

From monolith to single-source to single-deployment



From monolith to single-source to single-deployment

Speaker



Dirk Fauth

*Research Engineer
Eclipse Committer*

ETAS GmbH
Borsigstraße 24
70469 Stuttgart

dirk.fauth@etas.com
www.etas.com

blog.vogella.com/author/fipro/
Twitter: [fipro78](#)

From monolith to single-source to single-deployment

A decorative red line with two white circular dots. The line starts at the first dot, goes down and to the right to the second dot, then continues horizontally to the right edge of the slide.

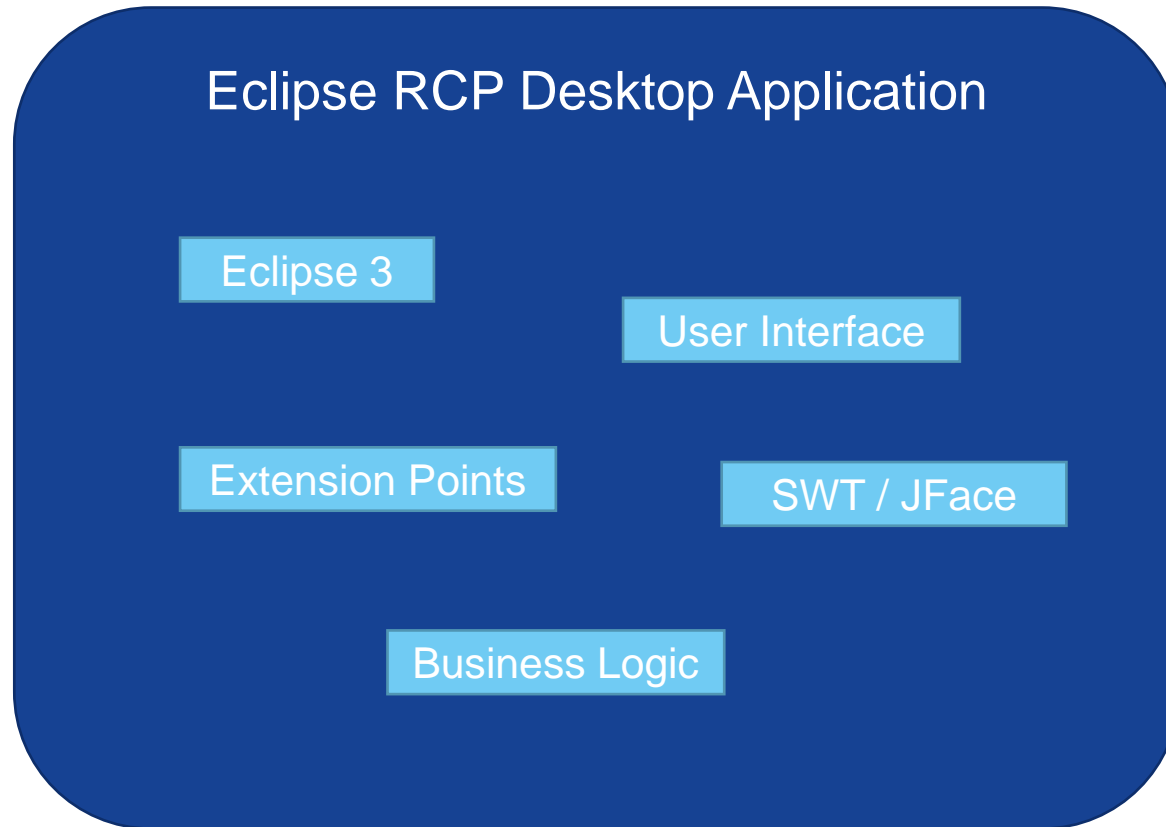
Overview

1. The Monolith
2. The Modulith
3. Single Source
4. Single Deployment
5. Conclusion

The Monolith



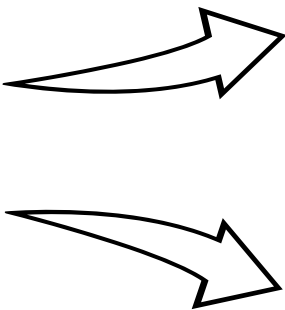
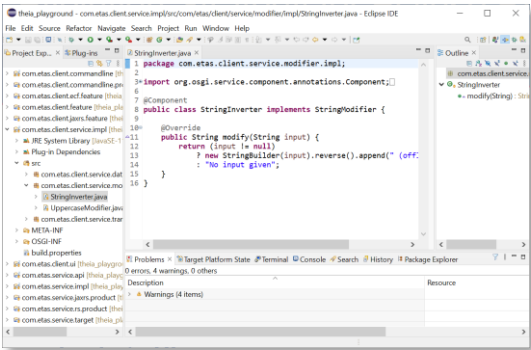
The Monolith



- Deployment monolith
- User interface and business logic not clearly separated
- Usage of “old” Eclipse 3.x Platform
 - Platform Singletons
 - Extension Points
 - Abstract Superclasses
- Tooling limited to Eclipse PDE

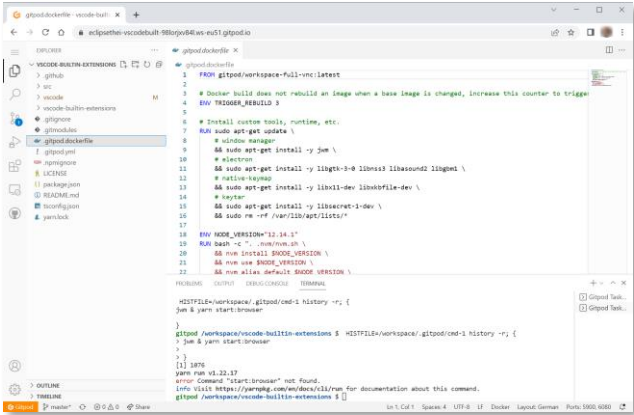
The Monolith Problems

Eclipse RCP Application

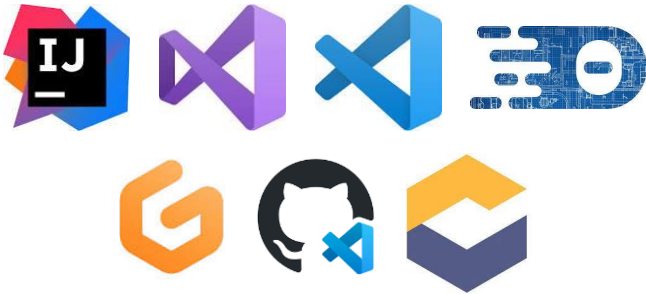
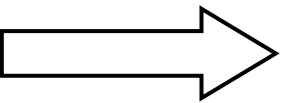


DevOps

	Declarative: Checkout SCM	prepare	migration	model processing	Model Simulation	Model Analysis	finalize
Average stage times (Average full run time ~2min 37s)	17s	2s	2s	175ms	2min 1s	5s	1s
100	14s	3s	2s	177ms	2min 2s	8s	2s
101	18s	2s	1s	161ms	2min 2s	8s	1s



Web-/Cloud-based Tooling



From monolith to single-source to single-deployment

Questions

- **New customer requirements**
 - DevOps
 - Web-/Cloud-based Tooling
 - Tools should be available faster and process faster
- **New developer requirements**
 - Freedom of choice regarding the IDE
 - Tools should be available faster and process faster

How to support new requirements by keeping the existing applications alive?

