Creating *Twelve Factor Applications* with WebSphere Liberty on Bluemix: A practical guide

Erin Schnabel
schnabel@us.ibm.com
@ebullientworks

InterConnect 2017

Cloud Native Systems

- Container packaged
  - Isolated deployment

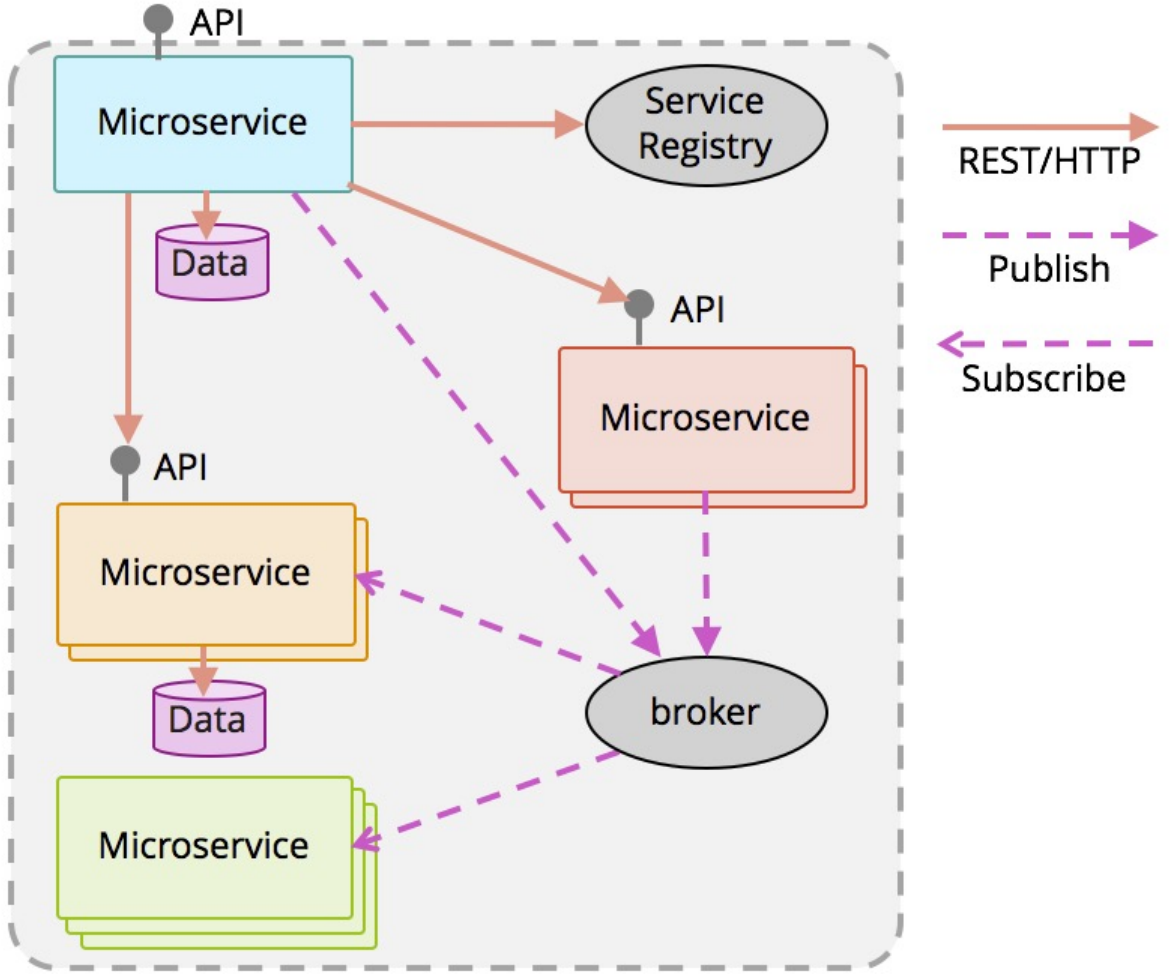
- Dynamically managed
  - Centralized orchestration and automation
  - Active scheduling and management → resource utilization

