The twelve-factor app

... and beyond

by **vinhnt44** & **hieulq2**

About us

Vinh Nguyen Trong - @tovin07

- Organizer of Vietnam OpenStack UG
- Core reviewer of OpenStack OSProfiler
- Viettel Network - Technology researcher
- Fujitsu Vietnam Senior Software Engineer
- Email: vinhnt44@viettel.com.vn

Hieu LE - @hieulq

- Organizer of Vietnam OpenStack UG
- Organizer of Vietnam OPNFV User Group
- Member of VFOSSA Executive
- Viettel Network - Technology researcher
- Email: hieulq2@viettel.com.vn

Slide | Q&A session

Link to presentation

https://goo.gl/kGaobn

Q&A

goo.gl/slides/nkuttn

Agenda

... from 10k foot view

Cloud-native apps

12-factor app

Beyond 12, we have 15 for now

Nobody puts Java in a container

Java 10

Cloud-native application

What is cloud native?

Cloud native computing uses an open source software stack
https://www.cncf.io/about/faq/

Containerized

Orchestration

Microservices

Containerized

reproducibility
transparency
resource isolation

Orchestration

scheduled & managed
to optimize resource utilization

Microservices

stateless
increase agility
and maintainability

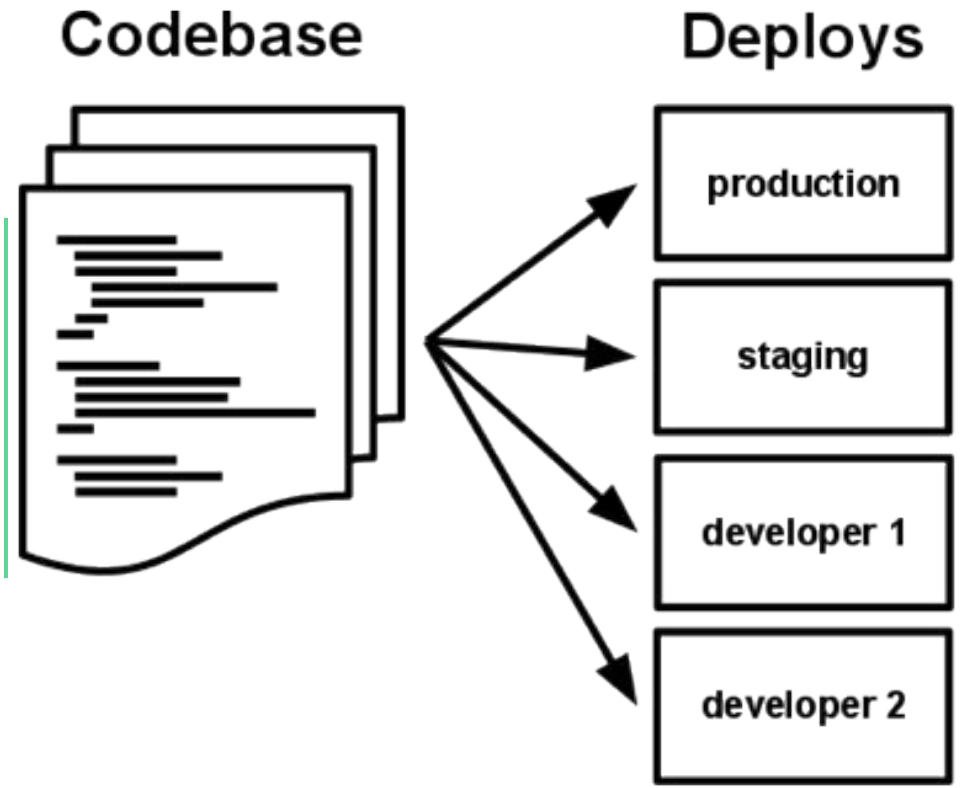
The twelve-factor app

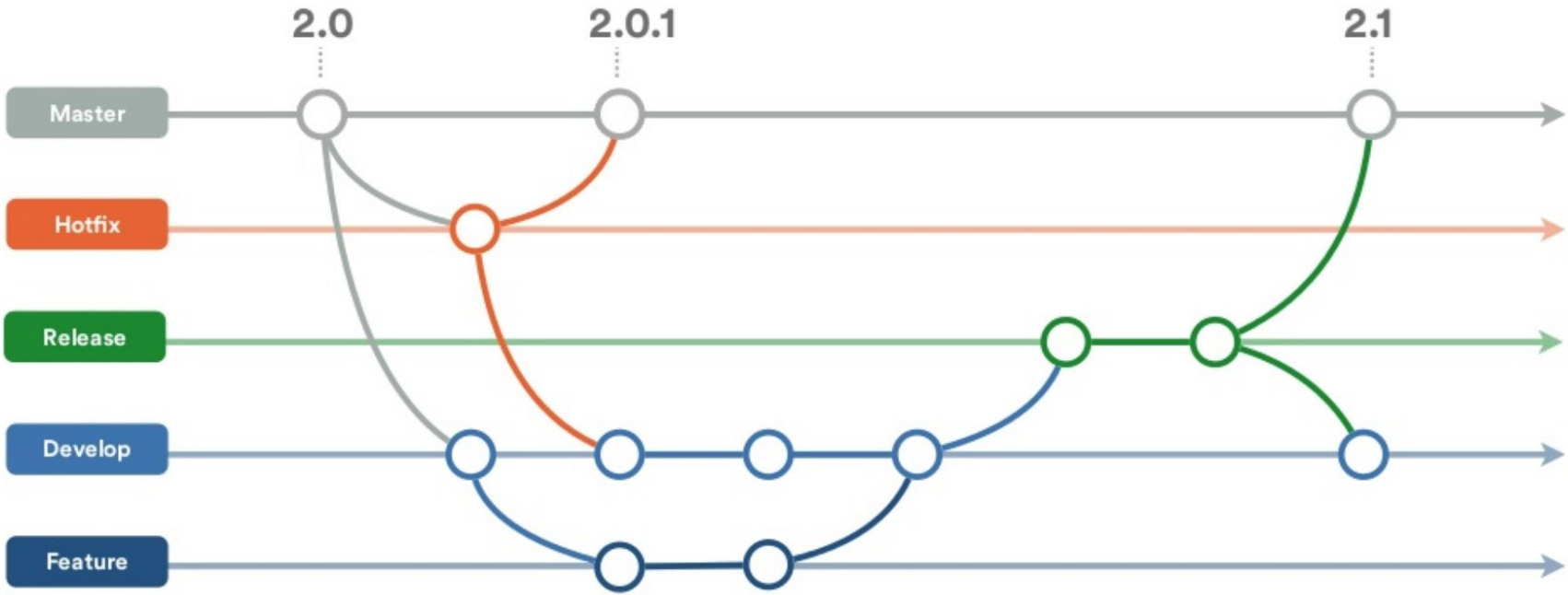
12-factor app

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

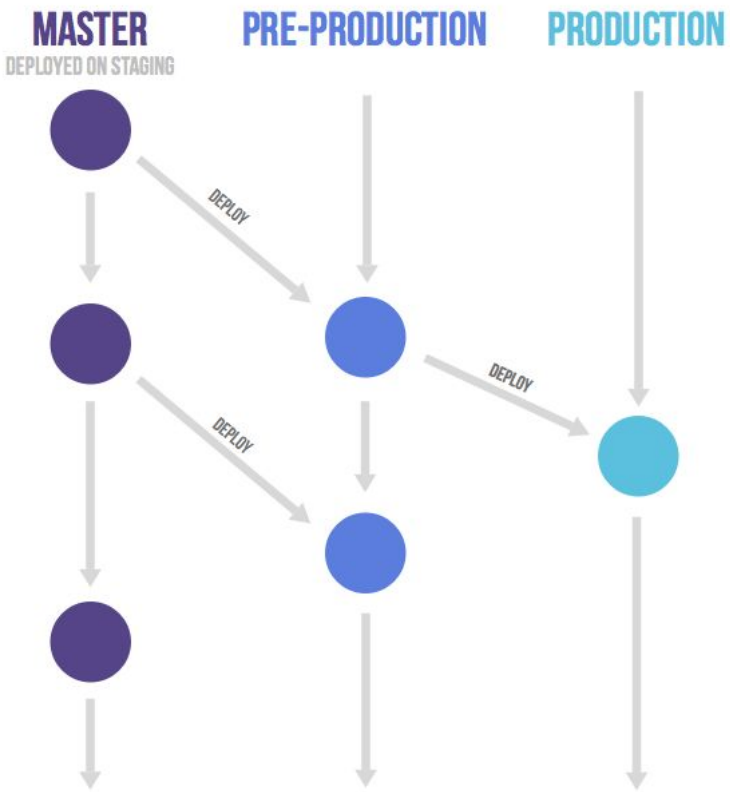
1
Codebase

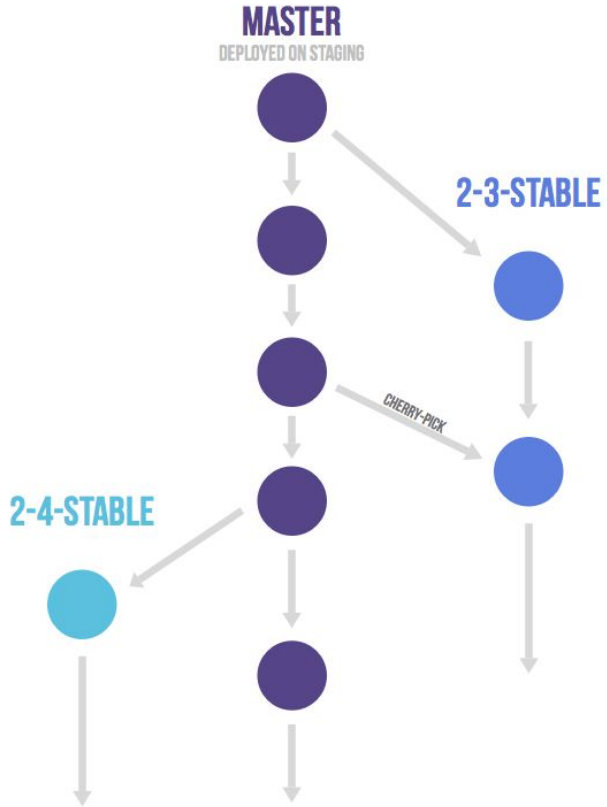
One codebase tracked in revision control, many deploys





Gitflow

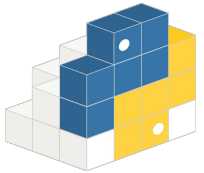




Environment & release branch

2
Dependencies

Explicitly declare
and **isolate dependencies**



3
Config