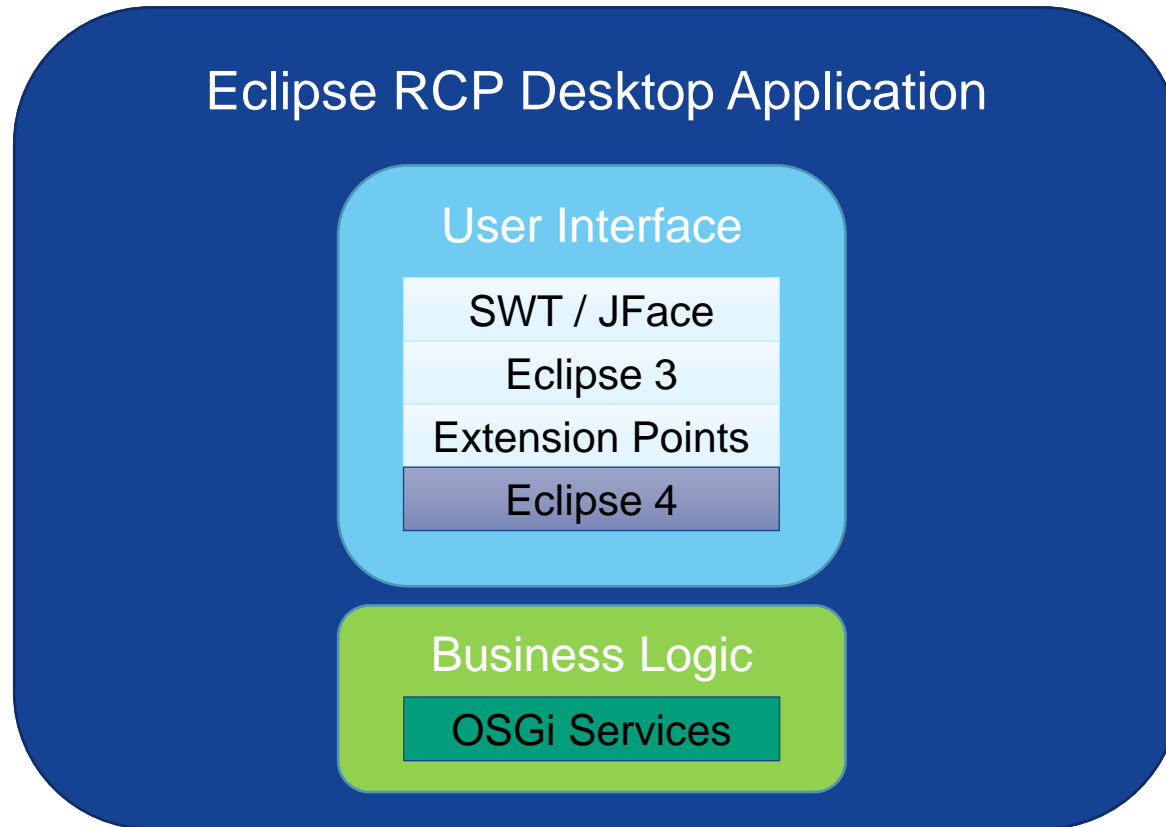
How to reuse existing functionality in different scenarios?

How can OSGi help in answering those questions?

The Modulith



The Monolith



- Deployment monolith
- Clearly defined and isolated modules
 - User Interface vs. Business Logic
- Eclipse 3.x for editors with navigator support
- Eclipse 4.x for business logic integration
- Business Logic encapsulated in OSGi Services

The Modulith

Refactoring

- **Extract business logic and provide it as OSGi Services**
 - Use OSGi Declarative Services
 - Access Services via Eclipse 4 Injection
- **Use plain Java/Maven or Java/Gradle project layout with bnd plugins**
 - Make development IDE agnostic
 - Add resulting bundles to Eclipse PDE Target Definition via ***m2e PDE Integration***

Mind the bundle activation policy!

A: Add the bundle to the auto-start configuration of the Eclipse launch configuration

B: Add the necessary header in the bundle via `@Header` annotation

```
@Header(name="Bundle-ActivationPolicy", value="lazy")
```

Note:

As a migration path you can also start with a PDE project layout and switch later. Remember that you are forced to use Eclipse as your IDE in that case.

The Modulith

OSGi Declarative Services & Eclipse 4 Service Injection

Service Interface

```
public interface StringModifier {  
    String modify(String input);  
}
```

Service Implementation

```
@Component  
public class StringInverter implements StringModifier {  
  
    @Override  
    public String modify(String input) {  
        return new StringBuilder(input)  
            .reverse().toString();  
    }  
}
```

Field injection in Eclipse 4

```
@Inject  
@Service  
StringModifier modifier;
```

