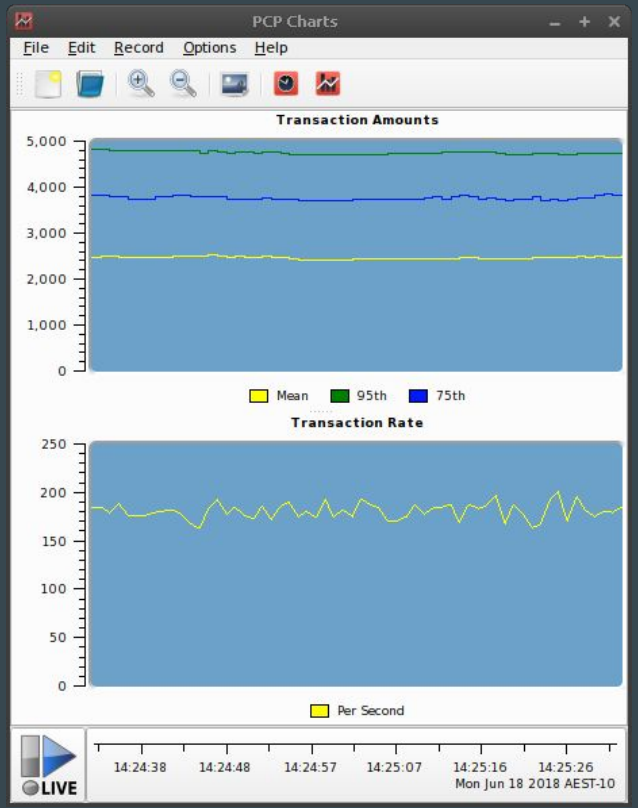


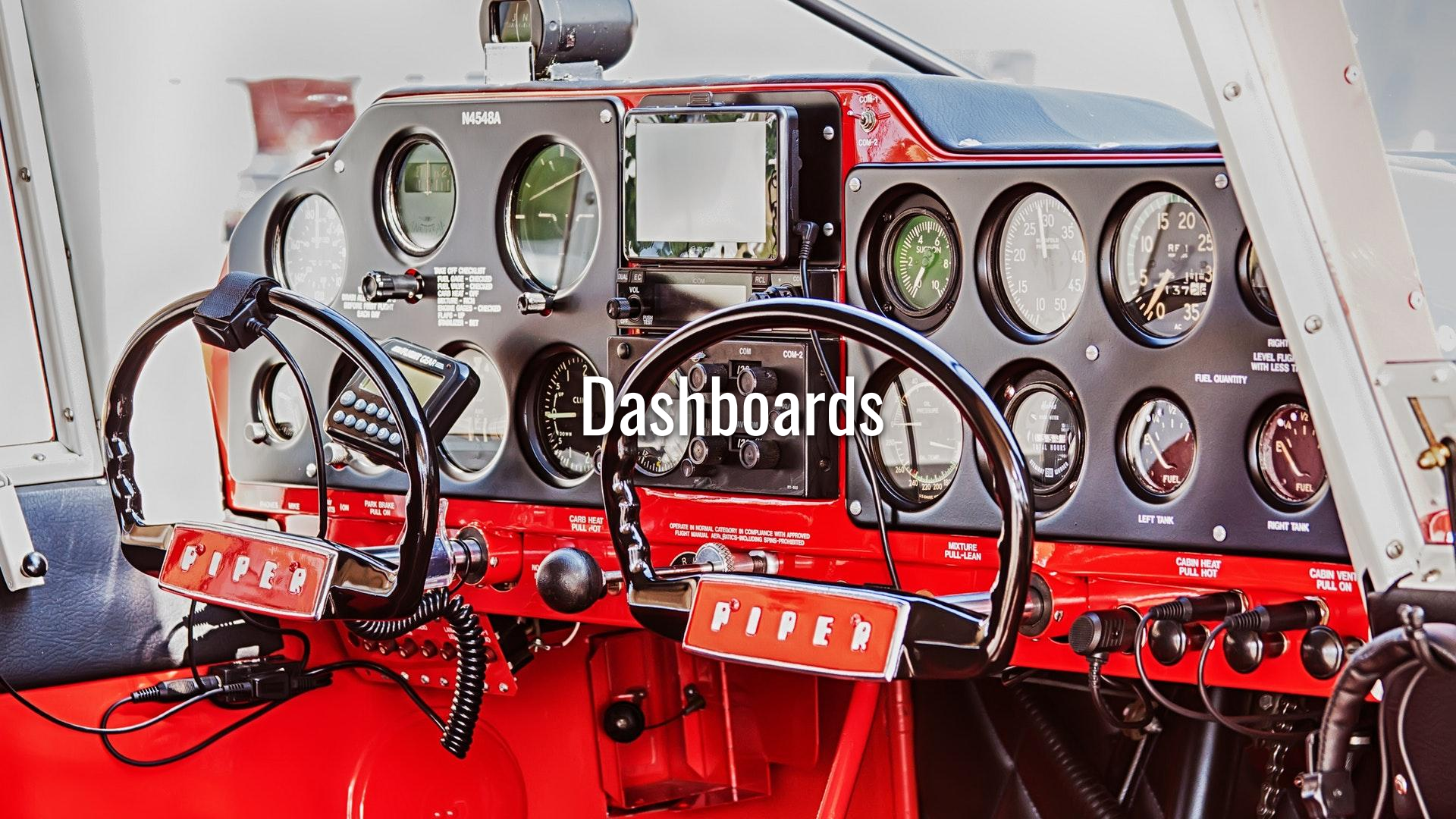
DEMO - Integrations

See:
com.example.pcpconf18.pcpcoin.configuration.TransactionMetricsAspect#monitorSuccessfulTransaction

```
pminfo -f mmv.pcpcoin.transaction.amount
