Lightweight Java in the Cloud

Shaun Smith
Oracle Application Container Cloud
shaun.smith@oracle.com
@shaunMsmith

Gerrit Grunwald
Java Technology Evangelist
gerrit.grunwald@oracle.com
@hansolo_

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.

Java ♥ Cloud

- Java SE is ideal for building lightweight applications and (micro)services
- Cloud platforms offer
  - High availability
  - Affordability
  - Ease of management
  - Access to supporting services like object storage, messaging, and databases

What do we mean by Cloud?

- A wonderfully amorphous word!
- Clouds offer a range of services including Iaas / SaaS / PaaS

- Application Platform as a Service—**APaaS**
  - Focused on running lightweight applications—not heavy weight services like database
  - Typically employ containers to run applications
  - Many vendors to choose from
  - Most, if not all, APaaS platforms support Java

Cloud Application Architecture

Configuration

Dev

Test

Prod

Configuration

- Application should be “parameterizable”—externalize all config data
- Configuration should be defined as environment variables and read by applications on start
- Makes it easy to deploy the application to any environment where configurations will differ
- Deployment configuration can be managed by Ops post Dev

This includes database and other backing service locations, third-party credentials like AWS or Twitter…such configuration should not be stored in the codebase…**exposes private resources in the version control** system.

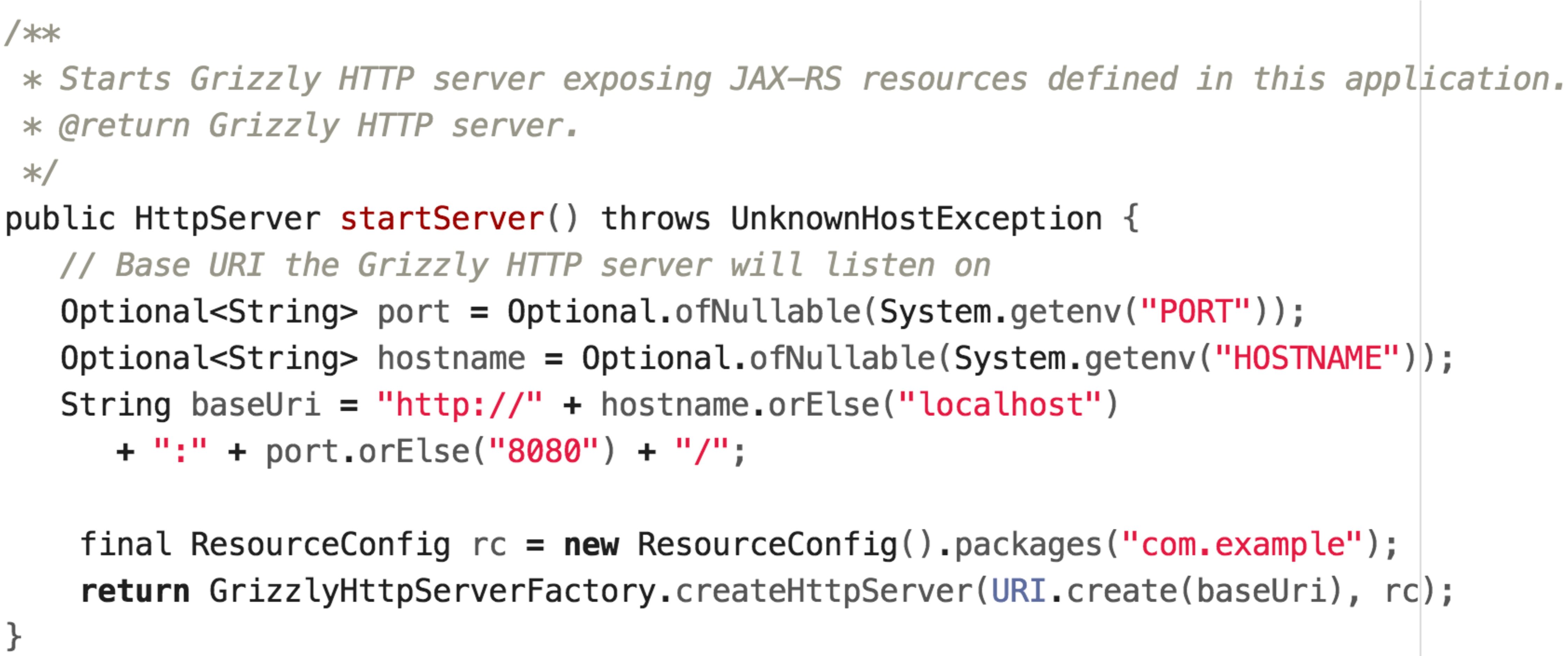
heroku

Environment variables…useful when the values are likely to be different from one host system to the next…makes the application code more portable and flexible…**critical for writing applications that are easily deployed and scaled**

