# ESB with Openshift

COMPANY

GEPARDEC IT SERVICES GMBH

SUPERVISOR

DI (FH) PETER KULCZYCKI

# Content

Infrastructure as Code

Openshift Basics

MicroProfile Specifications

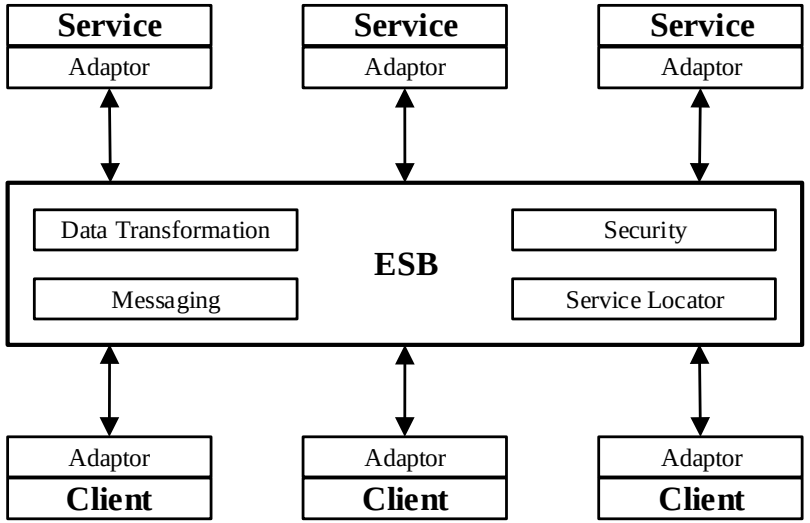
Prototype Design / Implementation

Demo

Evaluations

JBoss Fuse

http://dev.solace.com




OPENSHIFT

https://www.openshift.com/

# Infrastructure as Code

## Consistency

All systems are consistent with the definition

## Repeatability

Changes can be repeated with same outcome

## Reproducibility

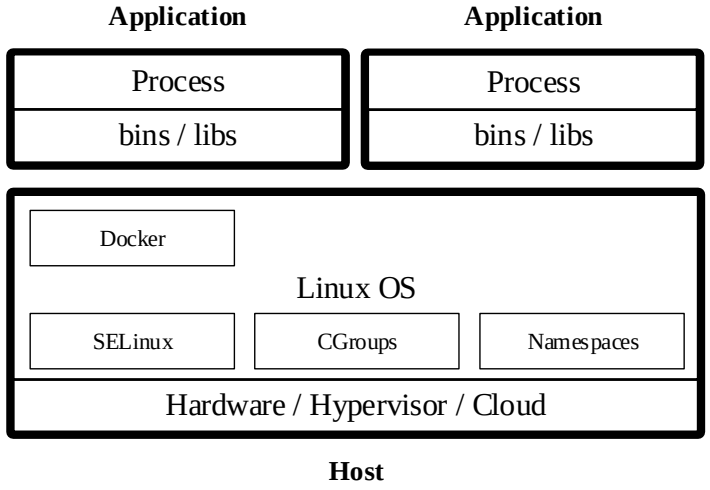
System and changes can be reproduced

## Disposable

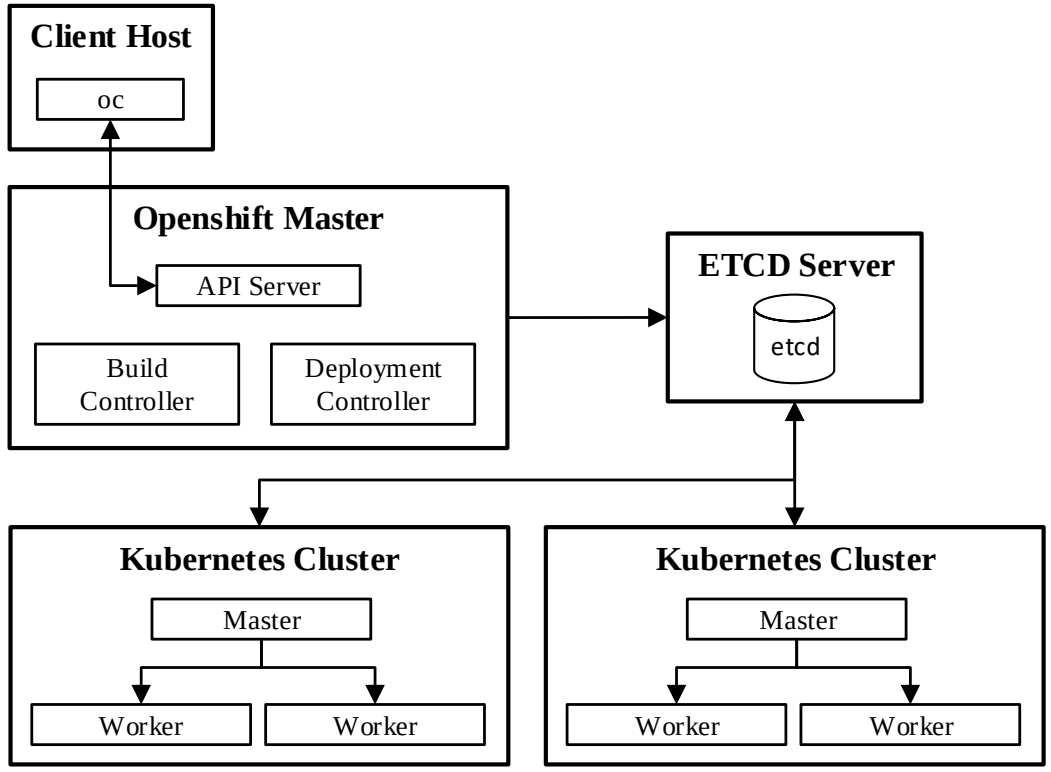
Dispose and Re-Create instead of heavy updates

```
FROM nginx:1.13
MAINTAINER <christoph.ruhsam@gepardec.com>

ENV NGINX_RUN_USER swag
ENV NGINX_RUN_GROUP 0
ENV NGINX_HOME_RUN_USER /home/${NGINX_RUN_USER}
ENV DEBIAN_FRONTEND noninteractive

RUN mkdir -p ${NGINX_HOME_RUN_USER} \
    && useradd \
      --home-dir ${NGINX_HOME_RUN_USER} \
      --shell /sbin/nologin \
      -g ${NGINX_RUN_GROUP} \
      -u 1001 \
      ${NGINX_RUN_USER} \
    && apt-get update -y \
    && apt-get install -y curl \
    && apt-get autoremove -y \
    && apt-get autoclean -y \
    && rm -r /var/lib/apt/lists/*
```

