Build a 12 factor microservice in half an hour

Emily Jiang: Liberty Architect for MicroProfile and CDI, IBM

@emilyfhjiang

nts

oncept of 12 factor app

f creating a 12 factor microservice using MicroProfile

tors in a nut shell

A methodology

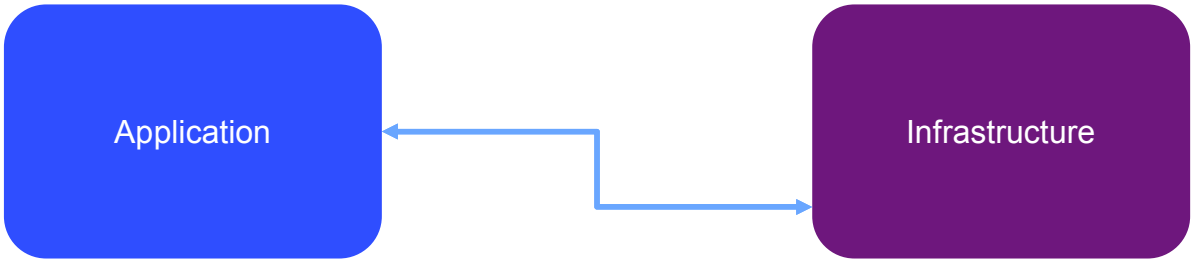
Best Practices

Manifesto

2factor.net/ by Heroku

2 factor?

fine the contract between applications and infrastructure



s a Twelve-Factor App?

ern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building s
pps that:

ative formats for setup automation, to minimize time and cost for new developers joining the project;

n contract with the underlying operating system, offering **maximum portability** between execution environments;

e for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;

divergence between development and production, enabling **continuous deployment** for maximum agility;

ale up without significant changes to tooling, architecture, or development practices.

-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queu

FACTORS

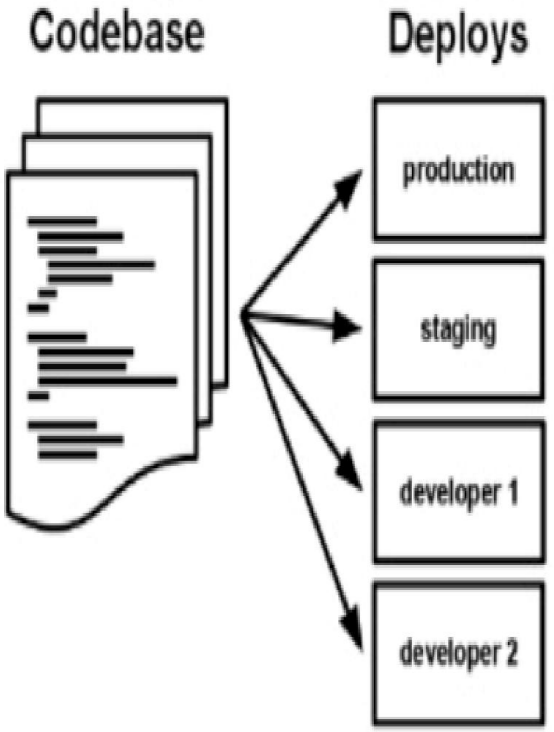
1. debase
2. pendencies
3. nfig
4. cking Services
5. ild, Release, Run
6. ocesses
7. Port binding
8. Concurrency
9. Disposability
10. Dev / Prod parity
11. Logs
12. Admin Processes

ebase

base tracked in revision control, many deploys.”

ate smaller teams to individual applications or microservices.

ving the discipline of single repository for an application forces the teams to analyze the of their application, and identify potential monoliths that should be split off into services.



single source code repository for a single application (1:1 relation).
ent stages are different tags/branches
.e. use a central git repo (external Github/GitHub Enterprise also suitable)