OPENSHIFT®
by Red Hat®

JavaOne™ ORACLE®

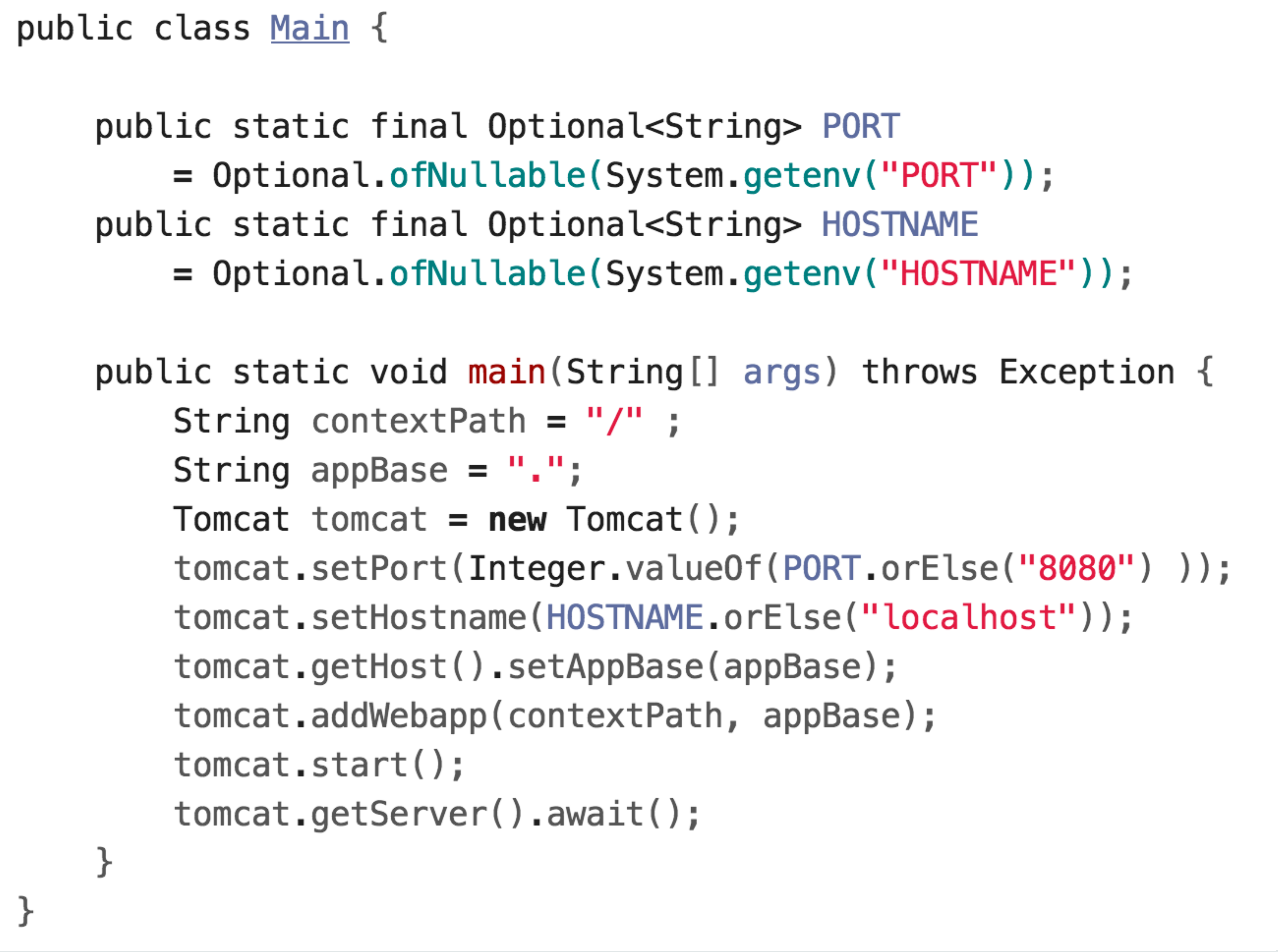
Grizzly / Jersey



```
/**
 * Starts Grizzly HTTP server exposing JAX-RS resources defined in this application.
 * @return Grizzly HTTP server.
 */
public HttpServer startServer() throws UnknownHostException {
   // Base URI the Grizzly HTTP server will listen on
   Optional<String> port = Optional.ofNullable(System.getenv("PORT"));
   Optional<String> hostname = Optional.ofNullable(System.getenv("HOSTNAME"));
   String baseUri = "http://" + hostname.orElse("localhost")
      + ":" + port.orElse("8080") + "/";

    final ResourceConfig rc = new ResourceConfig().packages("com.example");
    return GrizzlyHttpServerFactory.createHttpServer(URI.create(baseUri), rc);
}
```