# Openshift Basics




**Docker**




**Openshift / Kubernetes Cluster**

# Openshift Basics

## Kubernetes

Namespaces not protected

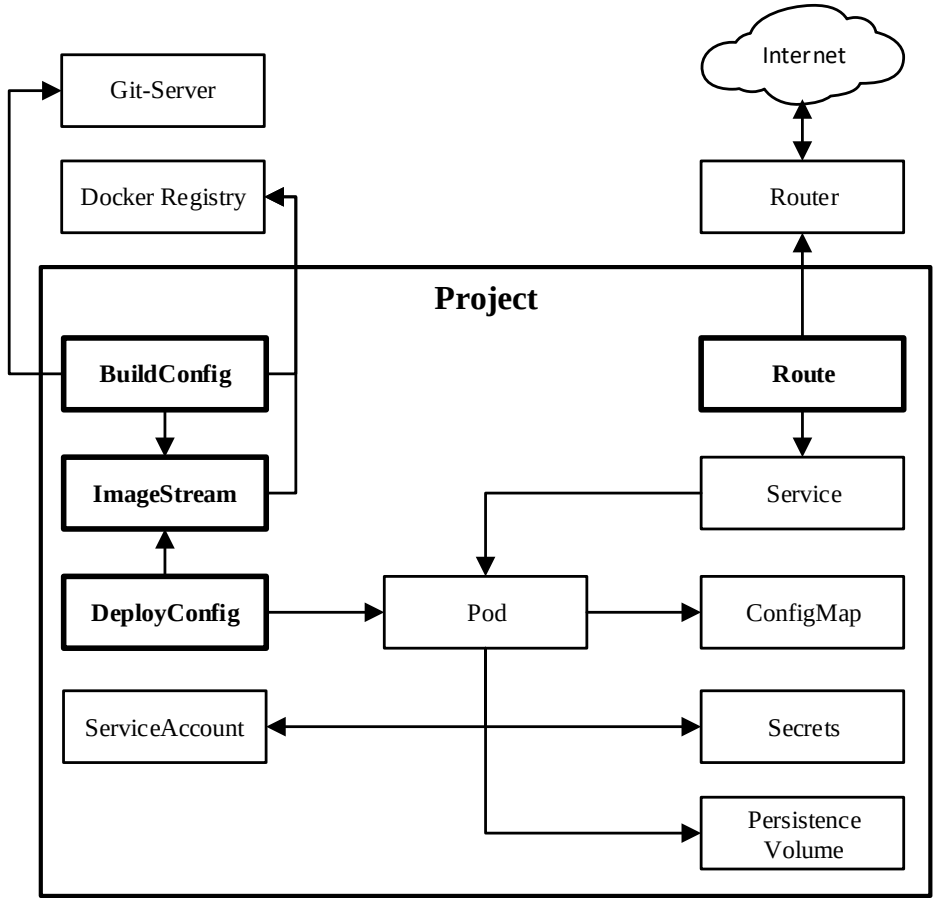
Direct usage of Docker Images

Only cluster state ensured

No rollout behaviors

- Starts Pods
- Stops Pods

No application templates




**Openshift Project**

## Openshift

Project isolates Namespaces

Abstraction of Docker Image (IS)