The Modulith

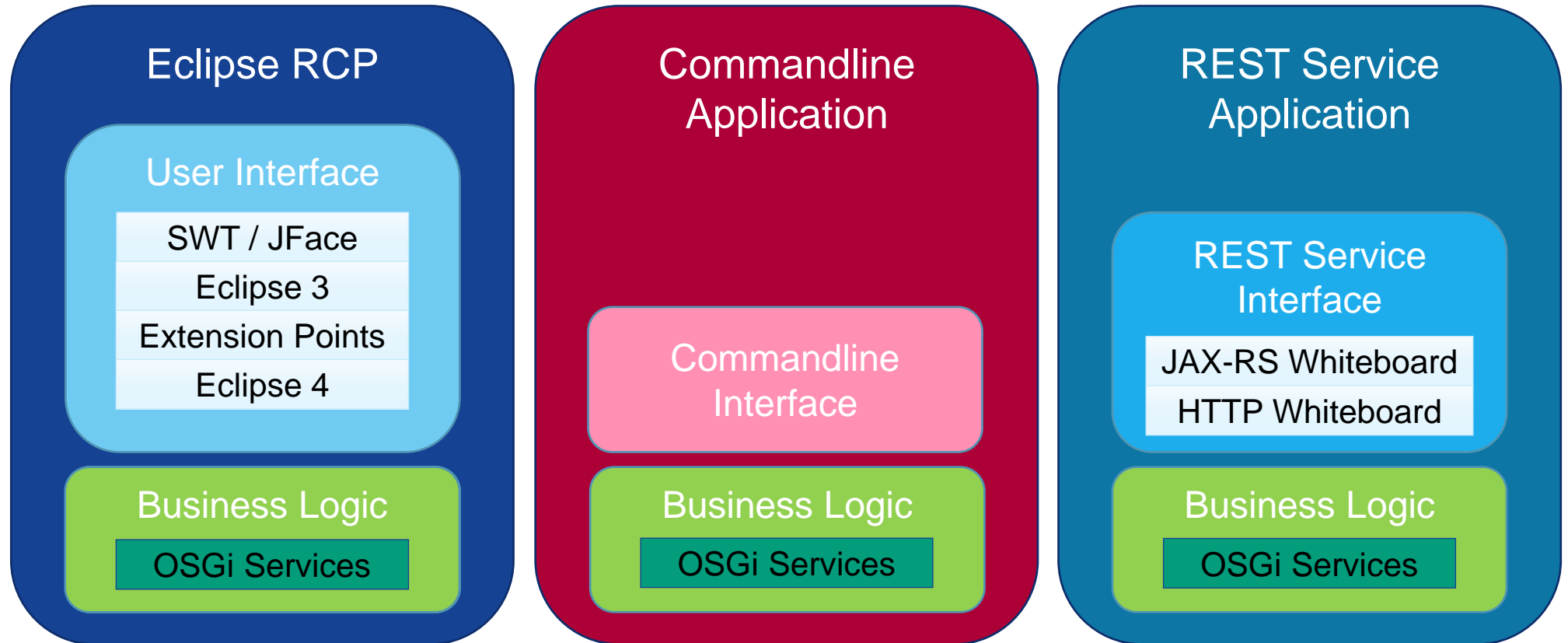
Why OSGi Declarative Services and Eclipse 4

- OSGi Declarative Services
 - Service oriented architecture
 - “state-of-the-art” programming model (dependency injection, annotation based)
 - No additional dependency (OSGi is the core of the Eclipse Platform)
 - Pre-requisite for further steps
 - Eclipse 4
 - “state-of-the-art” programming model (dependency injection, annotation based, POJOs)
 - Access OSGi Services via injection not via low-level OSGi API
- It is still Eclipse RCP with Eclipse SWT UI
- It is still a deployment monolith

Single Source



Single Source



Single Source

Multiple Applications from Single Source

- **Create new application types out of the existing services**
- **Minimal effort required**
 - Starter
 - Component Configuration
 - Mainly application setup with required dependencies
- **Used OSGi Specification Implementations**
 - Declarative Services
 - HTTP Whiteboard
 - JAX-RS Whiteboard

Single Source

Commandline Application – Starter Component

```
@Component(immediate = true)
public class BndStarter {
    String[] launcherArgs;

    @Reference(target = "(launcher.arguments=*)")
    void setLauncherArguments(Object object, Map<String, Object> map) {
        this.launcherArgs = (String[]) map.get("launcher.arguments");
    }

    @Reference
    StringModifier modifier;

    @Activate
    void activate() {
        // 1. inspect the cmd line args
        // 2. execute the logic
        // 3. shutdown
    }
}
```


Single Source

Service Implementation with JAX-RS annotations and JAX-RS Whiteboard configuration

```
@Path("/inverter")
@Component(
    immediate = true,
    property = { "osgi.jaxrs.resource=true" } )
public class StringInverter implements StringModifier {

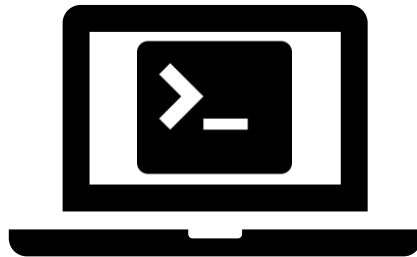
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    @Path("/{value}")
    @Override
    public String modify(@PathParam("value") String input) {
        return (input != null)
            ? new StringBuilder(input).reverse().toString()
            : "No input given";
    }
}
```