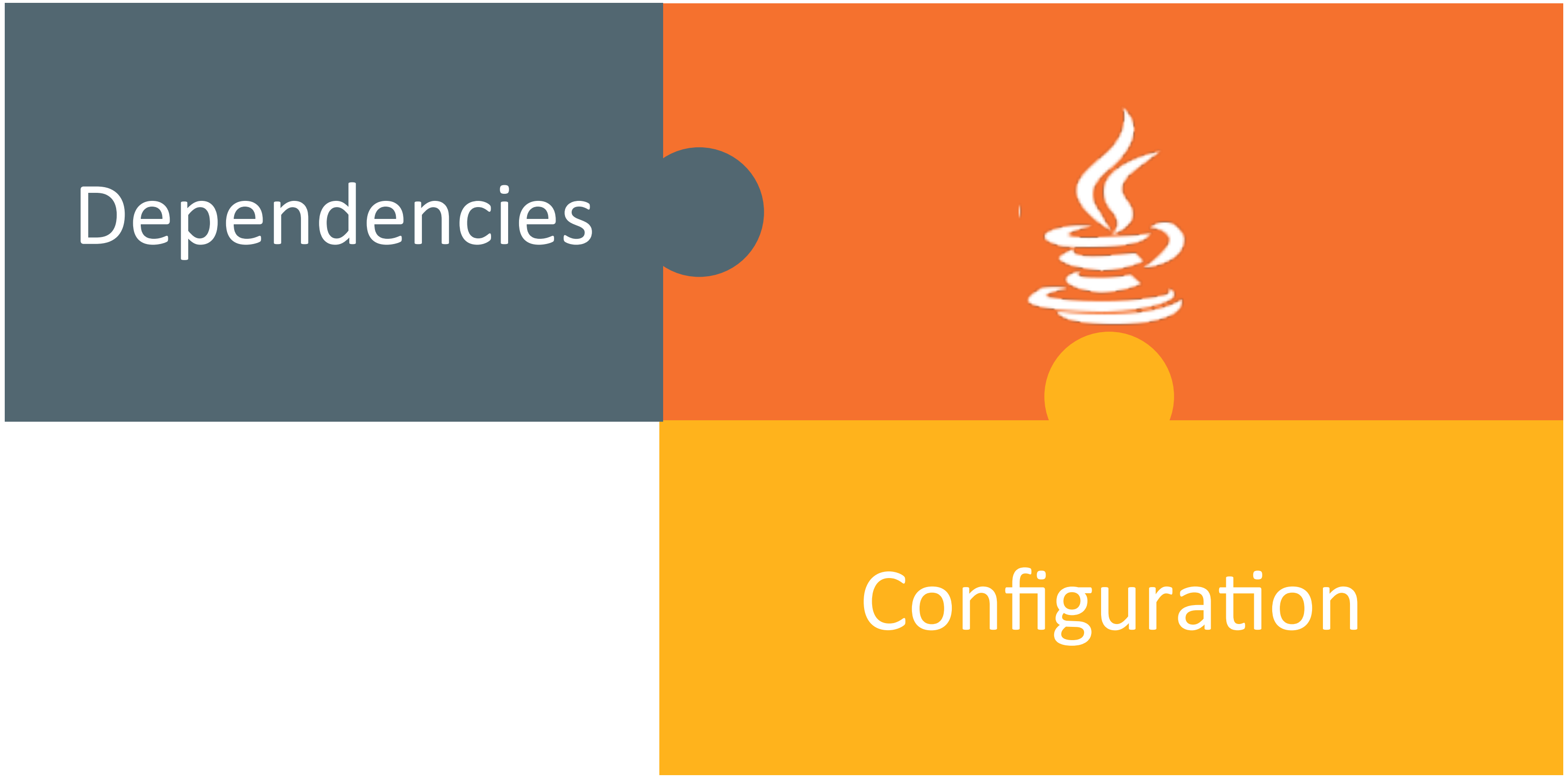
Embedded Tomcat



```
public class Main {

    public static final Optional<String> PORT
        = Optional.ofNullable(System.getenv("PORT"));
    public static final Optional<String> HOSTNAME
        = Optional.ofNullable(System.getenv("HOSTNAME"));

    public static void main(String[] args) throws Exception {
        String contextPath = "/" ;
        String appBase = ".";
        Tomcat tomcat = new Tomcat();
        tomcat.setPort(Integer.valueOf(PORT.orElse("8080") ));
        tomcat.setHostname(HOSTNAME.orElse("localhost"));
        tomcat.getHost().setAppBase(appBase);
        tomcat.addWebapp(contextPath, appBase);
        tomcat.start();
        tomcat.getServer().await();
    }
}
```