Automated Build -> Deployment

Different Rollout behaviors

- Canary releases
- Blue – Green Releases

Lots of application templates

# Microprofile Specifications

## MicroProfile-Config

ConfigSource, Parameter Injection

## MicroProfile-Fault-Tolerance

Resilience, Retries, Timeouts, Fallbacks

## MicroProfile-Health

Health endpoint ready / liveness probes

## MicroProfile-Metric

Counter, Timer, node state, runtime state

## MicroProfile-OpenTracing

Service / Method call chain tracing

OPTIMIZING ENTERPRISE JAVA

https://microprofile.io


https://wildfly-swarm.io/

HYSTRIX

DEFEND YOUR APP

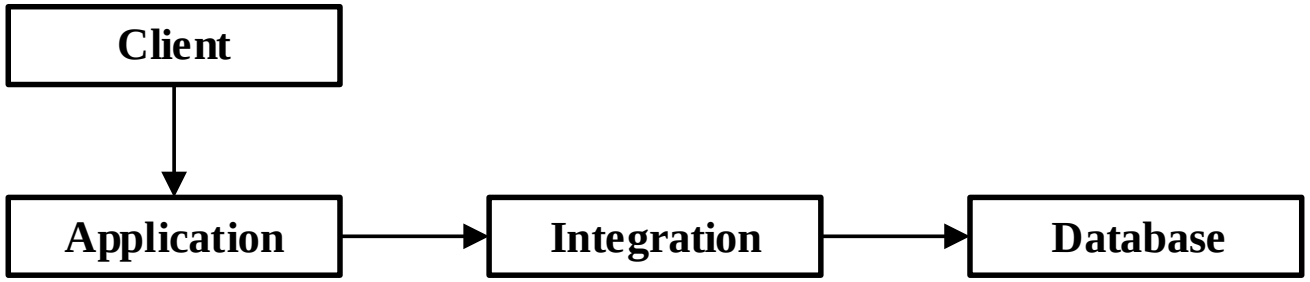
https://github.com/Netflix/Hystrix

# Prototype

https://www.keycloak.org/

https://www.jaegertracing.io/

https://www.graylog.org/




https://microprofile.io

https://wildfly-swarm.io/

https://www.openshift.com/

# Prototype Openshift Project

## Legacy Database

Only accessed by the integration service

## Integration Service

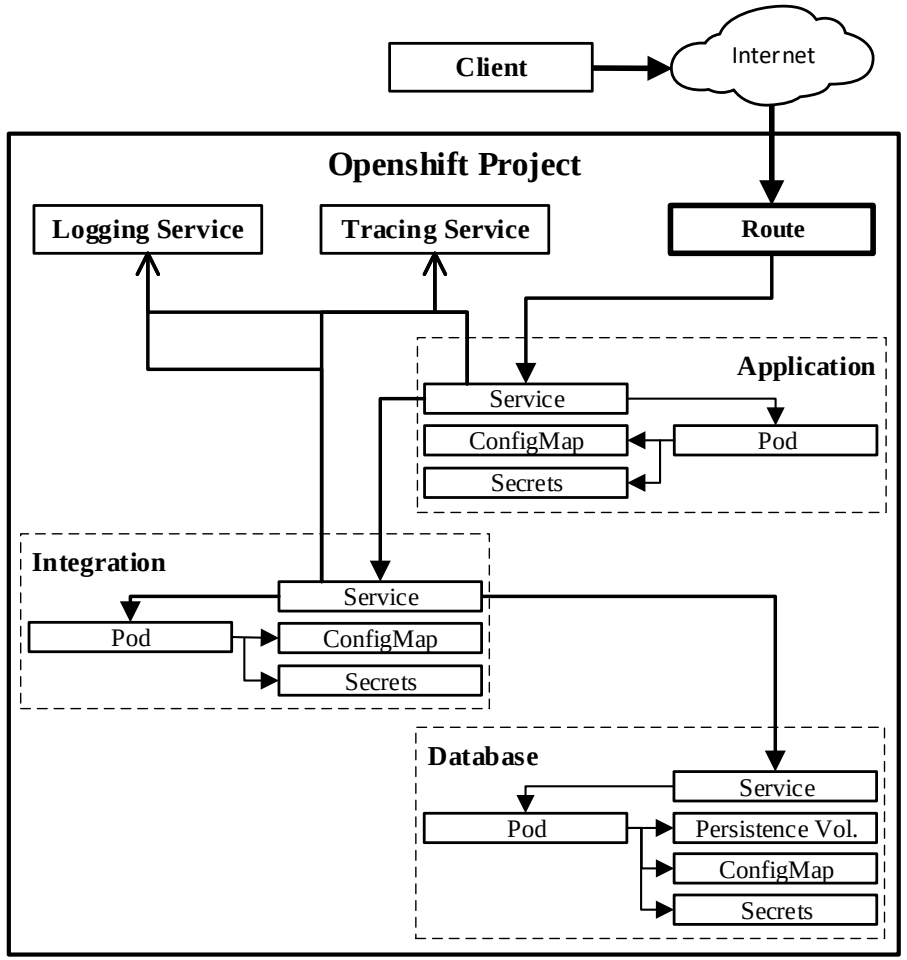
Performs database access and data transformation

## Application Service

Consumes data loaded by the integration service

## Client

Consumes data loaded by the application service



# Prototype Monitoring

## Jaeger (Opentracing)

Tracks spans of calls

JAEGER

https://www.jaegertracing.io/

## Graylog

Log aggregation over all services

https://www.keycloak.org/

## Keycloak

Oauth2 authentication between services/client-service

https://www.graylog.org/