Store config in the **environment**

Hard-coded

Config files

Environment variables

Config includes

Docker secret
Hashicorp Vault

- Resource handles to the database, Memcached, and other backing services
- Credentials to external services
- Per-deploy values such as the canonical hostname for the deploy

Current style

In java source code (a lot):

```
service = new VsaadminServiceService(url,
     new QName("http://service.vsaadmin.viettel.com/", "VsaadminServiceService")
);
```

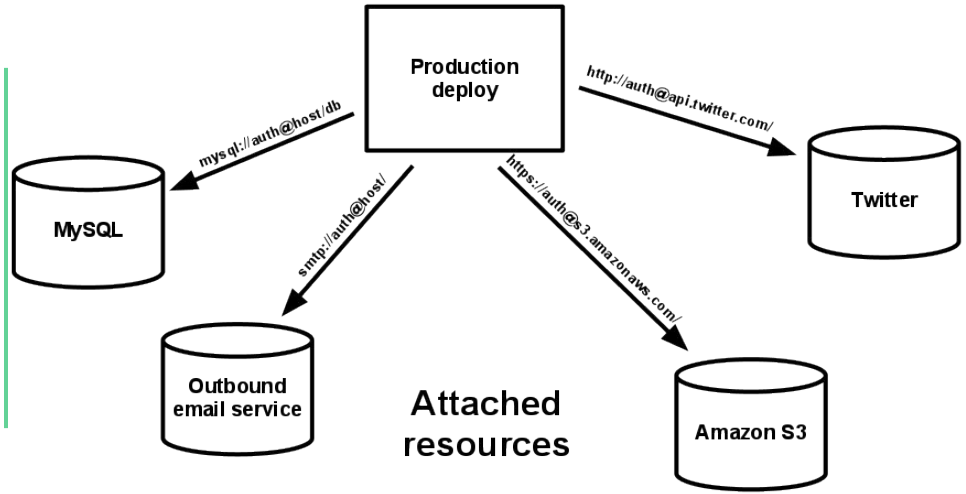
In xml file (a lot):

```
<property name="hibernate.connection.url">jdbc:oracle:thin:@11.22.33.44:1234:PM1</property>
<property name="hibernate.connection.username">qlctkt</property>
<property name="hibernate.connection.password">qlctkt@superpassw0rd</property>
```