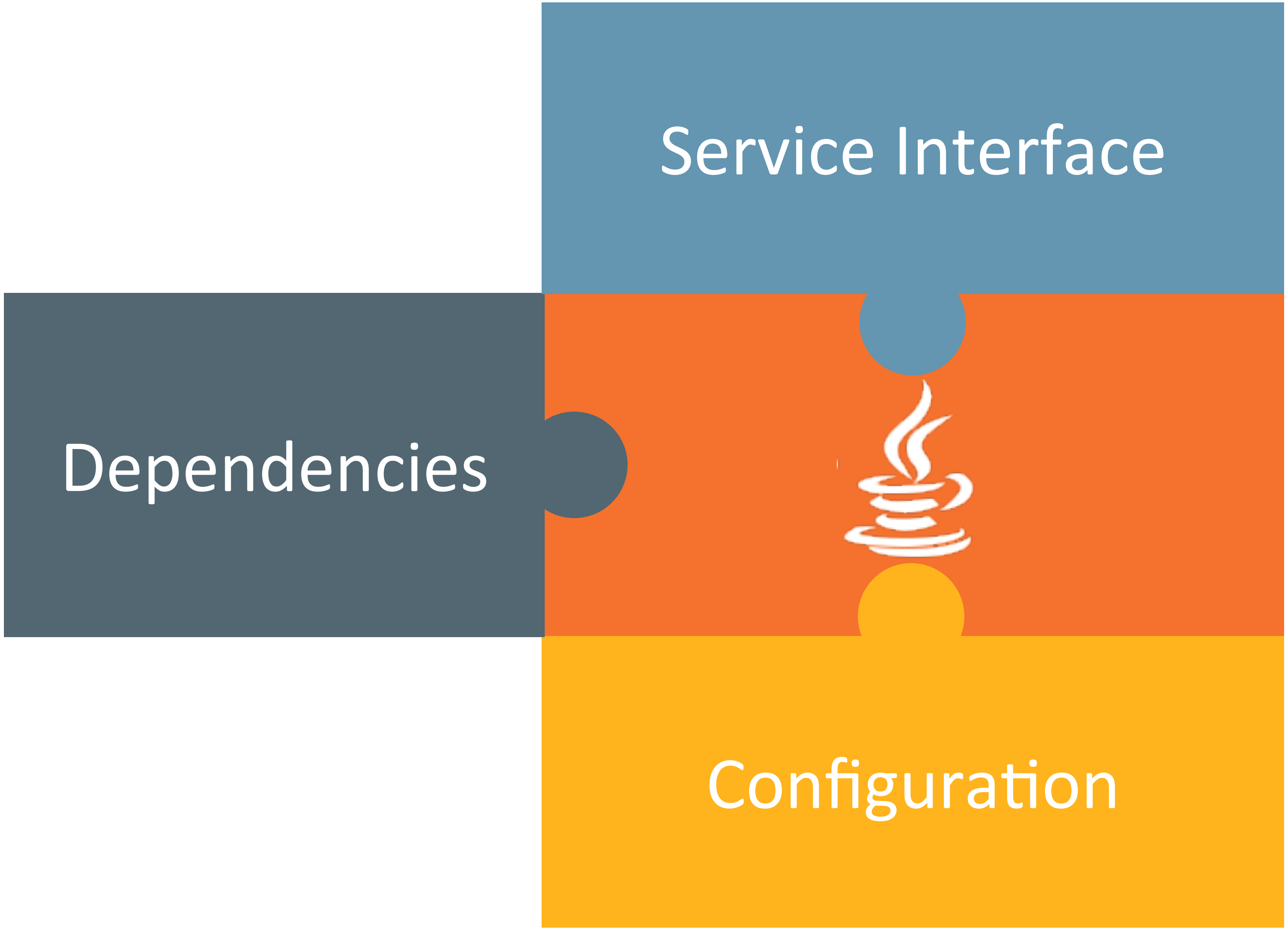


Dependencies

Configuration

Dependencies

- Application dependencies must be explicitly declared
- Relying on the availability of **system wide libraries** in the runtime environment can lead problems that are to hard to diagnose
- Applications with explicit dependencies are easy to move between environments
- Maven & Gradle provide a nice way to declare application dependencies