## Swagger-UI

Tooling for testing documented REST-API

SWAGGER

SMARTBEAR

https://swagger.io/

# Prototype Implementation Security

## Secured Service

```xml
<dependency>
    <groupId>org.wildfly.swarm</groupId>
    <artifactId>keycloak</artifactId>
</dependency>
```

**1. Declare Wildfly-Swarm Keycloak fraction**

```yaml
volumes:
  - name: keycloak-config
    secret:
      secretName: ${oc.secret-service-integration-keycloak}
```

**2. Inject secret holding adapter config**

```yaml
volumeMounts:
  - name: keycloak-config
    mountPath: ${oc.secret-service-integration-keycloak.dir}
```

**3. Mount injected secret volume**

```yaml
project:
  stage: openshift
swarm:
  keycloak:
    json:
      path: ${oc.secret-service-integration-keycloak.dir}/keycloak.json
  deployment:
    ${project.build.finalName}.war:
      web:
        login-config:
          auth-method: KEYCLOAK
        security-constraints:
          - url-pattern: /rest-api/customer/*
            roles: [consumer]
          - url-pattern: /rest-api/order/*
            roles: [consumer]
          - url-pattern: /rest-api/admin/*
            roles: [consumer]
```

**3. Configure Openshift stage security**

# Prototype Implementation Security

## Secured Service Consumer

```
volumes:
  - name: app-config
    secret:
      secretName: ${oc.secret-service-app}
```

**1. Inject secret holding adapter config**

```
volumeMounts:
  - name: app-config
    mountPath: ${oc.secret-service-app.dir}
```

