
ORACLE®

JavaOne™

ORACLE®

Keep Learning with Oracle University

ORACLE UNIVERSITY

Classroom Training

Learning Subscription

Live Virtual Class

Training On Demand

Cloud

Technology

Applications

Industries

education.oracle.com

Session Surveys

Help us help you!!

- Oracle would like to invite you to take a moment to give us your session feedback. Your feedback will help us to improve your conference.
- Please be sure to add your feedback for your attended sessions by using the Mobile Survey or in Schedule Builder.

How to Thrive on REST/ WebSocket-based Microservices

Pavel Bucek (pavel.bucek@oracle.com)
Michal Gajdoš (michal@sapho.com)

Oracle/Sapho
October 27, 2015

JavaOne ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Introduction

Why are we doing this?

- Proof of concept
  - Micro is the new black
  - Very small runtime can deliver key functionality
- JAX-RS (Jersey) + WebSocket (Tyrus)
- Java SE 8 adoption
- Reactive APIs

What's already there?

- Vert.x
- Dropwizard
  - Jetty, Jersey, Jackson, …
  - No WebSocket
- Glassfish / Payara
  - JAX-RS, WebSocket, Servlet, JSF, BV, CDI, …
  - ~60MB

QUESTION
Typical production deployment in my company is:

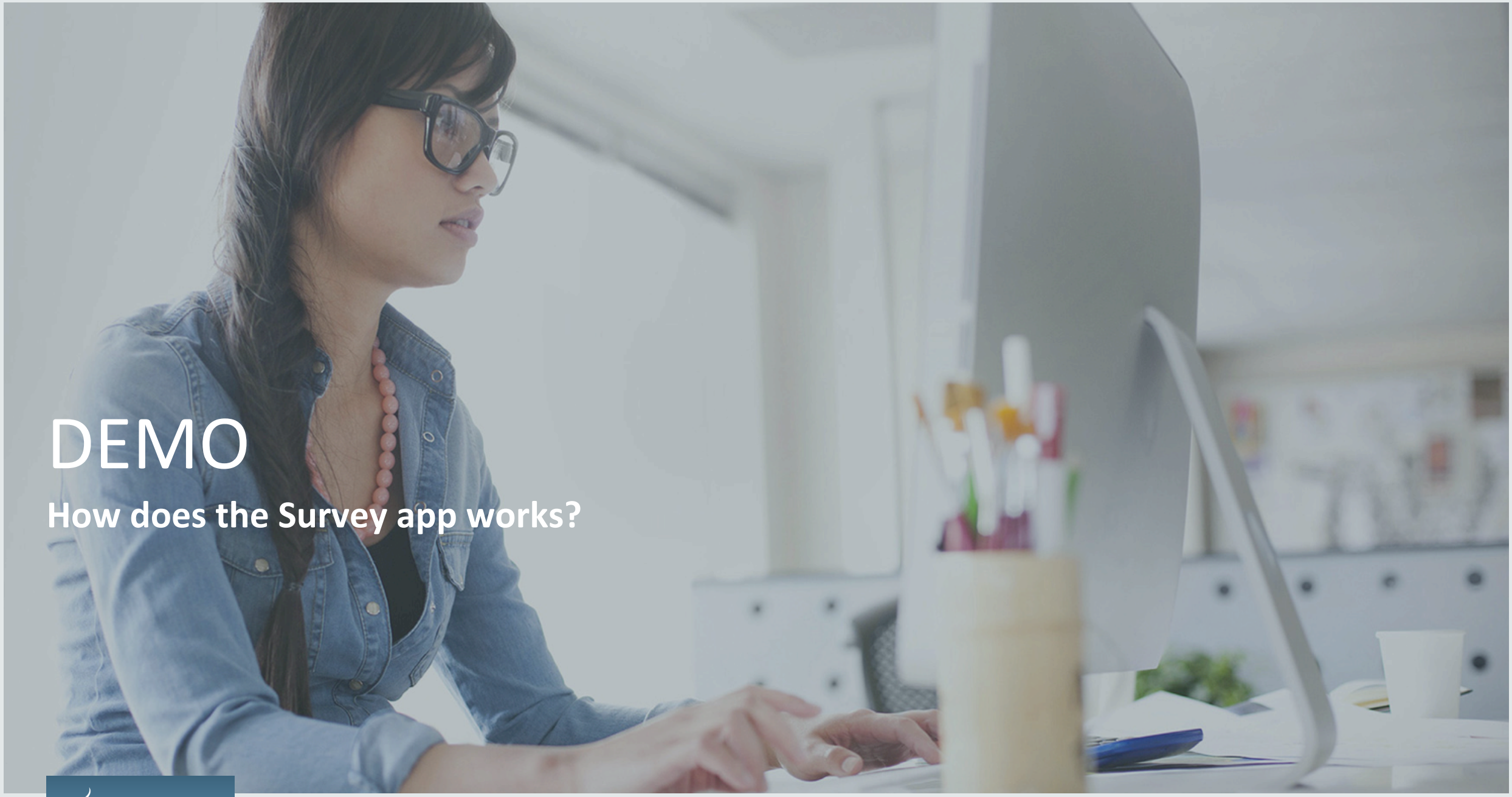
A. Micro-service (small containers, dynamic up/down scaling)