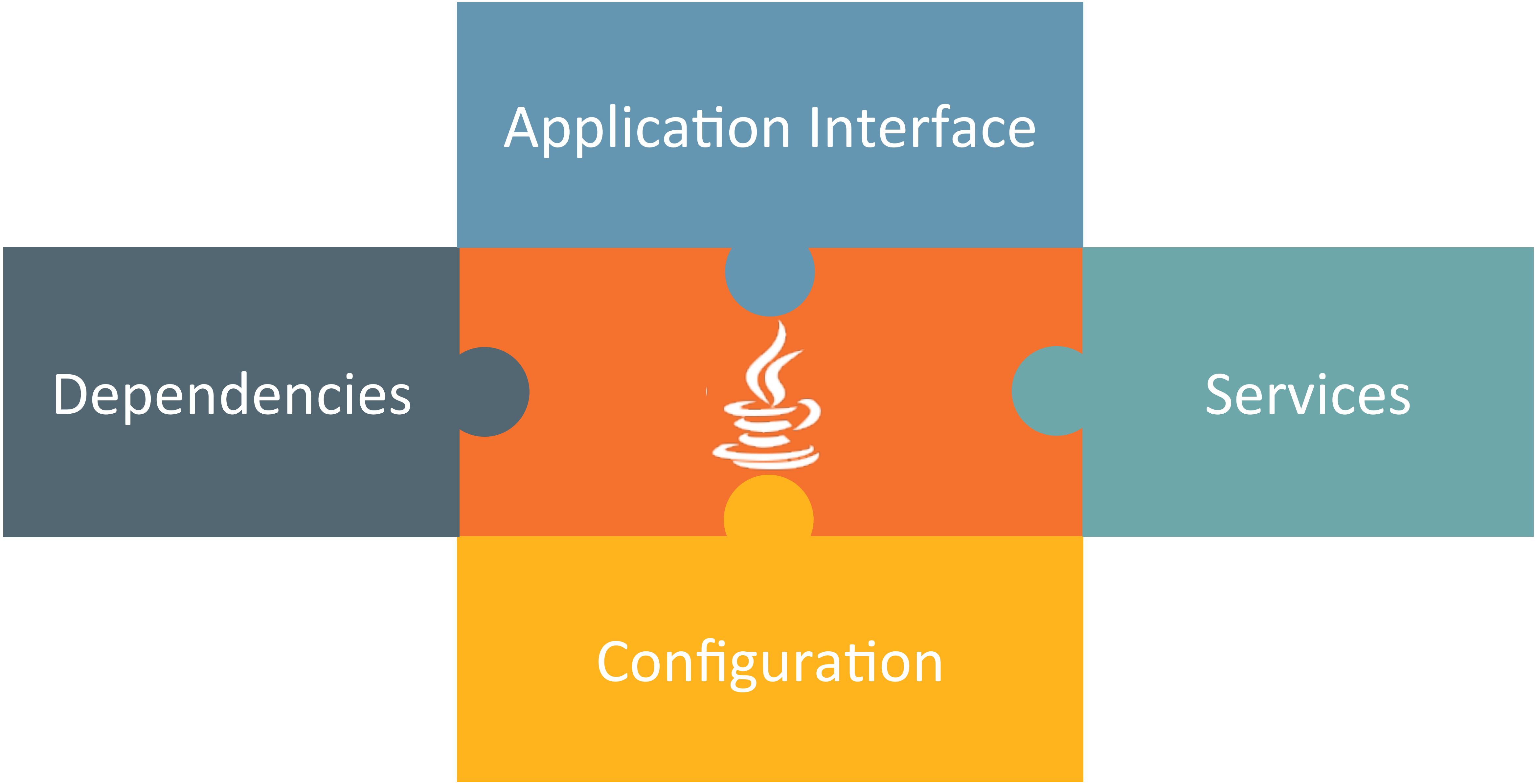
Service Interface

Dependencies

Configuration

Service Interface

- Cloud applications typically expose Web/REST/Websocket services using embedded servers like Tomcat, Jetty, or Grizzly/Jersey/Tyrus.
- Embedded servers must bind to the hostname and port defined by the runtime environment, typically in an environment variable like $PORT
- Applications running in containers will likely bind to a local port that is receiving load balanced traffic forwarded from a pubic port
- By exposing functionality over REST, applications can be provide services to other applications (the core of the microservices approach)



Application Interface

Dependencies

Services

Configuration

Service Dependencies

- Service coordinates published in the runtime environment for consumption by application
- Backing services should be pluggable
- Easy to change services—restart application to pick up changes
- Treat 3rd party or platform services the same as your own services

DEMO

Cloud Application Qualities

Stateless / Disposable

- Applications should be stateless with all persistent data stored in external services like database or key/value stores
- Stateless applications makes scaling easy (esp. when scaling in by disposing of instances)
- Configuration changes will result in the restarting (disposing & creating) of application instances
- Ephemeral disk is useful but there is no guarantee a subsequent request will be handled by the same instance

Share Nothing

- Ideally, applications should be stateless and share-nothing
- Horizontal scaling of stateless applications by adding instances to handle increased load is simple and reliable
- Not everyone agrees on this point…

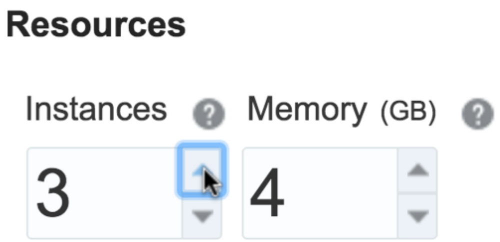
Why We Aren’t Joining the Cloud Foundry Foundation

…we are also working hard to support both stateless and stateful applications inside those containers…