pmchart -c pmchart-views/demo4 &
```

DEMO - Integrations



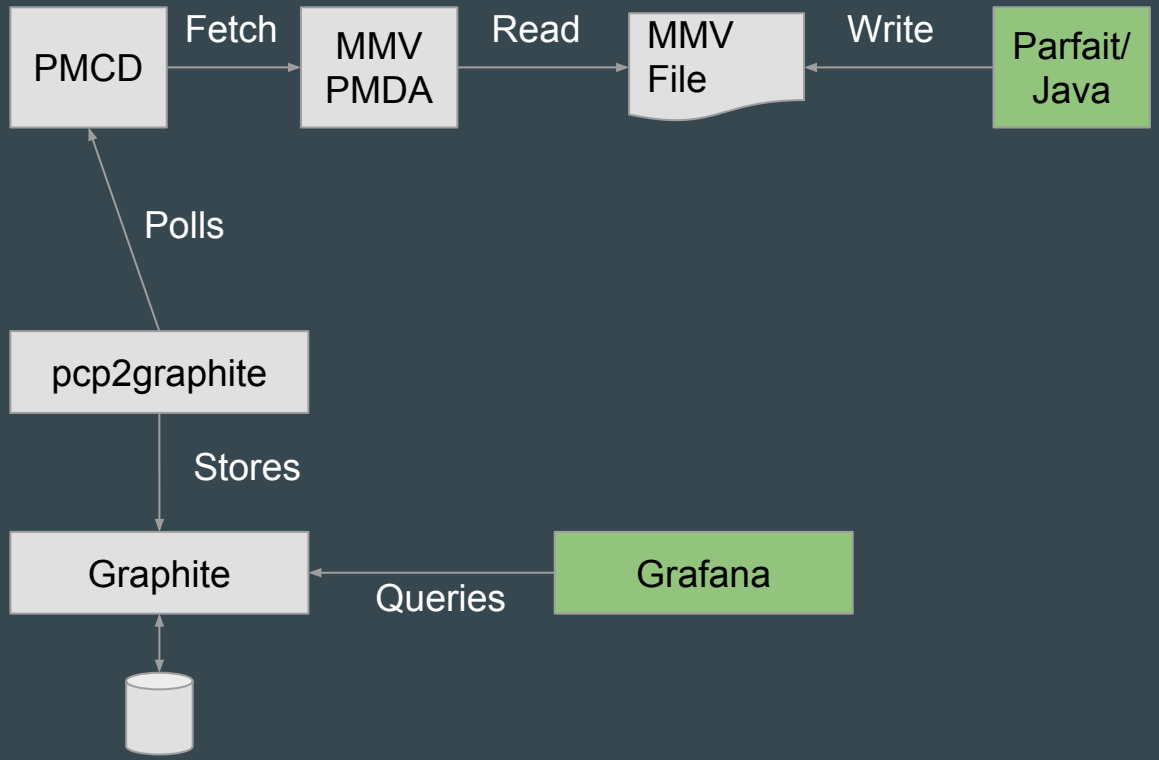

N4548A
PIPER
PIPER
FUEL QUANTITY
LEFT TANK
RIGHT TANK
CARB HEAT
PULL HOT
MIXTURE
PULL-LEAN
CABIN HEAT
PULL HOT
CABIN VENT
PULL ON
PARK BRAKE
PULL ON