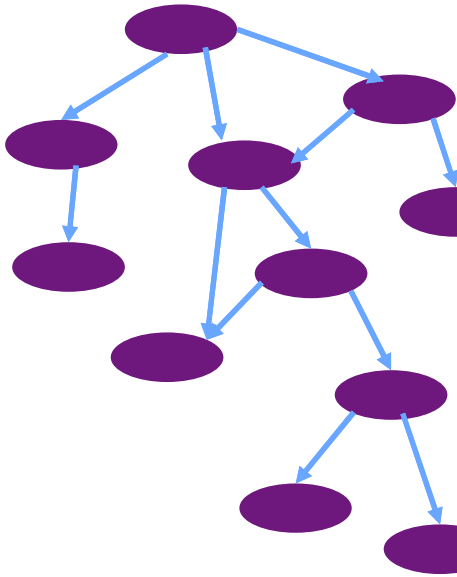
pendencies

ly declare and isolate dependencies”

d-native application does not rely on the pre-existence of dependencies in a deployment target.

per Tools declare and isolate dependencies

laven and Gradle for Java



microservice has its own dependencies declared (e.g. pom.xml)

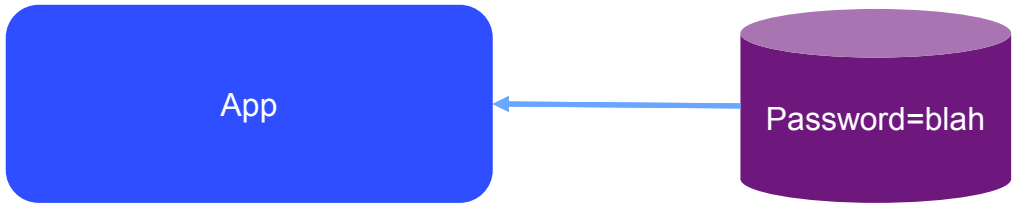
nfig

onfig in the environment”

ging config should not need to repackage your application

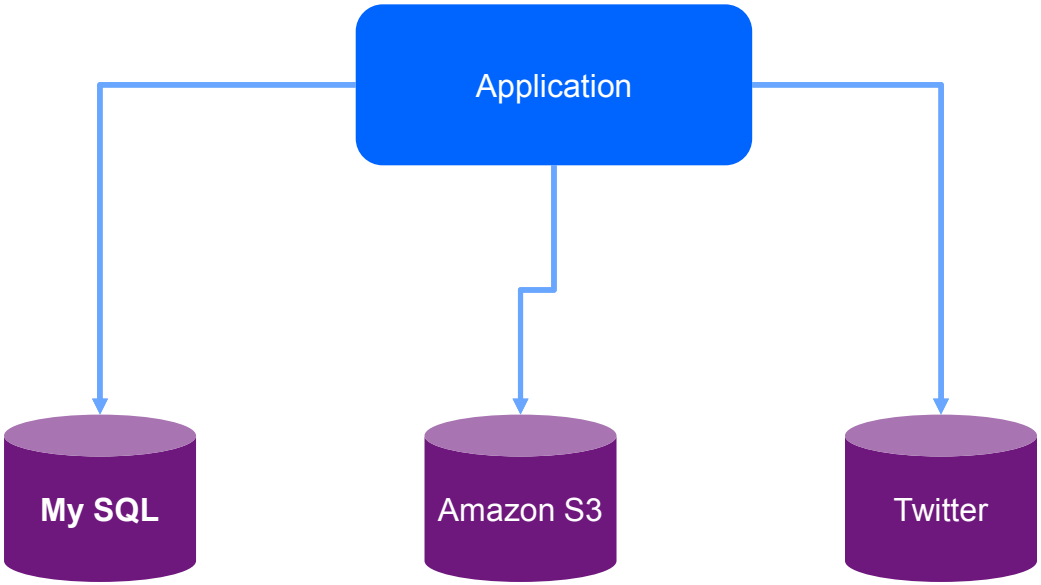
Kubernetes configmaps and secrets (rather than environment variables) for container services