OPENSHIFT®
by Red Hat®

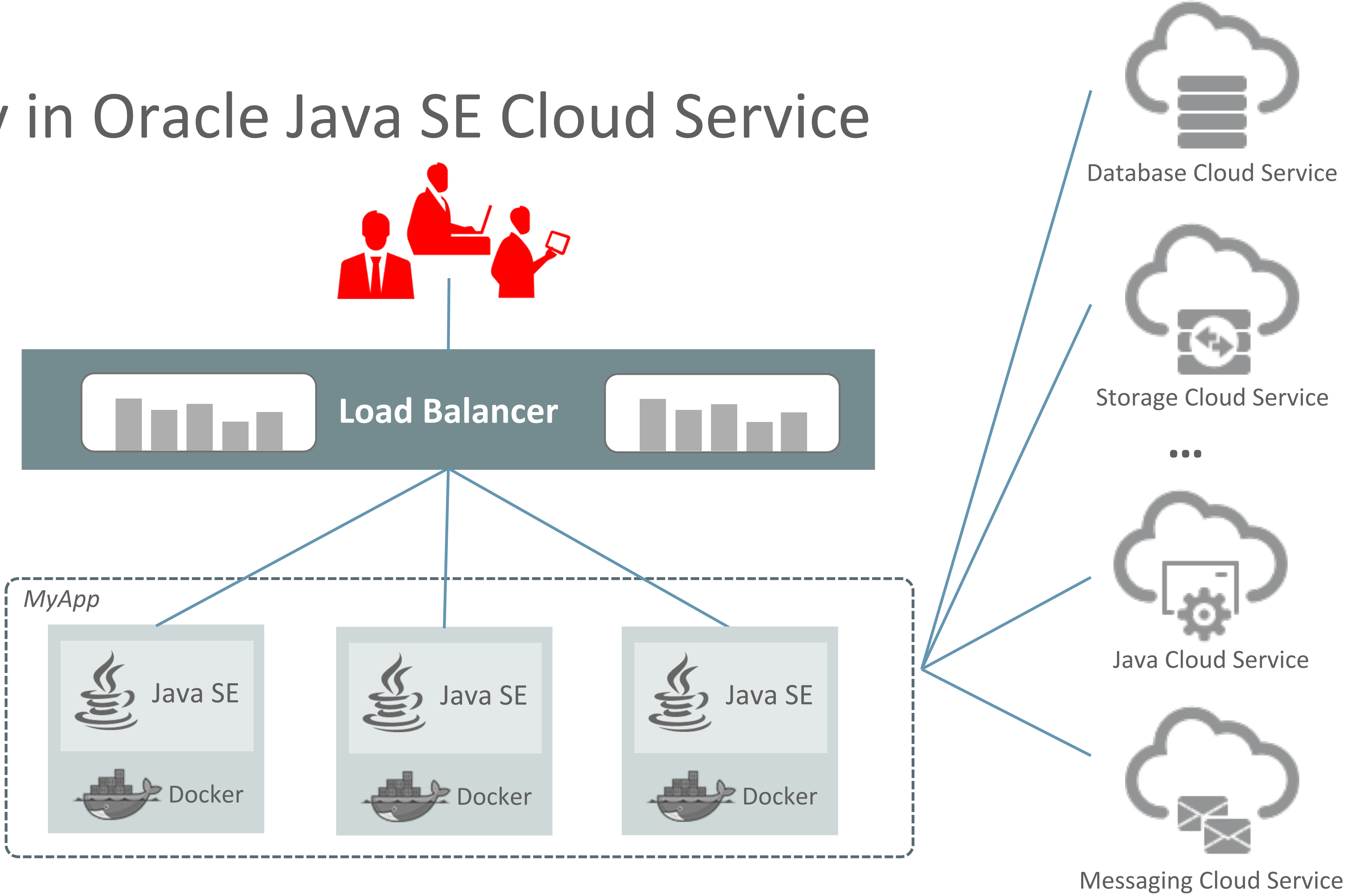
JavaOne™
ORACLE®

Concurrency in Oracle Java SE Cloud Service



Resources

Instances	Memory (GB)
3	4



Practical Matters

Development vs. Deployment

- APaaS platforms are deployment platforms—you run apps on them
- Development happens where?
- How to develop and debug applications targeting APaaS?

Java APaaS Debug Options

Debug locally in emulated environment

- Define environment variables to match target
- Use same JDK release and version as in target
- Use tools like Foreman
- Challenge—”emulated” not identical