Dashboards

Why I like them

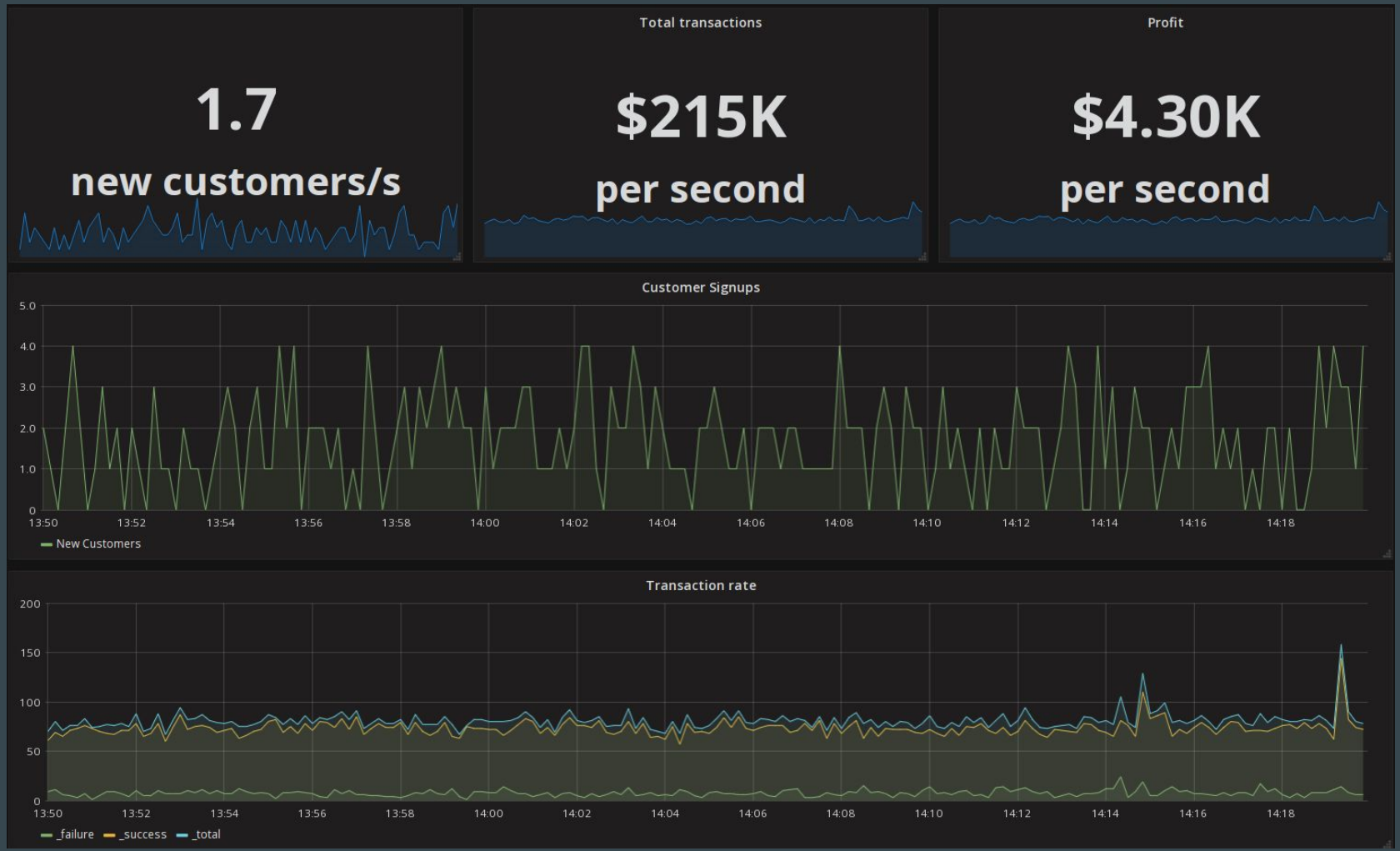
- Expose **business critical metrics**
  - Sales
  - Signups
  - aka, $$$
- Critical to establish shared language between engineers, product, sales etc...

What we will build



DEMO - Building dashboard

docker-compose up -d
./configure_grafana.sh
./pcp2graphite.py -t 1sec -h localhost mmv.pcpcoin &
Import grafana_dashboard.json


1.7
new customers/s
Total transactions
$215K
per second
Profit
$4.30K
per second
Customer Signups
5.0
4.0
3.0
2.0
1.0
0
13:50
13:52
13:54
13:56
13:58
14:00
14:02
14:04
14:06
14:08
14:10
14:12
14:14
14:16
14:18
New Customers
Transaction rate
200
150
100
50
0
13:50
13:52
13:54
13:56
13:58
14:00
14:02
14:04
14:06
14:08
14:10
14:12
14:14
14:16
14:18
_failure
_success
_total

Singlestat | General | Metrics | Options | Value Mappings | Time range

Data Source: default

A	pcp	mmv	pcpcoin	transaction	count	_success	multiplySeries(#B)	+
B	pcp	mmv	pcpcoin	transaction	amount	mean	+	
C	Add Query							

Thank you