Username and password were check-in VCS many times

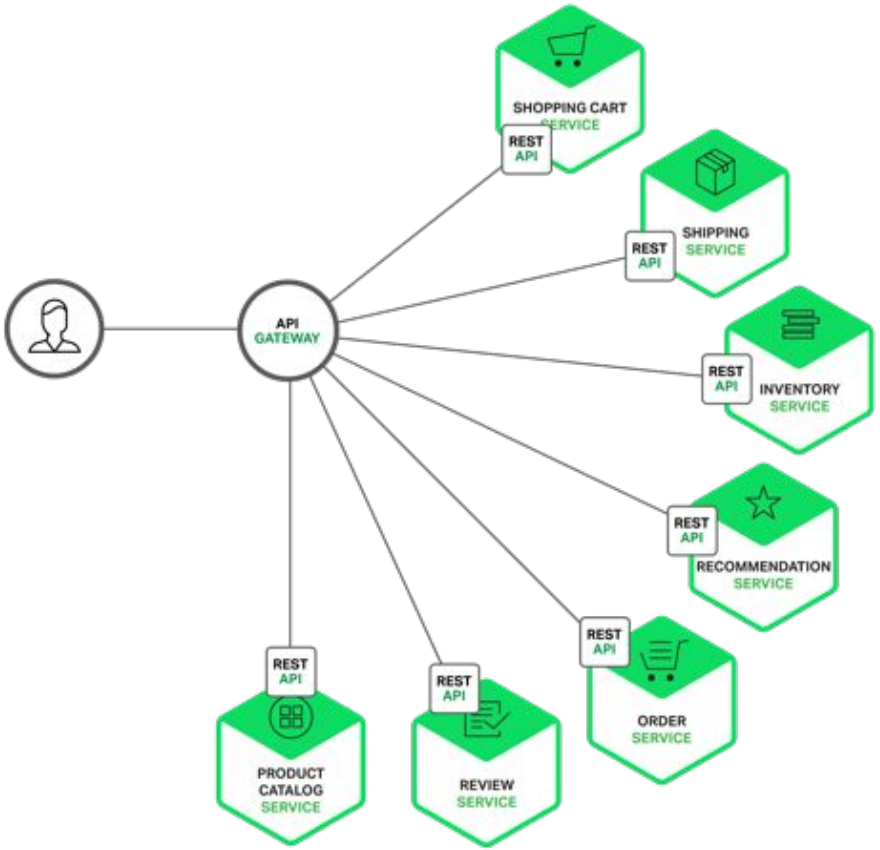
4
Backing services

Treat backing services
as **attached resources**

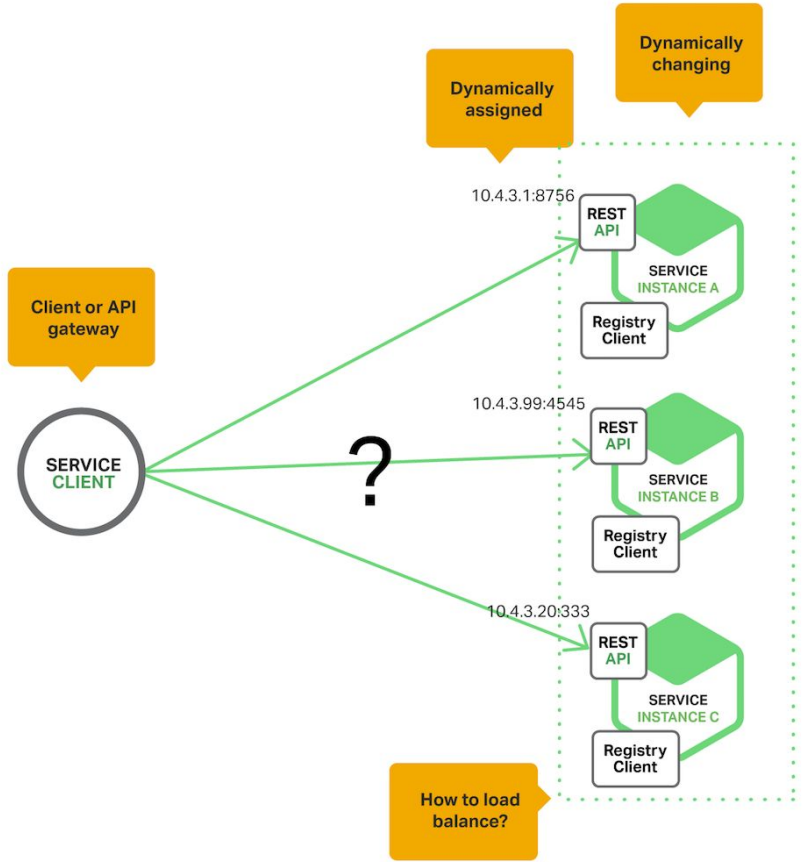


API Gateway

Netflix Zuul
Nginx Reverse Proxy
Linkerd
Spring Cloud Gateway

