Cloud Native Application

Ritu Maheshwari
Developer Advocate

IBM **Developer**

IBM

Contents

- Key tenets of Cloud Native Application
- Key tenets of a Microservices Architecture
- Why Microservices and Cloud Native?
- 12 Factor Apps

App Modernization is inevitable

What is Cloud native?

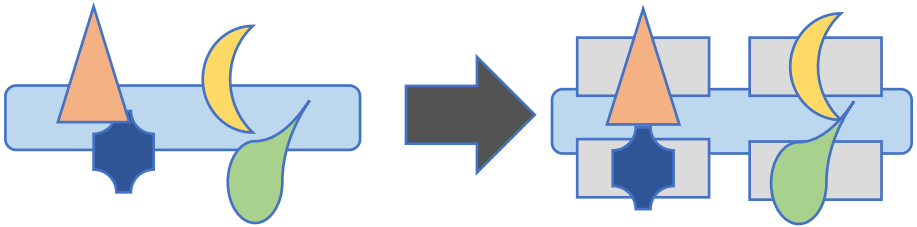
Cloud-native is about how we build and run applications taking full advantage of cloud computing rather than worrying about where we deploy it.

Key tenets of a cloud native application

1. Packaged as light weight **containers**
2. Developed with best-of-breed languages and frameworks
3. Designed as loosely coupled **microservices**
4. Centered around **APIs** for interaction and collaboration
5. Architected with a clean separation of stateless and stateful services
6. Isolated from server and operating system dependencies
7. Deployed on self-service, elastic, **cloud infrastructure**
8. Managed through agile **DevOps** processes
9. Automated capabilities
10. Defined, policy-driven resource allocation

Key tenets of a microservices architecture

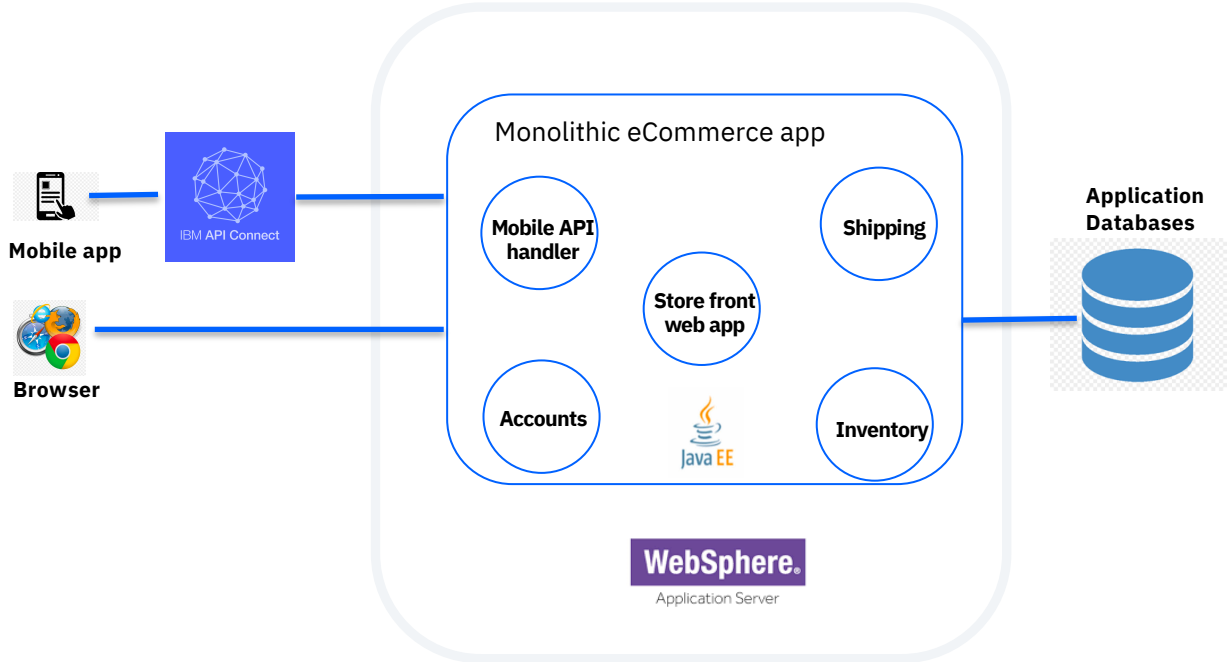
1. Large monoliths are broken down into many small services
2. Services are optimized for a single function or business capability
3. Teams that write the code should also deploy the code
4. Design for failure



Example monolithic application

eCommerce app

- Store front web interface
- Customer Accounts
- Inventory
- Shipping
- Back end for mobile app