Debug in Cloud

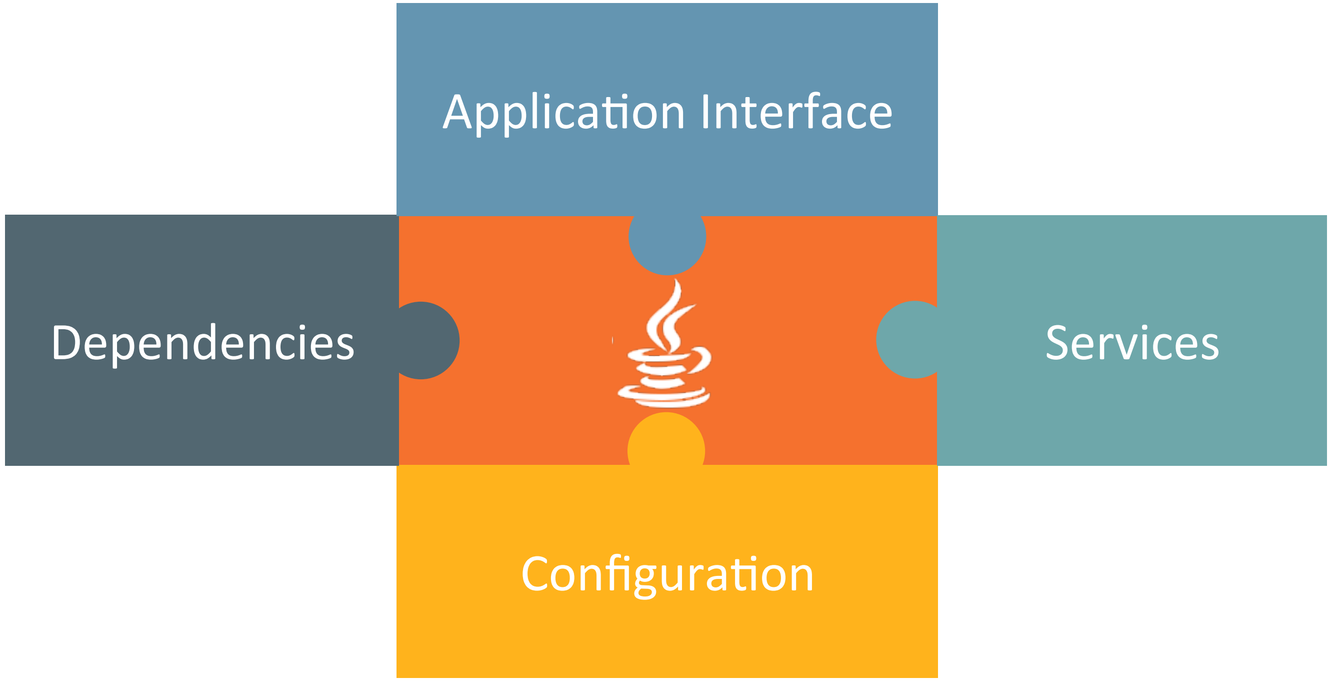
- Restrict to single instance
- Remote test & debug
- E.g., Spring Tools (experimental afaik)
- Challenge—opening up ports and supporting breakpoint callbacks from cloud to desktop

Debug locally using same Docker image as on APaaS

- Same runtime environment as Cloud
- Challenge—dependency on cloud services
- E.g. Heroku (not exactly the same since not Docker in cloud)

Summary

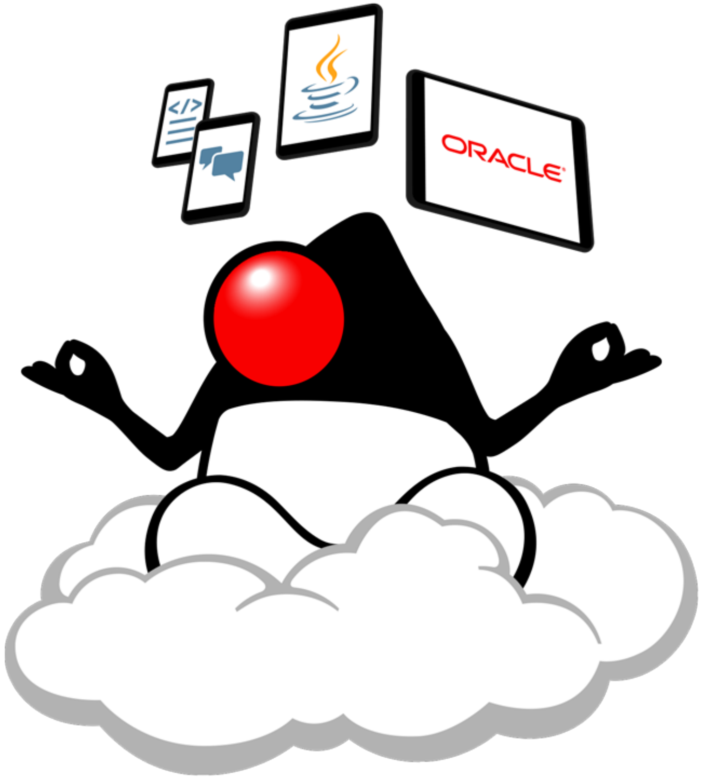
- Java is one of the best choices for Cloud application development
- Java has broad support from APaaS vendors
- Most of the architectural principals of Cloud apps are simply good practice
  - – Configurable
  - – Declared Dependencies
  - – Pluggable Service
  - – …



Copyright © 2015, Oracle and/or its affiliates. All rights reserved. |

Java is the Platform!

- Focus on your Java application and not on proprietary platform features
- Architect for platform independence
- Own the stack—don't let vendors dictate
- Use embedded servers, not containers to stay light
- Consider the microservices approach of many small services assembled to provide a complete solution



Integrated Cloud

Applications & Platform Services