**2. Mount injected secret volume**

```
swarm:
  microprofile:
    config:
      config-sources:
        app.secrets:
          dir: ${oc.secret-service-app.dir}
```

**3. Define where to load injected configurations**

```
@Inject
@ConfigProperty(name = "keycloak.token-url")
private String keycloakTokenUrl;
@Inject
@ConfigProperty(name = "keycloak.client.id")
private String keycloakClientId;
@Inject
@ConfigProperty(name = "keycloak.client.secret")
private String keycloakClientSecret;

private ClientCredentialsTokenRequest tokenRequest;

@PostConstruct
public void postConstruct() {
    tokenRequest = new ClientCredentialsTokenRequest(new NetHttpTransport(),
                                                     new JacksonFactory(),
                                                     new GenericUrl(keycloakTokenUrl));
    tokenRequest.setClientAuthentication(new ClientParametersAuthentication(keycloakClientId,
                                                                            keycloakClientSecret));
}

@Produces
@OAuthToken
@Dependent
@Counted(name = "retrieved-oauth-tokens", monotonic = true)
@Logging(mdcConfig = Logging.MDCConfig.GROUP_REST_SECURITY, skipResult = true)
@Retry(delay = 100L, maxRetries = 5, retryOn = {TokenResponseException.class, IOException.class})
@Timeout(value = 2L, unit = ChronoUnit.SECONDS)
String obtainOauthToken() throws IOException {
    return tokenRequest.execute().getAccessToken();
}
```

**4. Implement Oauth2 token retrieval**

# Prototype Implementation Logging

## All Services

```xml
<dependency>
    <groupId>org.wildfly.swarm</groupId>
    <artifactId>logging</artifactId>
</dependency>

<dependency>
    <groupId>org.jboss.slf4j</groupId>
    <artifactId>slf4j-jboss-logging</artifactId>
    <version>${slf4j.jbopss-logging.version}</version>
</dependency>
```

**1. Declare logging fraction and log implementation**

```java
@Override
public void filter(ContainerRequestContext requestContext) throws IOException {
    final String tracingId = spanContextInstance.get().toString().split( regex: ":")[0];
    scopeInstance.get().span().setTag(MDC_TX_ID, tracingId);
    log.info("Setting MDC transaction id");
    MDC.put(MDC_TX_ID, tracingId);
}
```

**2. Capture OpenTracing trace id to connect service logs to current trace**

```yaml
swarm:
  logging:
    pattern-formatters:
        DEFAULT_LOG_PATTERN:
          pattern: "%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p (%t) [%C{2}] \
                    trace.id=%X{trace.id} | \
                    trace.group=%X{trace.group} - %m%n"
    custom-handlers:
      SYSLOGGER:
        named-formatter: DEFAULT_LOG_PATTERN
        attribute-class: org.jboss.logmanager.handlers.SyslogHandler
        module: org.jboss.logmanager
        properties:
          serverHostname: graylog
          hostname: ${project.build.finalName}
          port: 10514
          protocol: UDP
```

**3. Configure log formatter and syslog log handler**

# Prototype Implementation Tracing

## All Services

```xml
<dependency>
    <groupId>org.wildfly.swarm</groupId>
    <artifactId>opentracing</artifactId>
</dependency>
```

**1. Declare OpenTracing fraction**

```yaml
env:
  - name: "JAEGER_HOST"
    valueFrom:
      secretKeyRef:
        name: ${oc.secret-service-app}
        key: jaeger.host
  - name: "JAEGER_PORT"
    valueFrom:
      secretKeyRef:
        name: ${oc.secret-service-app}
        key: jaeger.port
```

