Dropwizard

production ready restful web services...

... for java

preamble

... the case for dropwizard

Dropwizard is an exemplar for **ops-friendly**, **self contained**, **high performing**, **framework-less** web service building technologies

... written in java

The **concepts**, **principles** & **patterns** contained herein can and should be applied to any services you build irrespective of technology or language

“micro service architecture”

bit.ly/187C3Le

The rise of **real service oriented architecture** has meant that services are now an acceptable level of abstraction for defining system features

“the world needs devops”

develop ⇝ deploy ⇝ disaster

a typical release pipeline

When things go smoothly our initial reaction is one of **surprise** followed by **suspicion**

ITS THEIR FAULT!

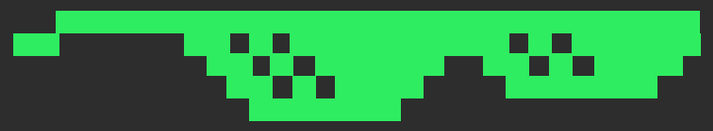
WHAT THE HELL IS THIS

develop ⇝ deploy ⇝ disaster

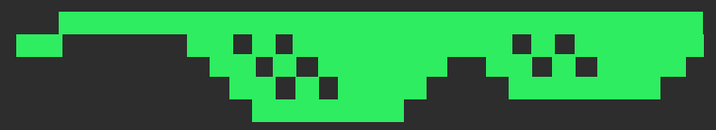
THEY DON'T GET IT!

news flash!

You are **not** writing code for your own gratification

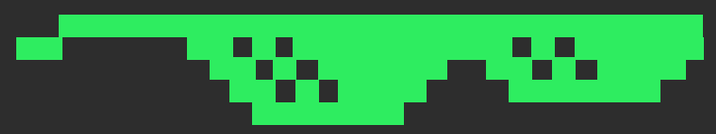


You are trying to make **others** lives better

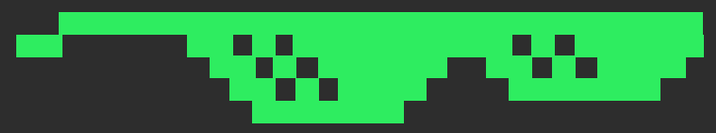


Your code and infrastructure **will**

fail a lot



You are **not** in competition with other departments



You **need** a shared understanding of system behaviour


DEAL WITH IT!

Dropwizard provides a set of features that enable you to build systems capable of adapting to, both, a changing **business** and **operational** domain

Dropwizard

the actual topic of the talk

Dropwizard is a Java framework for developing ops-friendly, high-performance, RESTful web services.

http://dropwizard.io

A **damn simple** library for building **production-ready** RESTful web services

github.com/dropwizard/dropwizard

It's a little bit of **opinionated glue code** which bangs together **a set of libraries** which have historically not sucked

github.com/dropwizard/dropwizard

Jetty	HTTP Library
Jersey	REST Library
Jackson	JSON Processor
Metrics	Instrumentation

Guava	Immutable Collections & Utils
Logback	Logging
Hibernate Validator	Validation
JDBI & Liquibase	Data Access & Migration
Freemarker & Mustache	Templating
JodaTime	Non-Terrible Time

The “Nobody’s Perfect”

- Java is Java
- Annotations are awful
- Documentation is dispersed

Lets Build a Service!!!

1. Create Configuration YML & Class

2. Create the Service

3. Create the Representation

4. Create & Register the Resource

5. Build & Run

So What?

I could have done this with

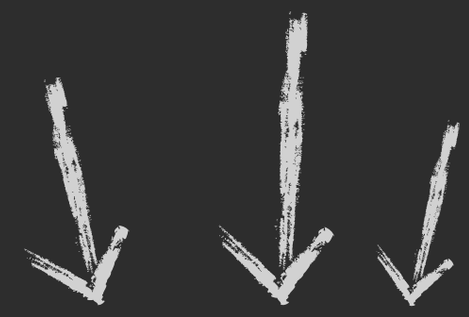
Rails, Sinatra, Spring, Play,
Finagle, Express, Nancy....

The Admin Server

Metrics

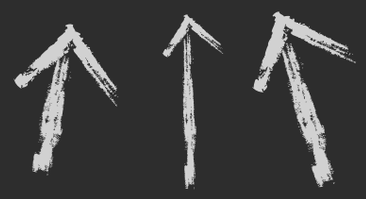
Healthchecks

Managed Objects



```
class DBManager implements Managed {
        public void start() { … }
        public void stop() { … }
}

// register in Service.initialize()
environment.manage(myDBManager)
```

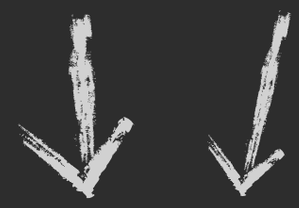


```
class DingusCommand implements Command {
  public void configure(parser) { … }
  public void run(bootstrap, ns) { … }
}
```

```
// register in Service.initalize()
bootstrap.addCommand(myCommand)
```

```
> java -jar app.jar dingus
```

Commands



```
class DingusTask implements Task {
  public void execute(params, out) { … }
}
```

```
// register in Service.run()
environment.addTask(myTask)
```

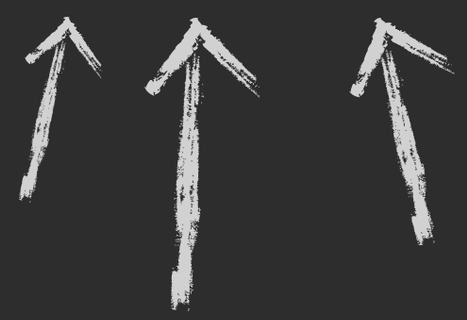
Tasks

```
> curl -X POST http://server/tasks/dingus
```

Bundles

Reusable modules of functionality

```
@Override
public void initialize(bootstrap) {
    bootstrap.addBundle(
      new AssetsBundle("/assets/", "/"));
}
```



Testing (Fixtures)

```
import static com.yammer.dropwizard
    .testing.JsonHelpers.*;

jsonFixture(“fixtures/person.json”)

fromJson(myFixture, Person.class)

asJson(person)
```

Testing (Resources)

```
public class MyTest extends ResourceTest {
    @Override
    protected void setUpResources() {
        addResource(...);
    }
}

client().resource(“user/1”).get(User.class)
```

More Other Things & Stuff

Views

Data Access

Banners

Migrations

Authentication

etc...

Dropwizard

production ready restful web services...