MicroProfile Config to inject the config properties into the microservices



cking services

backing services as attached resources”



ld, release, run

y separate build and run stages”

ce code is used in the build stage. Configuration data is added to define a release stage that can be deployed
in code or config will result in a new build/release

s to be considered in CI pipeline

BM

- UrbanCode Deploy
- IBM Cloud Continuous Delivery Service

AWS

- AWS CodeBuild
- AWS CodeDeploy
- AWS CodePipeline (not yet integrated with EKS)

Azure

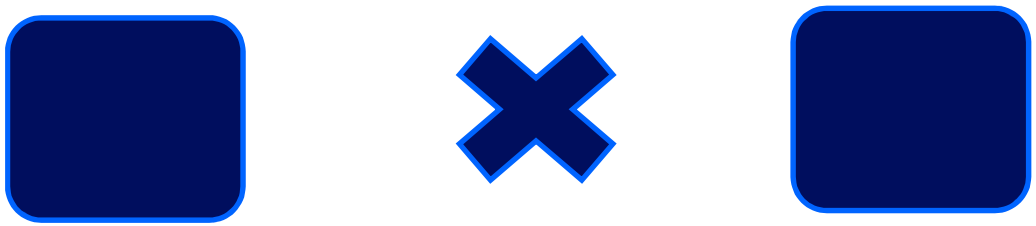
- Visual Studio Team Services (VSTS) (includes git)
- Web App for Containers feature of Azure App Service

ocesses

e the app as one or more stateless processes”

tateless and share-nothing

est API



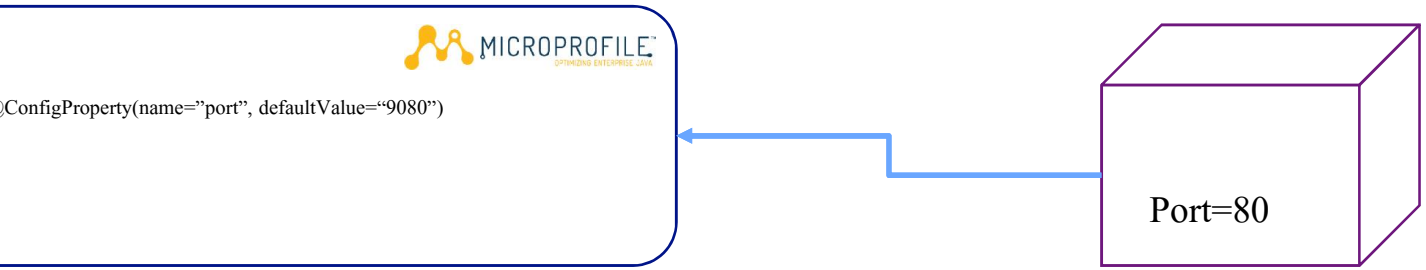
ort binding

services via port binding”

cations are fully self-contained and expose services only through ports. Port assignment is done by the exe
ment

ss/service definition of k8s manages mapping of ports

MP Config to inject ports to microservices for chain-up invocations



Concurrency

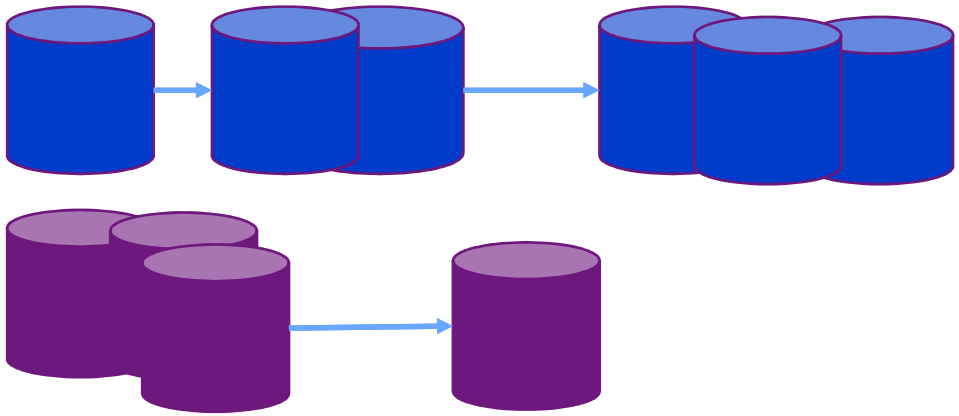
out via the process model”

cations use processes independent from each other to scale out (allowing for load balancing)

e considered in application design

d autoscaling services: [auto]scaling built into k8s

micorservices



sposability

ize robustness with fast startup and graceful shutdown”

sses start up fast.