- Microservice oriented
  - Loosely coupled, explicit dependencies

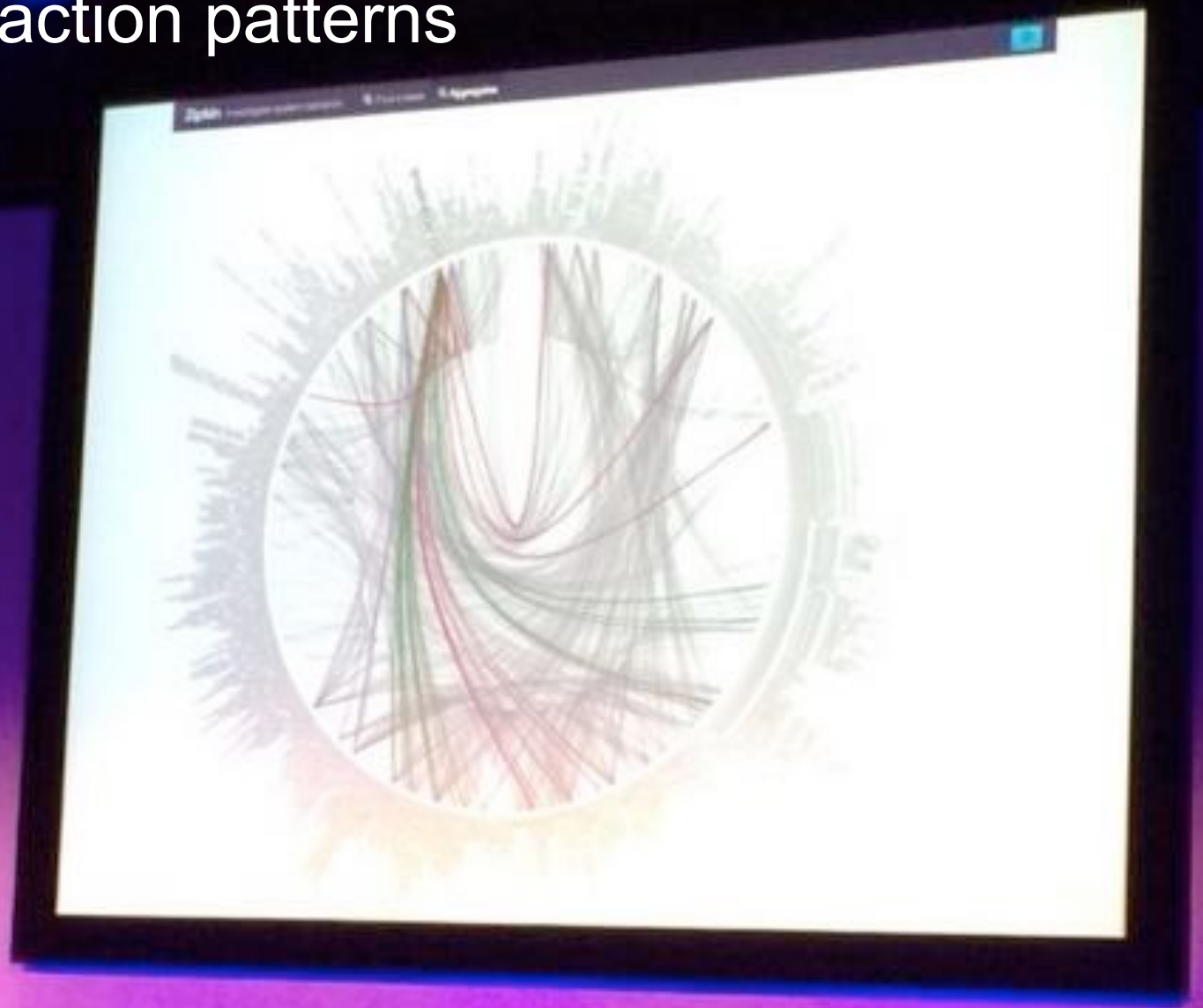
Cloud Native Computing Foundation: https://cncf.io/about/charter

Microservices are used to…

- compose a *complex application* using
  - "small"
  - independent (autonomous)
  - replaceable
  - processes
- that communicate via
  - language-agnostic APIs



Interaction patterns



"Twitter microservices map looks just like the Netflix one. We called this the 'Death Star' diagram"

— Adrian Cockcroft
via twitter

https://twitter.com/adrianco/status/441883572618948608

Fallacies of distributed computing

- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn’t change
- There is one administrator
- Transport cost is zero
- The network is homogenous