**2. Inject Jaeger host/port from secret into env variables**

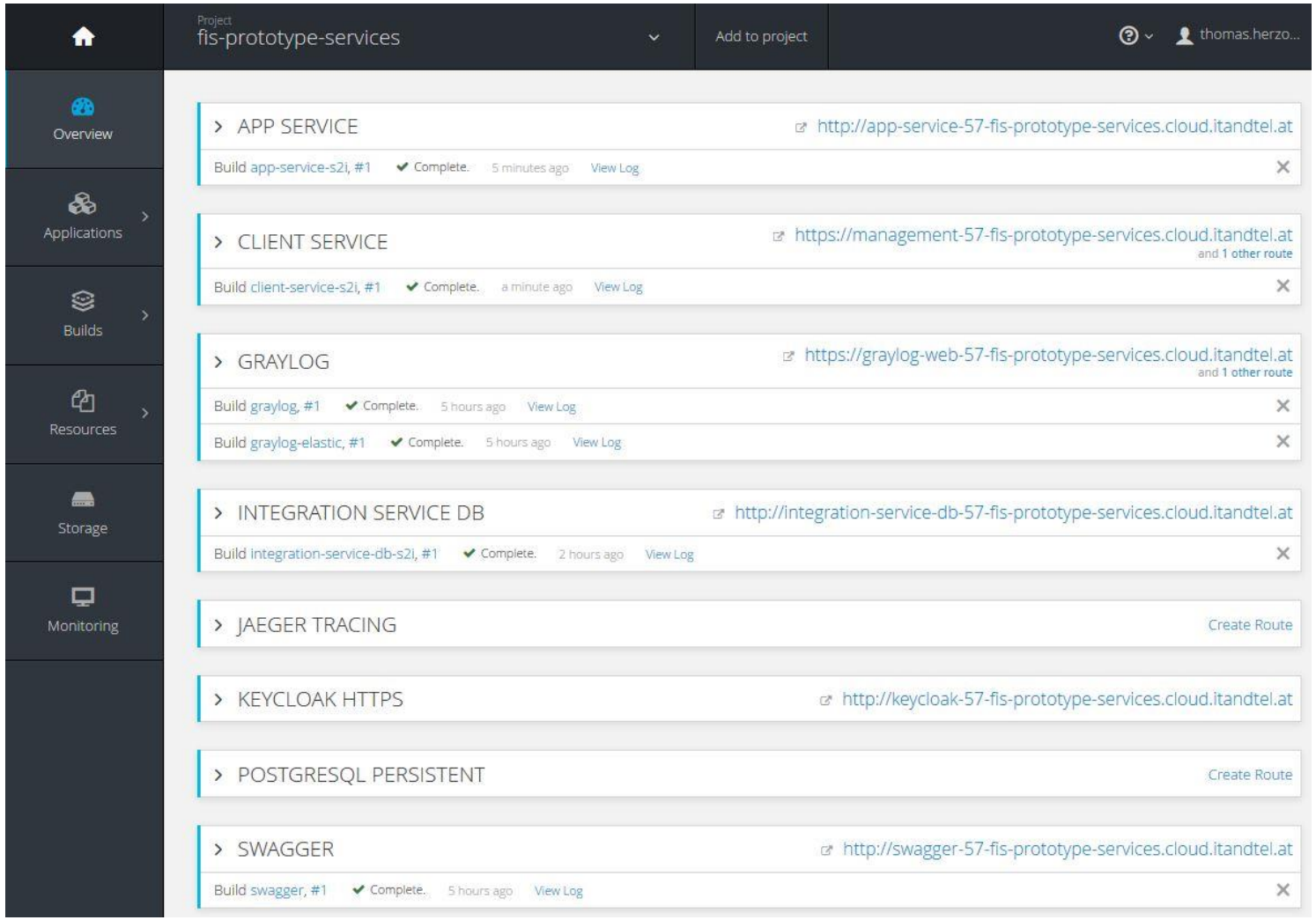
```yaml
# Necessary because config set via 'swarm.jaeger.*', breaks jaeger integration
JAEGER_SERVICE_NAME: "${project.build.finalName}"
JAEGER_AGENT_HOST: "${env.JAEGER_HOST}"
JAEGER_AGENT_PORT: "${env.JAEGER_PORT}"
JAEGER_REPORTER_LOG_SPANS: "true"
JAEGER_REPORTER_FLUSH_INTERVAL: "1000"
JAEGER_SAMPLER_TYPE: "const"
JAEGER_SAMPLER_PARAM: "1"
```

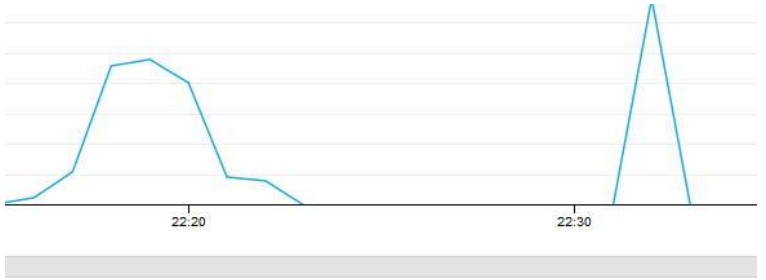
**3. Configure OpenTracing fraction**

```java
@ApplicationScoped
@Traced
@Logging(mdcConfig = Logging.MDCConfig.GROUP_SERVICE)
public class ReportServiceImpl implements ReportService {
```