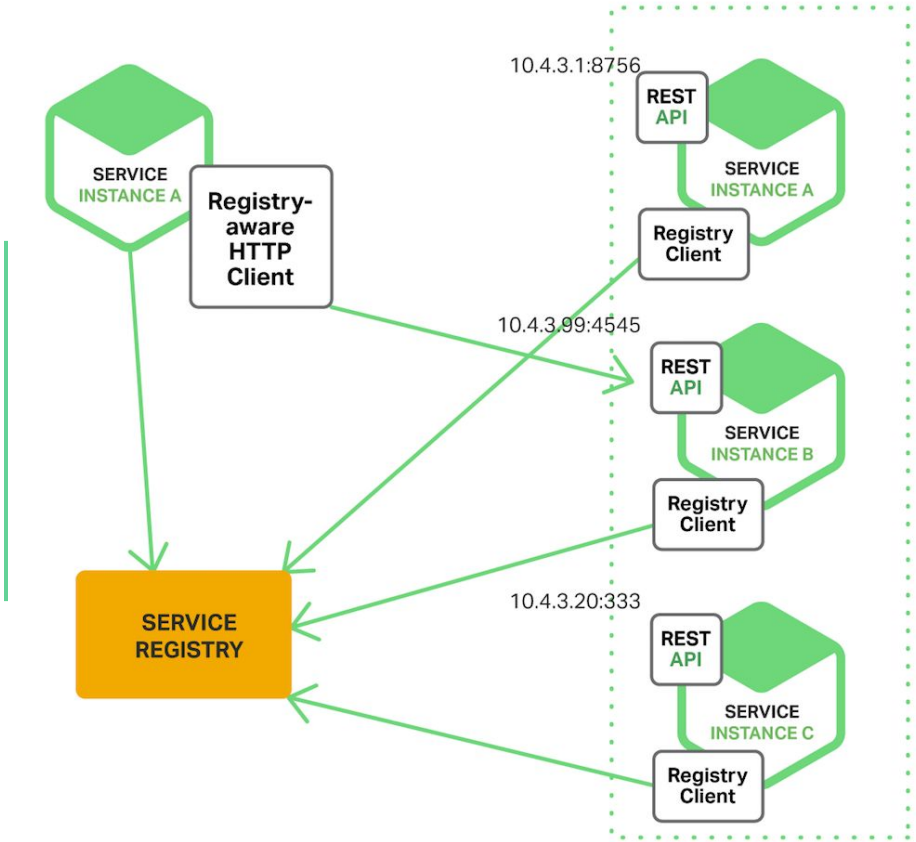


Service discovery



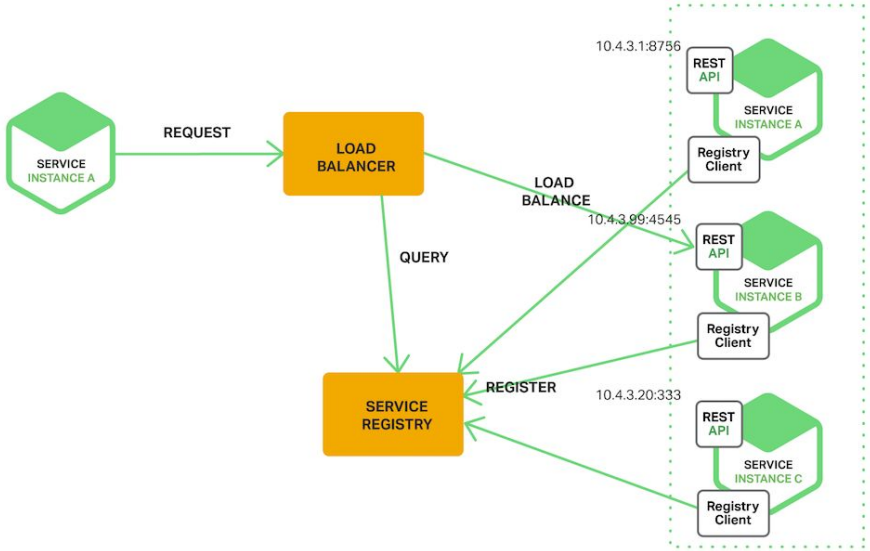
Client-side

Netflix Ribbon

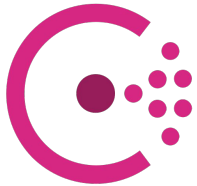


Server-side

Netflix Eureka



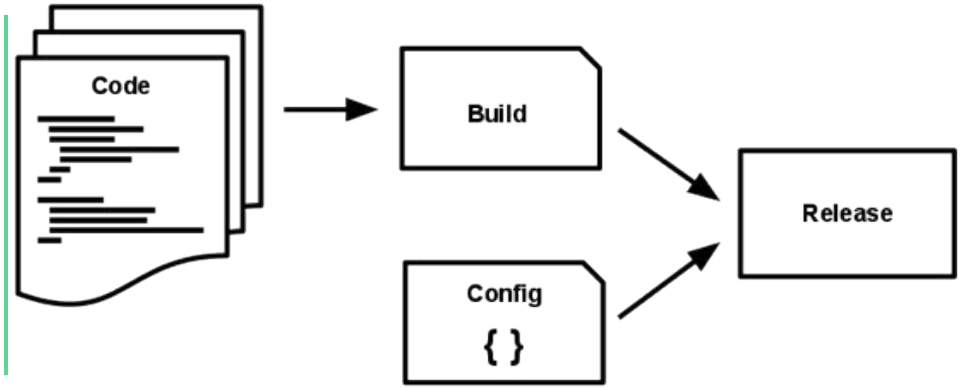
Apache ZooKeeper™



etcd

5
Build, release, run

Strictly separate build and run stages



6
Processes

Execute the app as one or more **stateless** processes

- Stateless & shared-nothing
- Persistent data is stored in stateful backing services
- ~~Sticky session~~ ➜ cached (memcached or redis)