Single Source

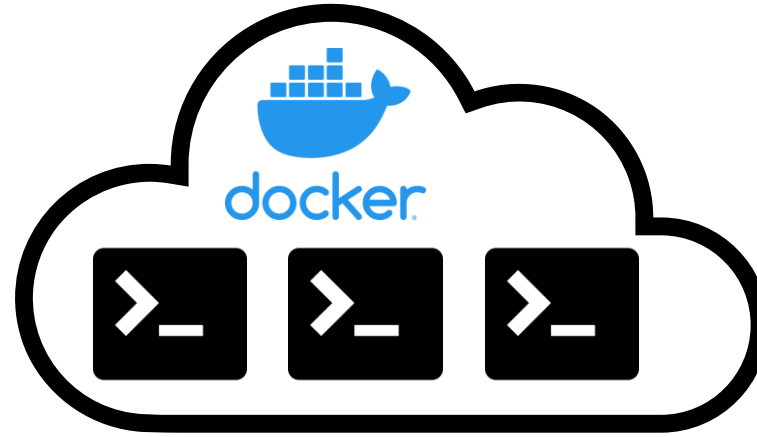
Use Cases - DevOps



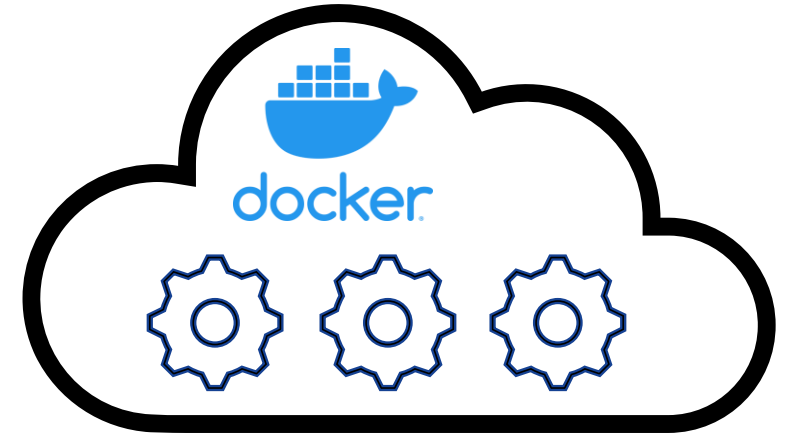
Local Rich Client



Local Commandline



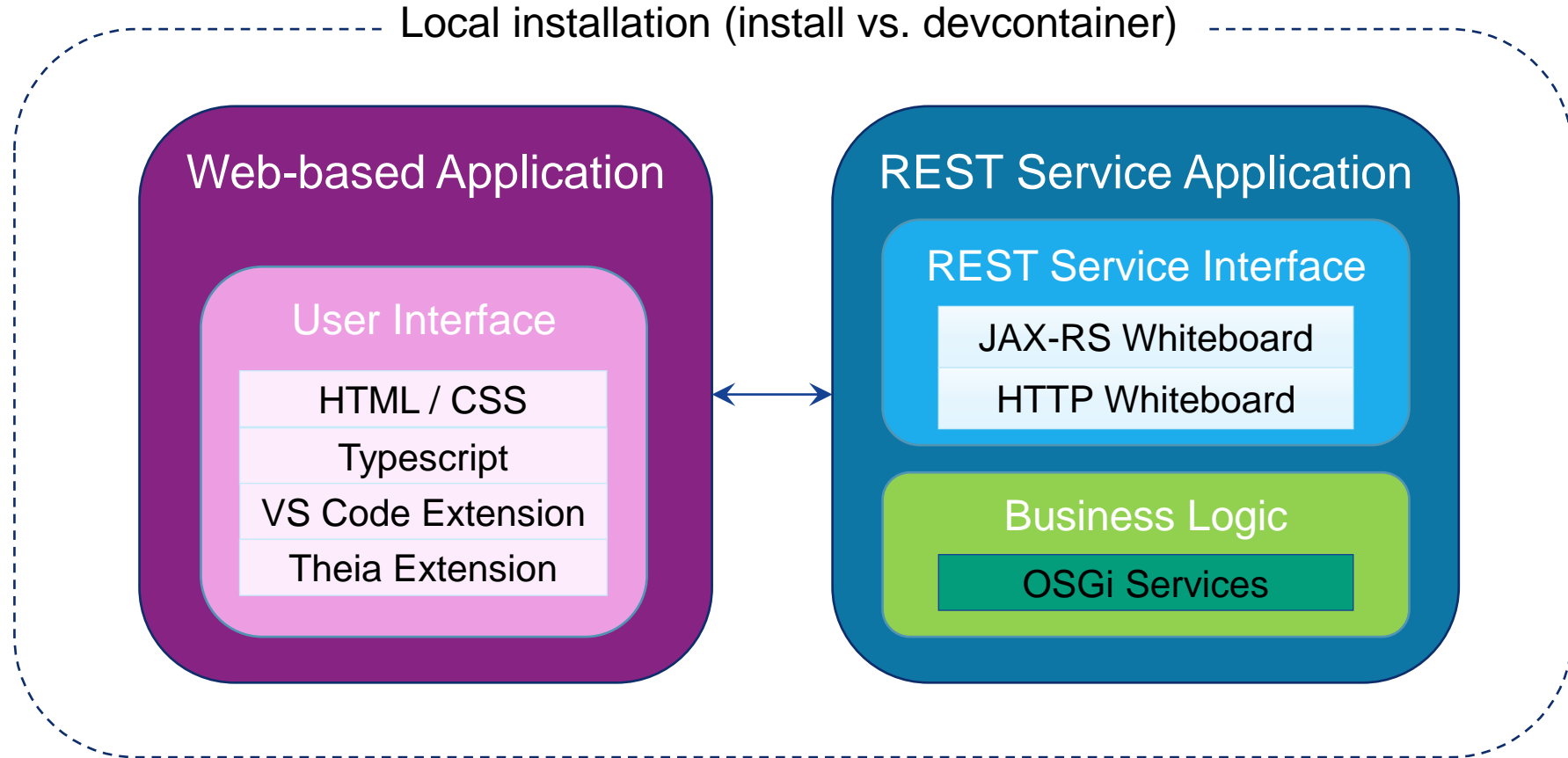
Commandline based Pipeline



REST API based Pipeline

Single Source

Web-based Application

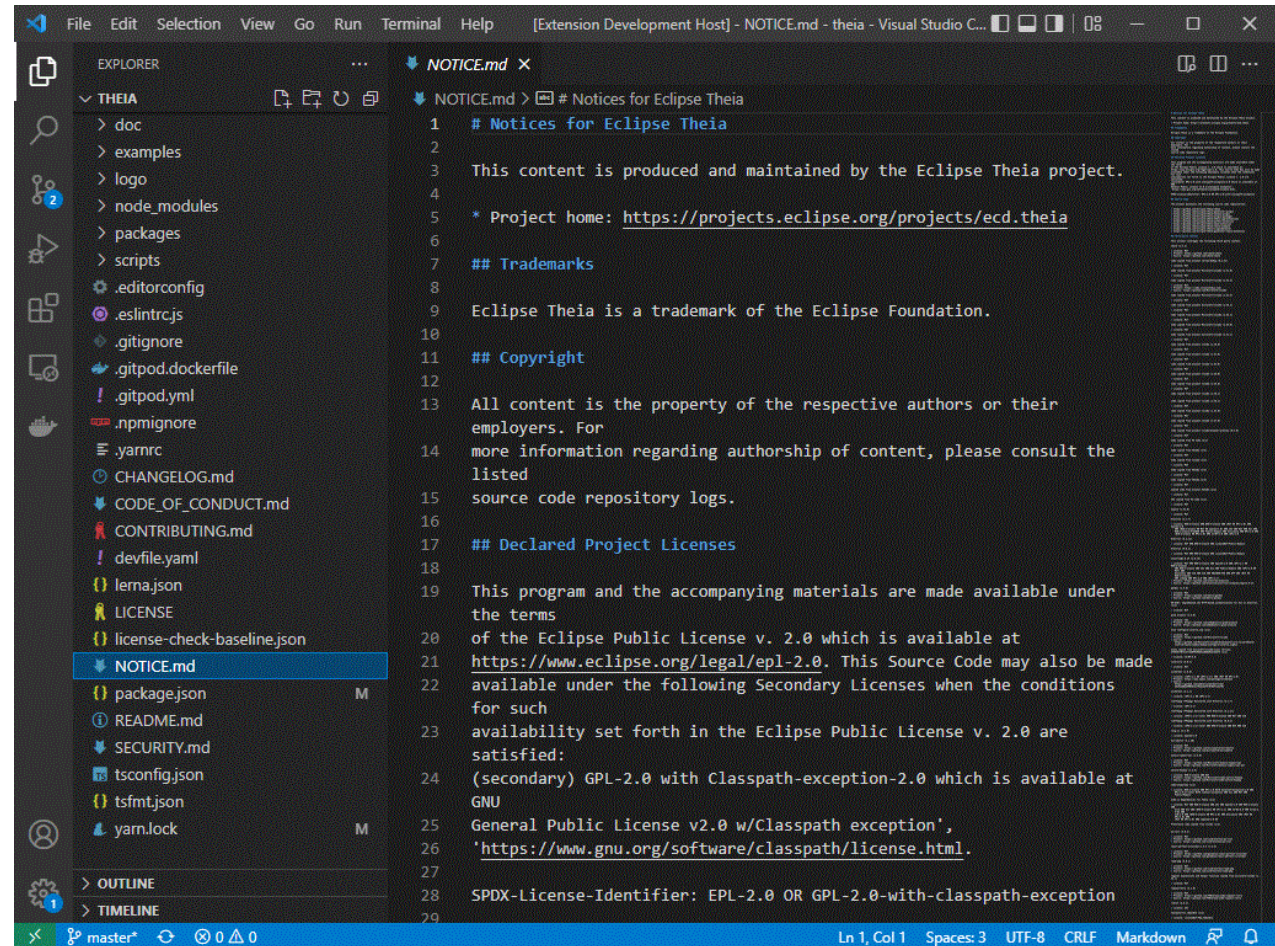