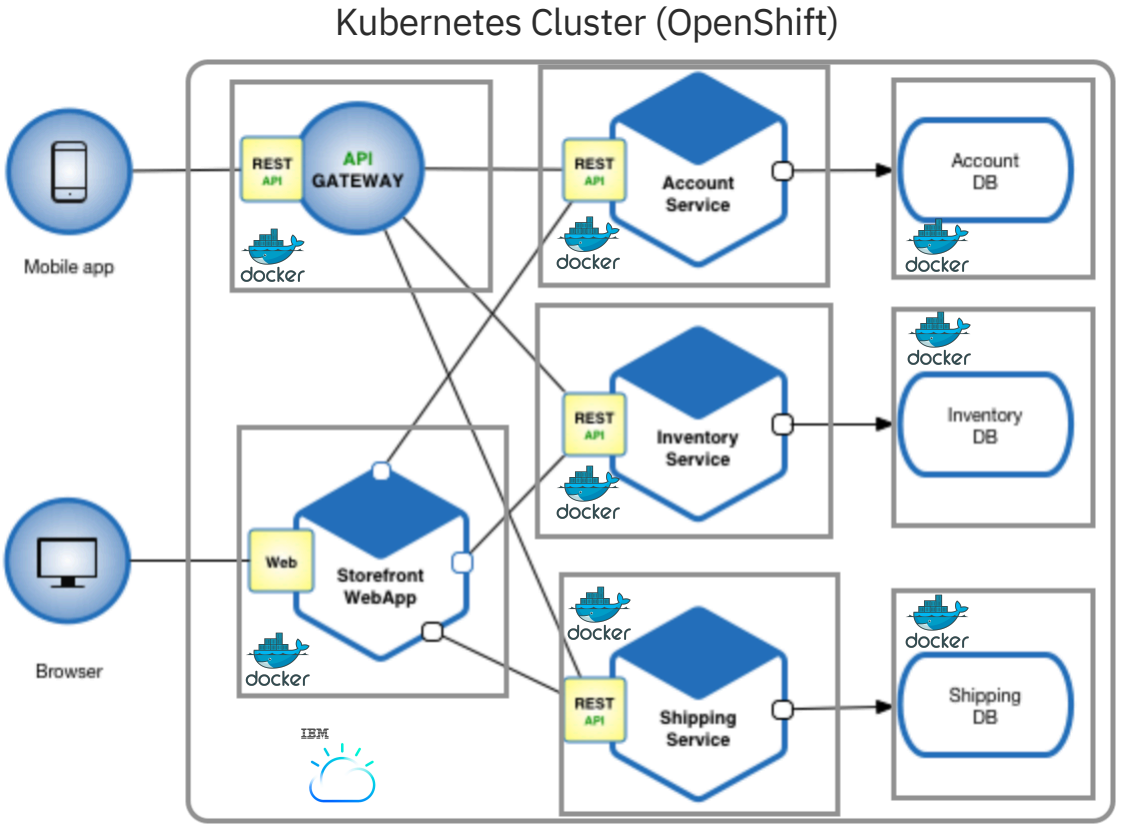


An eCommerce Java EE app on Websphere

Transformed application

Key technologies

- Containers (Docker)
- Container orchestration (Kubernetes)
- Transformation Advisor
- 12-Factor Best Practices
- CI/CD tools (e.g Jenkins)



An eCommerce microservices app on a Kubernetes cluster

Why microservices and cloud native?

Efficient teams	Simplified deployment	Right tools for the job	Improved application quality	Scalability
• End to end team ownership of relatively small codebases	• Each service is individually changed, tested, and deployed without affecting other services	• Teams can use best of breed technologies, libraries, languages for the job at hand	• Services can be tested more thoroughly in isolation	• Services can be scaled independently at different rates as needed
➢Teams can innovate faster and fix bugs more quickly	➢Time to market is accelerated.	➢Leads to faster innovation	➢Better code coverage	➢Leads to better overall performance at lower cost

12 Factor Apps

12 Factor is a methodology for building software

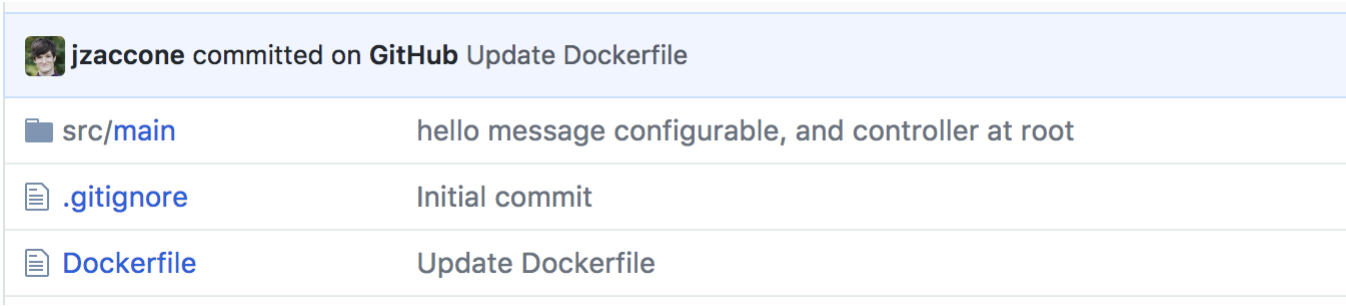
Tenets for a 12 Factor App

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release, Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin processes

I. Codebase

Code for a single application should be in a single code base

- Track running applications back to a single commit
- Use Dockerfile Maven, Gradle, or npm to manage external dependencies
- Version pinning! Don’t use latest
- No changing code in production



II. Dependencies

Explicitly declare and isolate dependencies. AKA: Remove system dependencies

How?