B. Monolithic application (big containers, static resources, active/passive backup)

C. Combination of previous

Sample application

- Simple poll
  - Jersey MVC
- Results are updated based on new votes
  - WebSocket connection (could be SSE..)
- Simple filtering (websocket based)
  - Interactive part of the app
  - WebSocket are more efficient to use



DEMO

How does the Survey app works?

QUESTION
Which Java framework do you use in production?

A. Java EE 6+

B. Spring

C. Java EE 5 (and older) + Java SE

D. Something else, text us what!

DEMO

+CDI

QUESTION
Do you use WebSocket in production?

- **A** Yes.
- **B** Not yet, but we plan to.
- **C** No (we don’t need bi-directional communication).
- **D** No (we use something else).

Micro-services

- Simple, small, decoupled standalone “applications”
  - They don’t do (and provide) anything else than they should
  - Easy to develop and test
- Fast startup time, Fault tolerant
  - Dynamic scaling
  - Cattle vs Pet

<<Placeholder>>

<<Placeholder>>

- Grizzly
  - HTTP container
- HK2
  - Dependency Injection
- Tyrus
  - WebSocket RI
- Jersey
  - JAX-RS RI

DEMO

<<Placeholder>>

QUESTION
Executable JAR size (<<Placeholder>> + poll app) is:

A ~100 MB

B ~50 MB

C ~20 MB

D ~10 MB

E ~5 MB

Summary

- Glassfish
  - Footprint: >200 M
  - Startup time: ~10 s
- Jetty + Weld
  - Footprint: ~28 M
  - Startup time: ~5 s
- <<Placeholder>>
  - Footprint: ~11 M
  - Startup time: ~1s

What do we miss?

Java 8 + Reactive APIs

- Lots of new features in Java 8
- Frameworks slowly up-taking new APIs
- Optional, Lambdas, Streams, …

- Reactive APIs are very handy when you are accessing more services
  - You don't need to wait (block) for the answer
  - Processing hooked to "response(s) received"

QUESTION
Java version in production:

- **A** Java 5 (and older)
- **B** Java 6
- **C** Java 7
- **D** Java 8
- **E** Java 9 (for cutting-edge enthusiasts)

Optional in Jersey

- Server
  - Incoming entities
  - Return values (Outgoing entities)
  - @*Param values
- Client
  - Outgoing entities

Optional in Jersey

```
@GET
@Path("results")
@Template(name = "/survey-results")
public SurveyResults results(@QueryParam("name") final Optional<String> name,
                             @QueryParam("thank") final Optional<String> thank) {
    final String message = name.map(str ->
                format(format: "Thank you, %s, for participating in this survey.", str))
            .map(Optional::ofNullable)
            .orElseGet(() -> thank)
            .orElse("Thank you!");

    return surveyService.surveyResults(id)
            .message(message);
}
```

Optional in Jersey

```
ClientBuilder.newClient()
        .target("http://localhost:8080")
        .request()
        .post(Entity.text(Optional.of(42)));
```