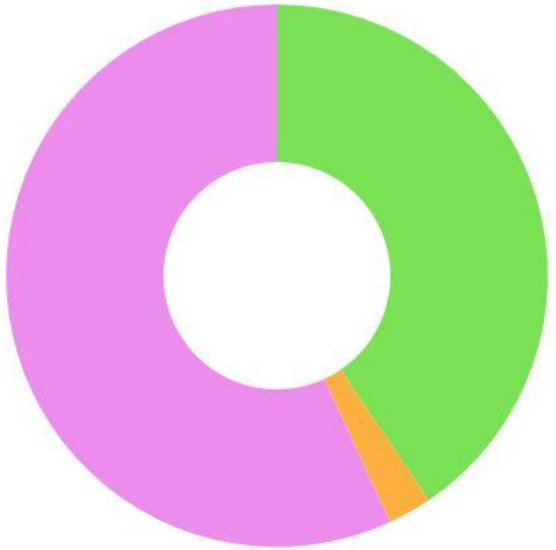
**4. Annotate traceable type or method with @Traced**

# Prototype DEMO




Messages per source



# Evaluations

## Goals of the thesis

### Multistage Configuration

How to configure different kind of services for multiple stages in Openshift, with least of effort ?

### API-Management

How to manage different version of services in Openshift ?

### Transformers and Adapters

How to integrate adapters and transformer services in Openshift ?

### Security

How to secure services in Openshift ?

# Evaluation Multistage Configuration

## MicroProfile-Conifg

Abstraction form underlying configuration source (ENV, System Props, properties files, custom source)

```
@Inject @ConfigProperty(name="project.developers") private Developer myProp;
project.developers=[{"name": "Thomas"}, {"name": "Erhard"}]
```

## Wildfly-Swarm project-stages.yml / project-<stage>.yml

Wildfly-swarm configurations support ENV, System Props and Maven Props

```
java –jar –s other-projet-stages.yml –Dswarm.project.stage=dev
```

## Openshift ConfigMap / Secret

Application environment provides configuration and secrets and protects them

One project, one stage, one configuration, same binary

# Evaluation API-Mangement

## Need for API Management

- Used by multiple clients
- External clients could be involved
- Are all clients known ?

## Requirement for good API Management

- Design a stable API from the beginning
- Keep abstraction between internal and external models
- Plan your migrations well
- Keep only backward compatible as long as needed
- Public services need API access tracking

https://www.benfranklinplumbingmn.com

# Evaluation API-Mangement

## Path Versioning

/rest-api/customer/v1/list

## Content-Type / Accept Header Versioning

Accept: application/vnd.myapp.user.v1+json;qs=0.9

Accept: text/json; version=1;qs=0.5

## Query Parameter Versioning

/rest-api/customer/get/1?version=2

## Swagger

Annotations, CodeGen, UI

SWAGGER

SMARTBEAR

https://swagger.io/


https://blog.cloud-elements.com

# Evaluation Transformers and Adapters

## Global Public Service API *(mostly REST)*

Clients use provided / generated client or integrate it manually. *(Swagger CodeGen)*

Public API must abstract from underlying models, no model pass through

## Data Transformation

Java-Code, Dozer, XSD, JSON, YAML, … possible in microservice itself

If separately needed, then build up the same way as the other services *(act as proxy service)*

## Adapter

Not needed for internal service hosted on ESB *(Openshift)*

Service in Openshift can act as Adapter for external service

Adapter can be provided for external service

# Evaluation Security

## Openshift Project

Isolates services within a namespace *(Controlled internal/external access)*

## Openshift Secrets

Only reference secret name and property keys, but never values

## Normal JEE-Security

Security Constraints, EJB-Security, Deltaspike-Security, JEE8-Security-API, …

## Microprofile-Config

No source code has access to configuration source

Thank you for your attention

Any questions ?