Single Source

VS Code Extension

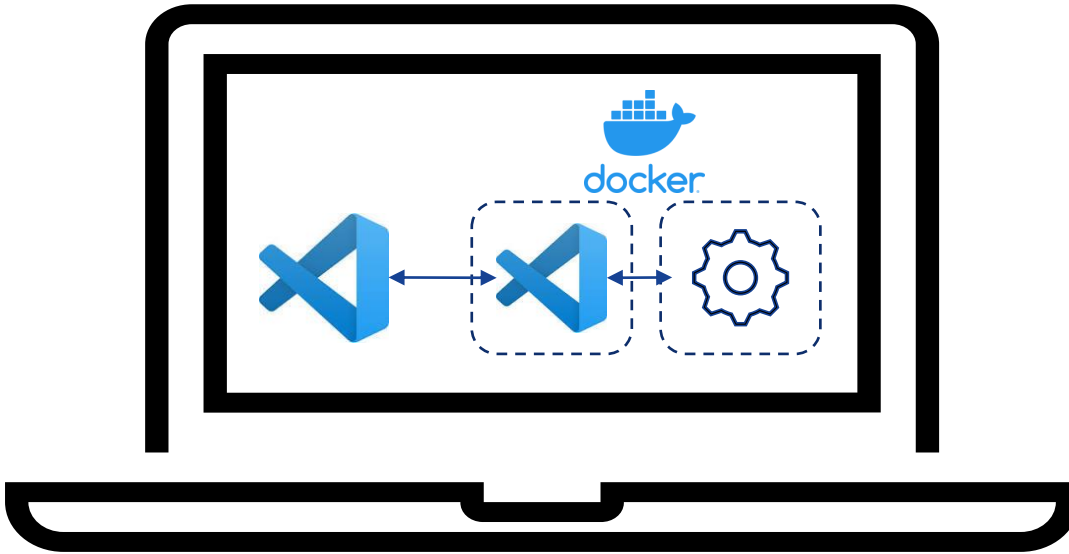
```
import fetch from 'cross-fetch';

async function modifyREST(input:string) {
  const response = await fetch (
    'http://localhost:8181/service/'
    + 'uppercase/modify/'
    + input);
  const data = await response.text();
  return data;
}
```