-- L Peter Deutsch, 1994

https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing

Twelve Factor Applications

- “a methodology for building software-as-a-service applications”
  - Created by developers at Heroku

- Factors are independent of
  - programming language,
  - backing services,
  - cloud provider

http://12factor.net/

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing Services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

A few core themes

- Independent / Autonomous
  - Portable
  - Scalable
- Resilient / Fault Tolerant
- Observable
- Automated

Twelve Factors + Liberty = ❤️

- Liberty is a flexible, fit-for-purpose runtime

- Immutable artifacts
  - Liberty buildpack in Bluemix
  - Liberty images in Dockerhub
  - Maven/Gradle targets for self-contained zip or runnable jar

- Native support in the runtime for processing environment variables

```
<jndiEntry jndiName="jndi.property" value="${env.ENV_VAR}" />
```

Twelve Factors

Factor 1: Codebase

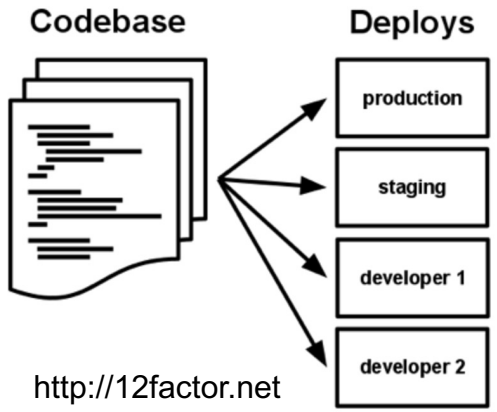
Independent, Automated

- “One codebase tracked in revision control, many deploys”-12factor.net

- Key points
  - One-to-one correlation between codebase and the app
    - ‘If there are multiple codebases, it’s not an app – it’s a distributed system”
  - Codebase is the same across environments: dev, staging, production

- Options
  - Git, Mercurial , Subversion, …

- Does this really mean one repository per service?



http://12factor.net

Factor 2: Dependencies

Independent, Automated

- “Explicitly declare and isolate dependencies” -12factor.net

- Key points:
  - App declares all dependencies, completely and exactly
  - App does not rely on “pre-requisite” system-wide packages
  - Use dependency isolation tool, e.g. gradlew

Explicit dependencies for Java

- Use mavenw or gradlew
  - New tool: bx dev …
- Use explicit versions

```
<!-- Enable features -->
<featureManager>
    <feature>websocket-1.1</feature>
    <feature>localConnector-1.0</feature>
    <feature>jaxrs-2.0</feature>
    <feature>cdi-1.2</feature>
    <feature>jsonp-1.0</feature>
    <feature>ssl-1.0</feature>
    <feature>concurrent-1.0</feature>
    <feature>appSecurity-2.0</feature>
</featureManager>
```

Explicit dependencies for Java

- Explicit feature versions
- Better Docker caching (layers)

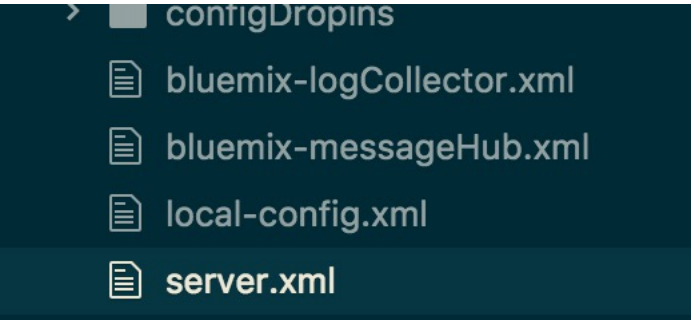
```
FROM websphere-liberty:beta
MAINTAINER Erin Schnabel <schnabel@us.ibm.

# Install required features
RUN /opt/ibm/wlp/bin/installUtility instal
    apiDiscovery-1.0 \
    bluemixLogCollector-1.1 \
    cdi-1.2 \
    concurrent-1.0 \
    couchdb-1.0 \
    localConnector-1.0 \
    jaxrs-2.0 \
    jndi-1.0 \
    jsonp-1.0 \
    ssl-1.0 \
    websocket-1.1
```

Factor 3: Config

- “Store config in the environment”-12factor.net

- Key points:
  - Config includes anything that can vary between deploys
  - Application config should stay with the app