Reactive Client API in Jersey/JAX-RS 2.1

- Utilize the reactive programming model when using Jersey/JAX-RS Client
- Already in Jersey
  - Java 8 – CompletionStage
  - RxJava
  - Guava
- Will be in JAX-RS 2.1

Reactive Client API – Jersey & RxJava

```
RxObservable.newClient().target("http://localhost:8080/destination/visited")
        .request()
        // Reactive invoker.
        .rx()
        // Return a list of destinations.
        .get(new GenericType<List<Destination>>() {})
        // Handle Errors.
        .onErrorReturn(throwable -> Collections.emptyList())
        // Flat the list of destinations.
        .flatMap(Observable::from);
```

Reactive Client API – JAX-RS 2.1 & CompletionStage

```
CompletionStage<List<String>> cs = ClientBuilder.newClient()
        .target("remote/forecast/{destination}")
        .resolveTemplate("destination", "mars")
        .request()
        .rx()
        .get(new GenericType<List<String>>() {});

cs.thenAccept(System.out::println);
```

Programmatic Resources in Jersey

```
new ResourceConfig()
        .registerResources(Resource.builder().path("/")
                .addChildResource("ping").addMethod("GET").handledBy(inflector -> "pong").parent()
                .addChildResource("pong").addMethod("GET").handledBy(inflector -> "ping").parent()
                .build());
```

Lambdas in Tyrus

- Endpoint is just a set of lambda functions
  - @OnOpen, @OnMessage, @OnError, @OnClose
  - WebSocket API already has programmatic version in the specification – see javax.websocket.Endpoint
- It can be applied to server side as well
  - Annotation way is often preferred for server-side deployment
  - Easier to read + integrate with other frameworks (DI, ..)

Lambdas in Tyrus

```
webSocketContainer.connectToServer(new Endpoint() {
    @Override
    public void onOpen(Session session, EndpointConfig config) {
        session.addMessageHandler(String.class, new Whole<String>() {
            @Override
            public void onMessage(String message) {
            }
        });
    }

    @Override
    public void onClose(Session session, CloseReason closeReason) {
    }

    @Override
    public void onError(Session session, Throwable thr) {
    }
}, URI.create("ws://localhost:8025/sample-echo/echo"));
```

Lambdas in Tyrus

```
final WebSocketContainer.SessionBuilder sessionBuilder =
        webSocketContainer.sessionBuilder()
        .path(URI.create("ws://localhost:8025/sample-echo/echo"))
        .messageHandler(String.class, this::onMessage)
        .onOpen(this::onOpen)
        .onError(this::onError)
        .onClose(this::onClose);

sessionBuilder.connect();
```

Lambdas in Tyrus

```
final WebSocketContainer.SessionBuilder sessionBuilder =
        webSocketContainer.sessionBuilder()
        .path(URI.create("ws://localhost:8025/sample-echo/echo"))
        .messageHandler(String.class, this::onMessage)
        .onOpen(this::onOpen)
        .onError(this::onError)
        .onClose(this::onClose);

sessionBuilder.connect();
```

```
var websocket = new WebSocket(window.wsUrl("/sample-btc-xchange/market"));
websocket.onopen = function () {...};
websocket.onmessage = function (evt) {...};
websocket.onclose = function () {...};
websocket.onerror = function () {...};
```

Streams in Tyrus

- Stream processing is another bigger and more notable feature of the Java 8
- MessageHanders (in general - any event handler) can be modeled as endless stream
  - there are some issues with “terminators”, but that could be solved - terminator would unregister current “message handler”; push vs pull..
  - streams can easily: filter, map (encode/decode)

```
session.processMessages(String.class, stream -> stream
        .filter(s -> s.startsWith("prefix"))
        .forEach(System.out::println)
);
```

Contacts

- Pavel Bucek (pavel.bucek@oracle.com)
- Michal Gajdoš (michal@sapho.com)

Github repo: https://github.com/pavelbucek/placeholder

- Jersey – http://jersey.java.net
- Tyrus – http://tyrus.java.net

Q & A

JavaOne™ ORACLE

Integrated Cloud

Applications & Platform Services

JavaOne™

ORACLE®


ORACLE®