sses shut down gracefully when requested.

sses are robust against sudden death

MicroProfile Fault Tolerance to make it resilient

Service Model

•Pets are given names like pussinboots.cern.ch
•They are unique, lovingly hand raised and cared for
•When they get ill, you nurse them back to health

•Cattle are given numbers like vm0042.cern.ch
•They are almost identical to other cattle
•When they get ill, you get another one

•Future application architectures should use Cattle but Pets with strong configuration management are viable and still needed

From “*CERN Data Centre Evolution*”

v/prod parity

evelopment, staging, and production as similar as possible”

lopment and production are as close as possible (in terms of code, people, and environments)

se helm to deploy in repeatable manner

name)spaces for isolation of similar setups

gs

ogs as event streams”

writes all logs to stdout

structured output for meaningful logs suitable for analysis. Execution environment handles routing and an

ucture

dmin processes

min/management tasks as one-off processes”

ng: standard k8s tooling like “kubectl exec” or Kubernetes Jobs

to be considered in solution/application design

xample, if an application needs to migrate data into a database, place this task into a separate component in
t to the main application code at startup

ACTORS

debase

pendencies

nfig 

cking Services

ild, Release, Run

ocesses

7. Port binding 

8. Concurrency
9. Disposability 

10. Dev / Prod parity
11. Logs
12. Admin Processes

Profile Config

gure Microservice without repacking the application

y the configuration in configure sources

- Access configuration via
  - Programmatically lookup

```
Config config =ConfigProvider.getConfig(

config.getValue("myProp", String.class);
```

  - Via CDI Injection

```
@Inject
@ConfigProperty(name="my.string.property
String myPropV;
```

ofile Config

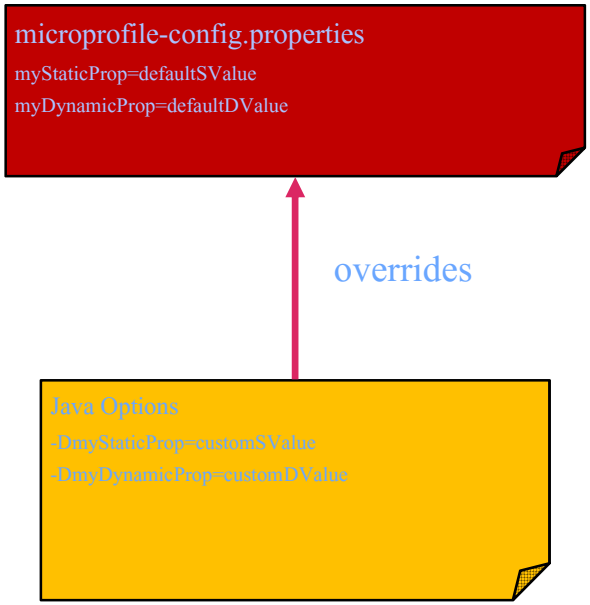
MICROPROFILE
OPTIMIZING ENTERPRISE JAVA

Static Config

```
@Inject
@ConfigProperty(name="myStaticProp")
private String staticProp;
```

Dynamic Config

```
@Inject
@ConfigProperty(name="myDynamicProp")
private Provider<String> dynamicProp;
```