Using Environment variables to toggle configuration




```
<jndiEntry jndiName="targetPlatform" value="${env.TARGET_PLATFORM}"/>
<include location="${env.TARGET_PLATFORM}-config.xml" optional="true" />
<include location="${env.TARGET_PLATFORM}-logCollector.xml" optional="true" />
<include location="${env.TARGET_PLATFORM}-messageHub.xml" optional="true" />
```

```
<server description="Settings for local development (overlays), skip/re

    <logging consoleLogLevel="INFO" />

    <applicationMonitor updateTrigger="polled" pollingRate="30s"/>
    <config updateTrigger="polled" monitorInterval="30s" />

</server>
```

Factor 4: Backing Services

- “Treat backing services as attached resources” -12factor.net

- Key points:
  - Backing services → Datastore, Watson, ….
  - Resources can be attached and detached at will

Injected Credentials

- couchdb in a docker container locally
- Cloudant in Bluemix
- Consistent libraries / code paths across environments

```
<couchdb id="couchdb" jndiName="couchdb/connector"
    libraryRef="couchdb-lib"
    password="${env.COUCHDB_PASSWORD}"
    url="${env.COUCHDB_SERVICE_URL}"
    username="${env.COUCHDB_USER}"/>
```

Factor 5: Build, Release, Run

Independent, Automated

- “Strictly separate build and run stages”-12factor.net

- Key points:
  - Strict separation between the three stages: Build, Release, Run
  - e.g. no code changes at runtime

Liberty maven/gradle tools

- Maven and gradle tools to support continuous integration and test

- Immutable artifacts:
  - Server package (zip)
  - Fat jar
  - Docker image

Factor 6: Processes

Independent, Resilient, Automated

- “Execute the app as one or more stateless processes”-12factor.net

- Key points:
  - 12-Factor processes are stateless and share-nothing
  - Never assume that anything cached in the process will be available on a future request
  - Cache as a backing service…

Factor 7: Port Binding

Automated

- “Export services via port binding”-12factor.net

- Key points:
  - App is completely self-contained
  - “…The web app exports HTTP as a service by binding to a port,…” ***
  - Host and port should be provided by the environment

- Liberty varies
  - Use variable in server.xml to specify port for Cloud Foundry PaaS
  - Use fixed port in a Docker container for IBM Container Service

  - Single application per instance

Factor 8: Concurrency

Independent, Automated

- “Scale out via the process model”-12factor.net

- Key points:
  - Recommends splitting processes based on the type of work
    - Request-driven (HTTP) vs. long running / background tasks
  - Scale by making more processes

- Bluemix scaling: CF, IBM Container or k8s cluster!

Factor 9: Disposability

Resilient

- “Maximize robustness with fast startup and graceful shutdown”-12factor.net

- Key points:
  - The 12-factor app's processes are disposable
  - Strive to minimize startup time
  - Robust against 'sudden death'

- Liberty has fast startup / clean shutdown
- Use the Java EE lifecycle methods to hook in to startup/shutdown events
- Don’t do expensive cache population at startup

Factor 10: Dev/Prod parity

Automated

- “Keep development, staging, and production as similar as possible” -12factor.net

- Key points:
  - Use the same (or very similar) services for dev and production
  - Dev/prod parity especially important for backing services

- Game On!
  - Microservices application running in containers on IBM Container Service
    - Bluemix services (Redis, MessageHub, Cloudant)
  - Docker-compose to coordinate containers locally
    - Including all required backing services! (Redis! Kafka! CouchDB!)

Factor 11: Logs

Observable, Automated

- “Treat logs as event streams” -12factor.net

- Key points:
  - App never concerns itself with routing or storage of its output stream
  - Process streams are captured by the execution environment

- Your application shouldn’t worry over logs.
- Liberty has features to optimize log collection (ELK, bluemix log services)

Factor 12: Admin Process

Automated

- “Run admin/management tasks as one-off processes”-12factor.net

- Key points:
  - Keep admin task code with application code
  - Run admin tasks in an identical environment to the app
    - Run against a “Release” / Same config
  - Same dependency isolation: gradlew, bundle exec, python virtualenv

Notices and disclaimers

Notices and disclaimers continued

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a particular, purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

InterConnect
2017