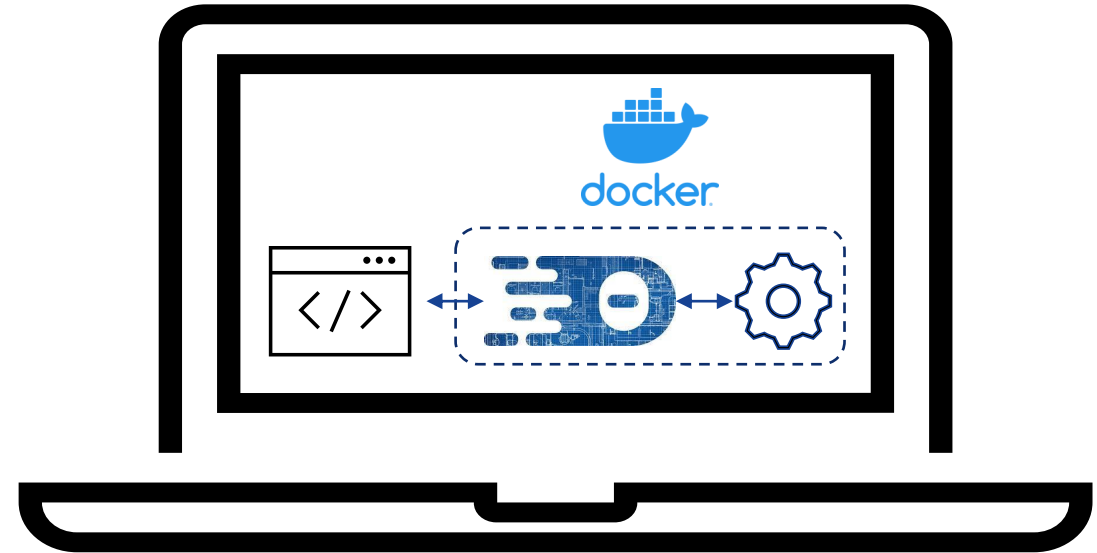
Single Source

Use Cases – Development Container



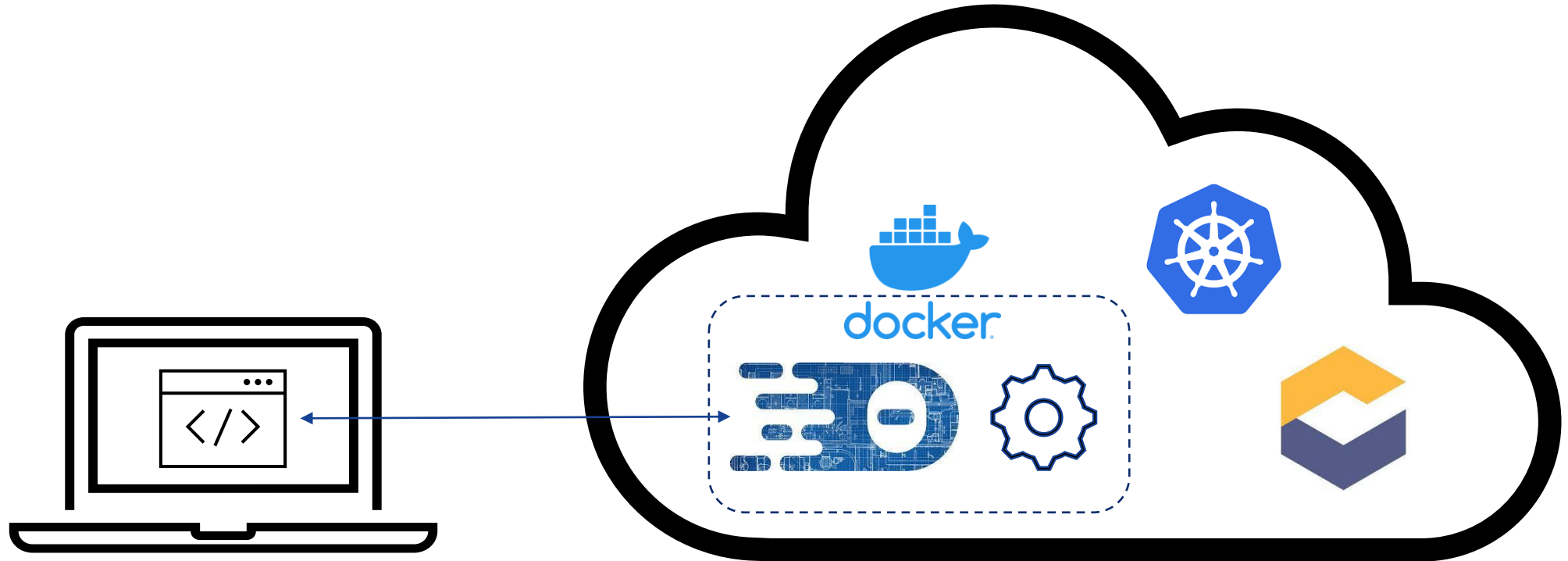
- VS Code
- Docker Compose
 - VS Code Remote Container
 - Service Container

- Browser
- Docker Container
 - Theia Application + Services



Single Source

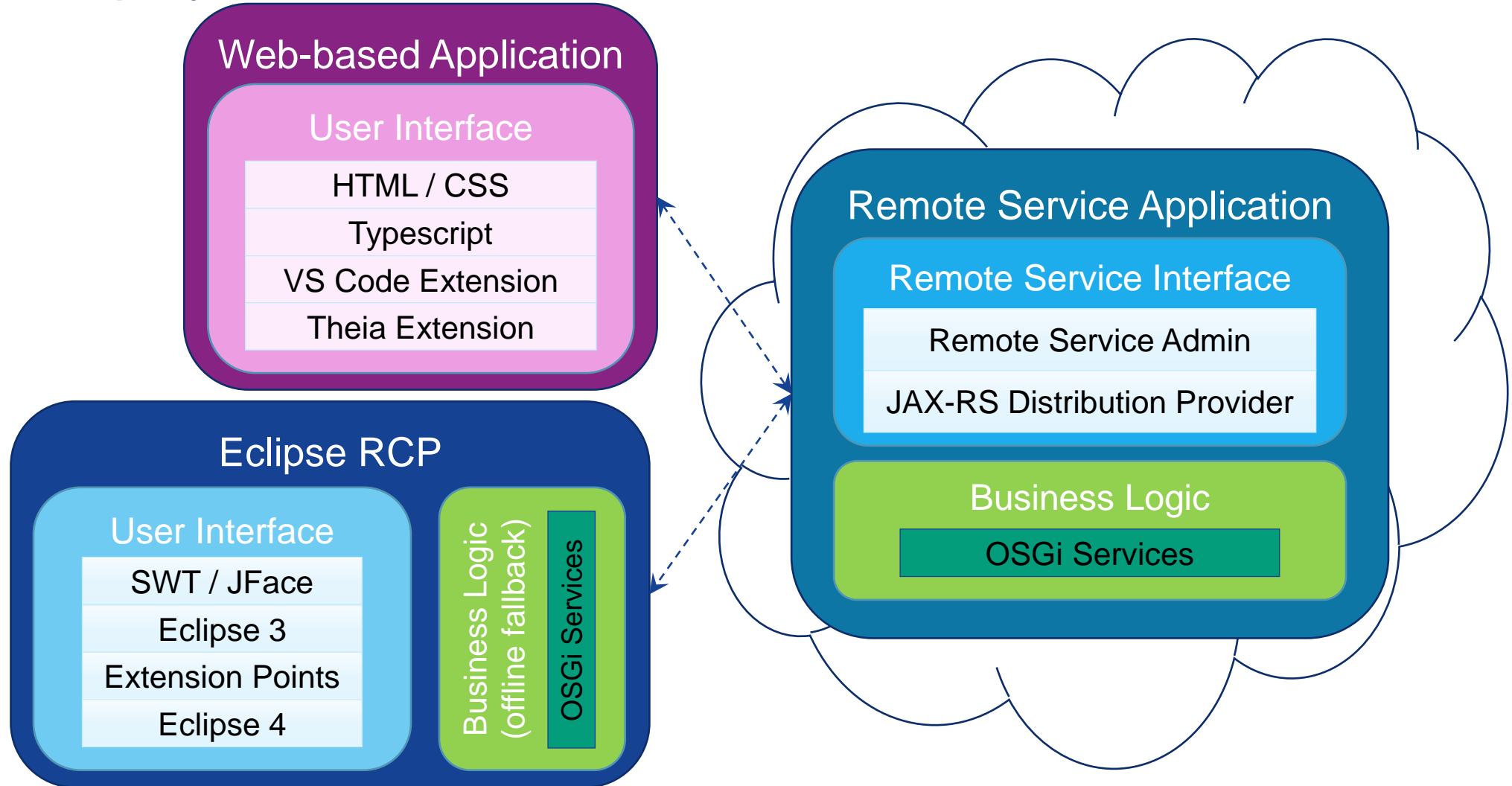
Use Case – Cloud IDE



Single Deployment



Single Deployment



Single Deployment

- **Create new application that provides services in different ways at the same time**
 - OSGi Remote Service
 - REST Service
- **Effort required**
 - Provider: Component Configuration and application setup
 - Consumer: Add required dependencies
- **Used OSGi Specification Implementations**
 - Declarative Services
 - Remote Services
 - Remote Service Admin
 - JAX-RS Whiteboard ???

incompatible



Aries JAX-RS Whiteboard
ECF JAX-RS Distribution Provider

Possible Solutions:

- Use different ECF Distribution Provider
- Use upcoming implementations in Eclipse namespace?