Profile Fault Tolerance

tion to build a resilient microservice

ry - `@Retry`

cuit Breaker - `@CircuitBreaker`

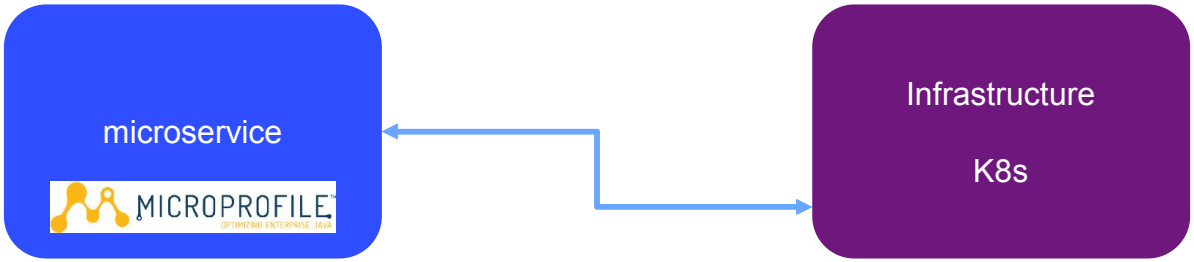
k Head - `@Bulkhead`

ne out - `@Timeout`

lback - `@Fallback`

tor app

e MicroProfile and K8s to build a microservice => 12 factor app

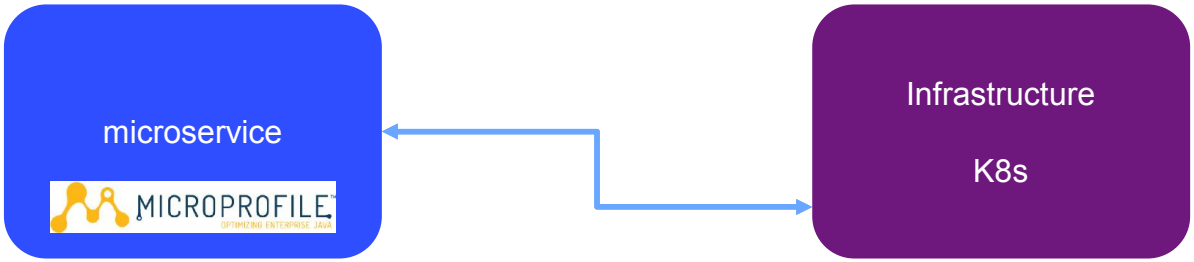


nces

://microprofile.io

://openliberty.io

s://www.12factor.net/



seCon Sessions – MicroProfile and Jakarta EE

r MicroService in Half Hour

Emily Jiang

Wednesday, June 13, 2018 – 9:45 to 10:20

d Native Java Development with MicroProfile

Alasdair Nottingham

Wednesday, June 13, 2018 – 10:45 to 11:20

ot Your Parent's Java EE

Kevin Sutter

Wednesday, June 13, 2018 – 14:40 to 15:15

Ignite - Wednesday

MicroProfile meets Istio (Ignite)

Speaker: Emily Jiang

Date/Time: Wednesday, June 13, 2018 – 17:15 to 18:00

Speed Dating with Jakarta EE (Ignite)

Speaker: Kevin Sutter

Date/Time: Wednesday, June 13, 2018 – 17:15 to 18:00

Thursday

JAX-RS 2.1 and Beyond…

Speaker: Andy McCright

Date/Time: Thursday, June 14, 2018 - 14:15 to 14:50

Resilient Microservices with Eclipse MicroProfile

Speaker: Emily Jiang

Date/Time: Thursday, June 14, 2018 - 15:15 to 15:50

p: Using IBM Cloud Private

	Source: Github Enterprise, github Images: any registry, IBM Cloud private registry
	Dependency management of language environment; container build process for repeatable inclusion of dependencies
	k8s configmaps and secrets
	Use configuration (see previous factor) to define target server as used by application
un	UrbanCode Deploy UrbanCode Release Plus k8s mechanisms with CI tooling
	To be considered in application design

Port binding	Application needs to expose ports. Ingress/service definition of k8s manages mapping of ports
Concurrency	App design ([auto]scaling built into k8s)
Disposability	App design
Dev/prod parity	Can use helm to deploy in same way Namespaces for isolation of similar areas
Logs	ELK as part of ICP (or RYO)
Admin processes	App design; standard k8s tooling lik “kubectl exec” or Kubernetes Jobs