- Step 1: Explicitly declare dependencies (Dockerfile)
- Step 2: Isolate dependencies to prevent system dependencies from leaking in (containers)

```
1  FROM openjdk:8-jdk-alpine
2  EXPOSE 8080
3  WORKDIR /data
4  CMD java -jar *.jar
5  COPY target/*.jar /data/
```

III. Config

Store config in the environment (not in the code).

How?

- Inject config as environment variables (language agnostic)
- ConfigMap in Kubernetes does this ^

IV. Backing Services

Treat backing resources as attached services. Swap out resources.

How?

- Pass in URLs via config (see III.)
- K8s built in DNS allows for easy service discovery

```
services:

  account-api:
    build:
      context: ./compute-interest-api
    environment:
      DATABASE_URL: http//account-database

  account-database:
    image: jzaccone/account-database
```

V. Build, Release, Run

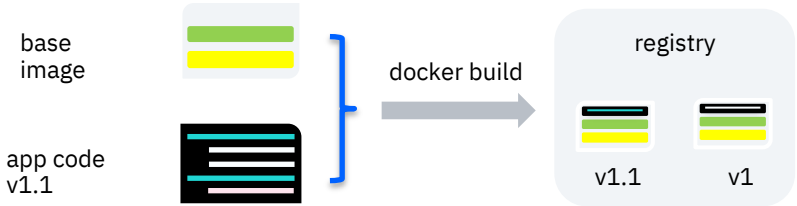
Strictly separate build and run stages.

Why?

Rollbacks, elastic scaling without a new build

How?

- Use Docker images as your handoff between build and run
- Tag images with version. Trace back to single commit (see I. Codebase)
- Single command rollbacks in Kubernetes



VI. Process

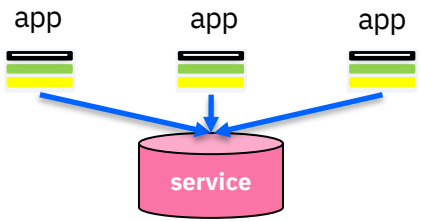
Execute app as stateless process

Why?

Stateless enables horizontal scaling

How?

- Remove sticky sessions
- Need state? Store in volume or external data service
- Use persistent volumes in Kubernetes for network wide storage



VII. Port Binding

Export services via port binding. Apps should be self-contained.

Why?

Avoid “Works on my machine”

How?

- Web server dependency should be included inside the Docker Image
- To expose ports from containers use the —publish flag

VIII. Concurrency

Scale out via the process model. Processes are **first-class citizens**

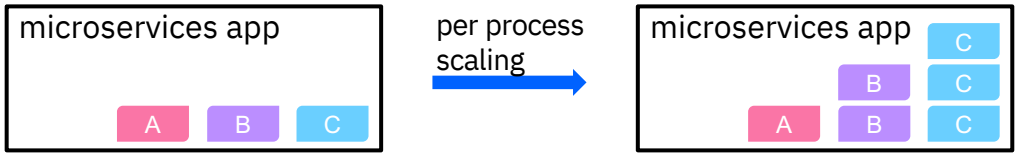
Why?

Follow the Unix model for scaling, which is simple and reliable

How?

- Scale by creating more processes
- **Docker**: really just a process running in isolation
- **Kubernetes**: Acts as process manager: scales by creating more pods

  Don't put process managers in your containers



IX. Disposability

Maximize robustness with fast startup and graceful shutdown

Why?

- Enables fast elastic scaling, robust production deployments. Recover quickly from failures.

How?

- multi-minute app startups!
- Docker enables fast startup: Union file system and image layers
- In best practice: Handle SIGTERM in main container process.

X. Dev/Prod Parity

Keep development, staging and production as similar as possible. Minimize time gap, personnel gap and tools gap

How?

- **Time gap:** Docker supports delivering code to production faster by enabling automation and reducing bugs caused by environmental drift.
- **Personnel gap:** Dockerfile is the point of collaboration between devs and ops
- **Tools gap:** Docker makes it very easy to spin up production resources locally by using `docker run ...`

XI. Logs

Treat logs as event streams

How?

- Write logs to stdout (Docker does by default)
- Centralizes logs using ELK or [your tool stack here]

  Don’t

  Don’t retroactively inspect logs! Use ELK to get search, alerts

  Don’t throw out logs! Save data and make data driven decisions

XII. Admin Processes

Run admin/management tasks as one-off processes.

Don’t treat them as special processes

How?

- Follow 12-factor for your admin processes (as much as applicable)
- Option to collocate in same source code repo if tightly coupled to another app
- “Enter” namespaces to run one-off commands via `docker exec ...`