Single Deployment

Service Implementation with JAX-RS annotations and Remote Service configuration properties

```
@Path("/modify")
@Component(
    immediate = true,
    property = {
        "service.exported.interfaces=",
        "service.exported.intents=jaxrs",
        "ecf.jaxrs.server.pathPrefix=/inverter"})
public class StringInverter implements StringModifier {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    @Path("/{value}")
    @Override
    public String modify(@PathParam("value") String input) {
        return (input != null)
            ? new StringBuilder(input).reverse().toString()
            : "No input given";
    }
}
```

Single Deployment

ECF JAX-RS Distribution Provider Issues

ECF JAX-RS Remote Service

```
@Path("/modify")
@Component(
    immediate = true,
    property = {
        "service.exported.interfaces=",
        "service.exported.intents=jaxrs",
        "ecf.jaxrs.server.pathPrefix=/inverter"})
public class StringInverter implements StringModifier {
```

JAX-RS Whiteboard Service

```
@Path("/inverter")
@Component(
    immediate = true,
    property = { "osgi.jaxrs.resource=true" } )
public class StringInverter implements StringModifier {
```

JAX-RS Annotations

Needed in **interface** and **impl** so the client can create the proxy.

- Not possible to have multiple services of the same type available at different paths
- API bundles have dependencies to JAX-RS

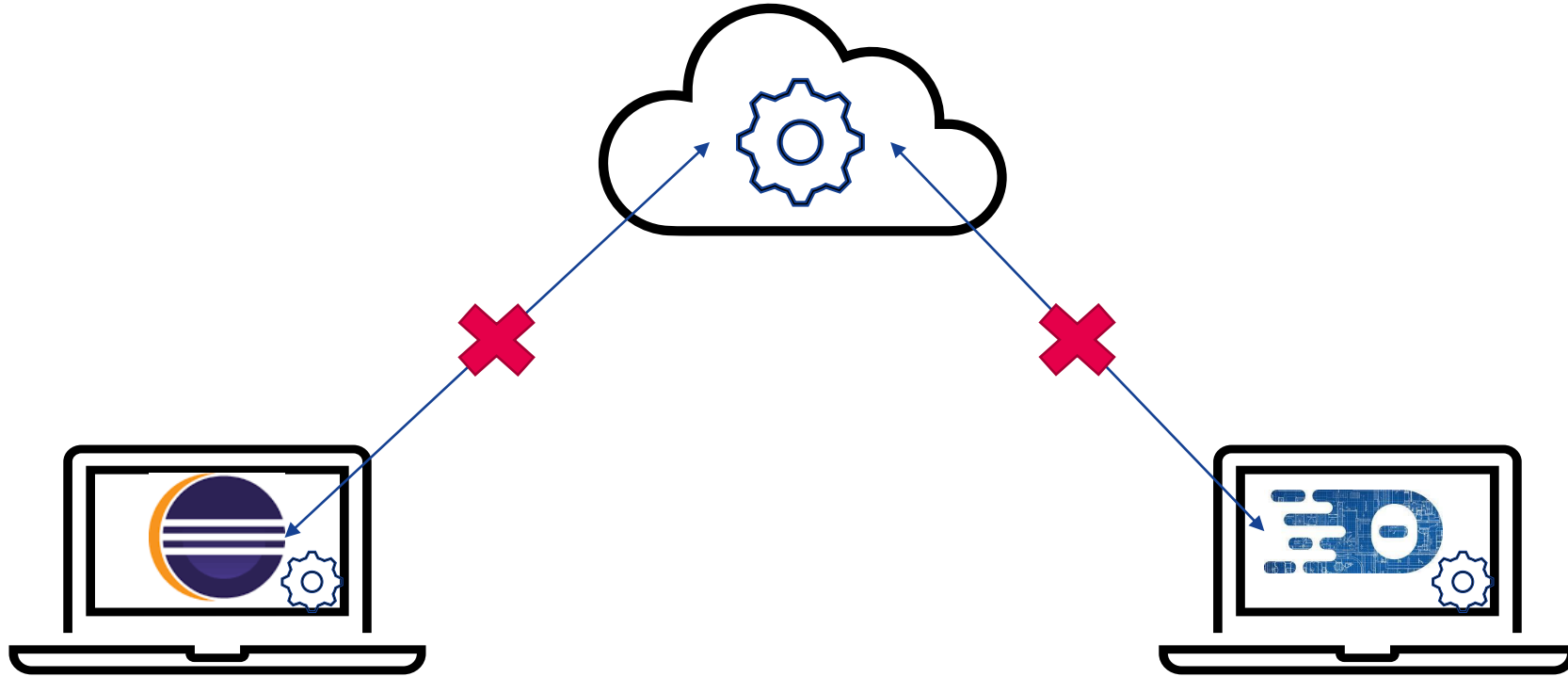
Maven Build

1. Transitive dependencies of ECF bundles not configured in pom.xml files (build via Tycho)
2. ECF JAX-RS Distribution Provider not available in Maven Central

→ Additional work needed on build setup

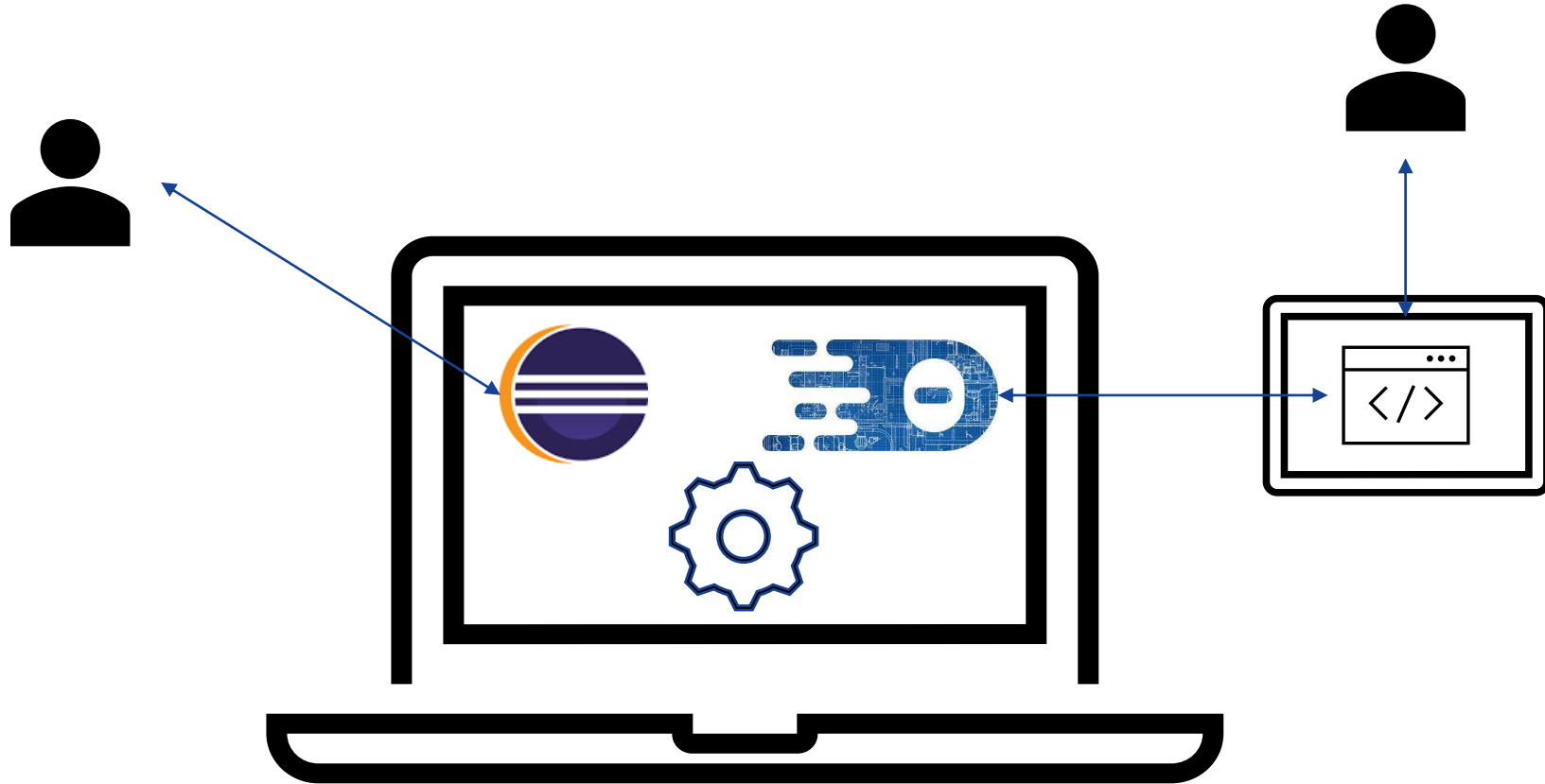
Single Deployment

Use Case – Cloud Service Deployment



Single Deployment

Use Case – Local Deployment with Remote Access



Conclusion

Conclusion

How to reuse existing functionality in different scenarios?

- **Extract business logic to services**

How to support new requirements by keeping the existing applications alive?

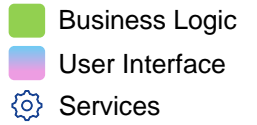
- **DevOps & Web-/Cloud-based tooling**
 - Build different applications using the same services
- **Freedom of choice regarding the IDE for developers**
 - Change the project layout

How can OSGi help in answering those questions?

- **Use OSGi Specifications and Implementations**
 - Declarative Services
 - HTTP Whiteboard
 - JAX-RS Whiteboard
 - Remote Services
 - Remote Service Admin

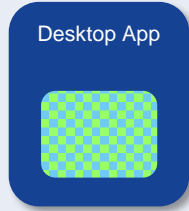
From monolith to single-source to single-deployment

Migration Path

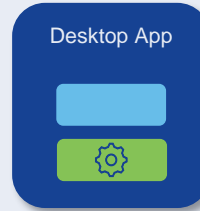


Focus: Reuse

(0) Monolith



(1) Modulith



- + Modularity
- + Testability
- + Service-Orientation



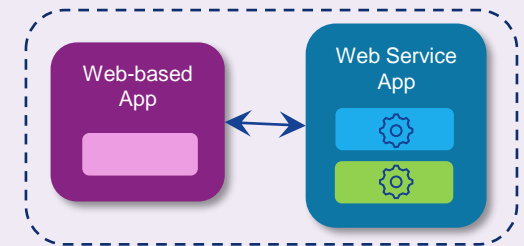
(2) Single Source



- + Automatization
- + Cloud-Processing



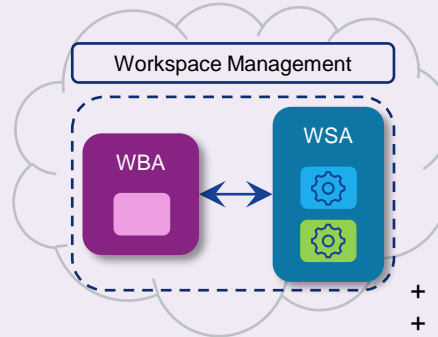
(3) Web-based



- + Modern UI
- + Fast setup



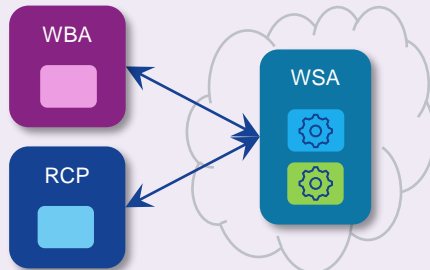
(4) Cloud-based



- + Instant setup
- + Shared workspace
- + No local installation required



(5) Single Deployment



- + Fast updates
- + Multiple access options
- + Online/Offline switches

Focus: Modernize

Further Information

– Blog Posts

- Getting Started with OSGi Declarative Services

<http://blog.vogella.com/2016/06/21/getting-started-with-osgi-declarative-services/>

- Building a “headless RCP” application with Tycho

<http://blog.vogella.com/2020/01/20/building-a-headless-rcp-application-with-tycho/>

- Build REST services with OSGi JAX-RS whiteboard

<http://blog.vogella.com/2021/01/08/build-rest-services-with-osgi-jax-rs-whiteboard/>

- Getting Started with OSGi Remote Services

<http://blog.vogella.com/2022/02/28/getting-started-with-osgi-remote-services-enroute-maven-archetype-edition/>

– Example

https://github.com/fipro78/monolith-single_deployment

Thank you

Dirk Fauth
ETAS/ENA
dirk.fauth@etas.com