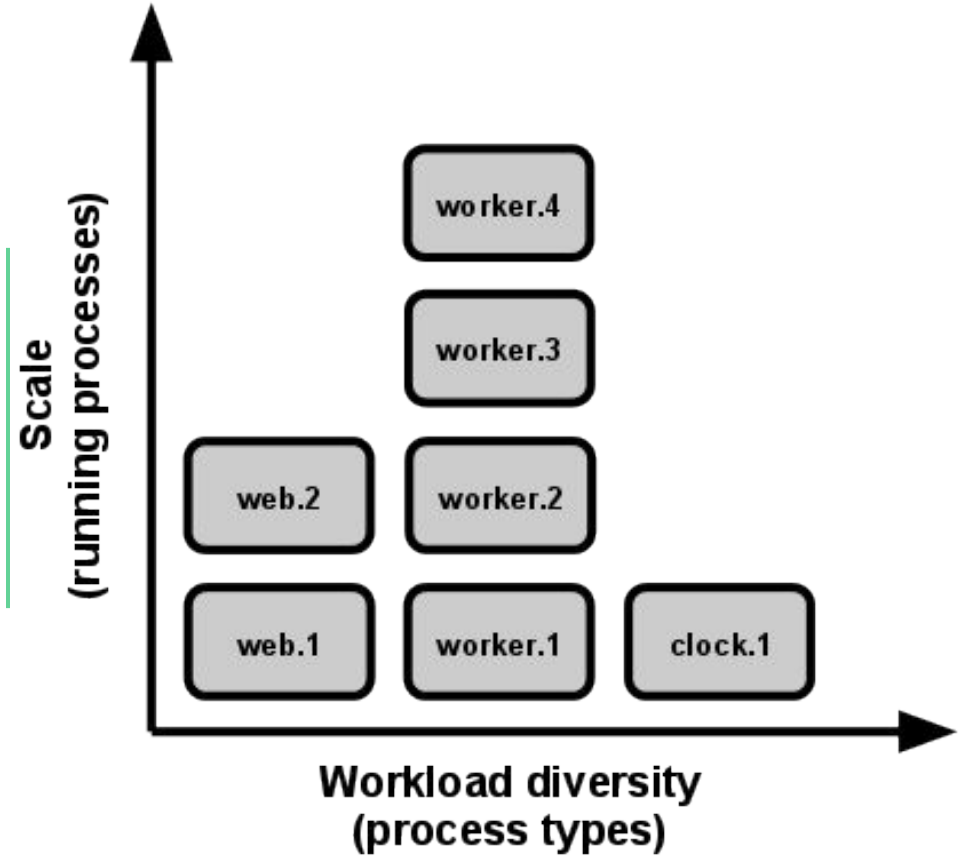
7
Port binding

Export services via **port binding**

- Self-contained
- Expose URI
- One app can become the backing service for another app

8
Concurrency

Scale out via the process model



Inter-process communication

- Async with message-based:
  AMQP, Kafka, ZeroMQ, ...
- Sync:
  REST, Thift, ProtoBuf, Avro

9
Disposability

Maximize robustness
with **fast startup**
and **graceful shutdown**

- Mutable configuration
- Auto restart
- Live reload for web apps
- Crash-only design ☺

Mutable configuration

- Change configuration without restart with Apache Common Configuration

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-configuration2</artifactId>
    <version>2.0</version>
</dependency>
```

Auto restart & live reload

- Auto restart services with spring boot devtools

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional>
</dependency>
```

- Live reload for web app with browser plugin

10
Dev/prod parity

Keep development, staging, and production
as similar as possible

- Small changes
  (large ➜ focused small part)
- Reduce time gaps
  (weeks ➜ hours)
- CI/CD shining here
- Tools: docker, vagrant, ansible, chef, puppet, saltstack

Comparison of traditional vs. 12-factor app

Criteria	Traditional	12-factor
Time between deploys	Weeks	Hours
Code authors vs. code deployers	Different people	Same people
Development vs. production environments	Divergent	As similar as possible

11
Logs

Treat logs as event streams

- ~~*Log file*~~ → Stream
- Log routers: fluentd, logstash
- Log destination:
  stdout, *log file*, log storages,
  elasticsearch
- Library: Log4j

12
Admin processes

Run admin/management tasks as one-off processes

- Types: db migration, console, commit script
- Migration: Flyway, Liquibase
- Run in identical env, same on any release
- Ship with app code to avoid synchronization issues

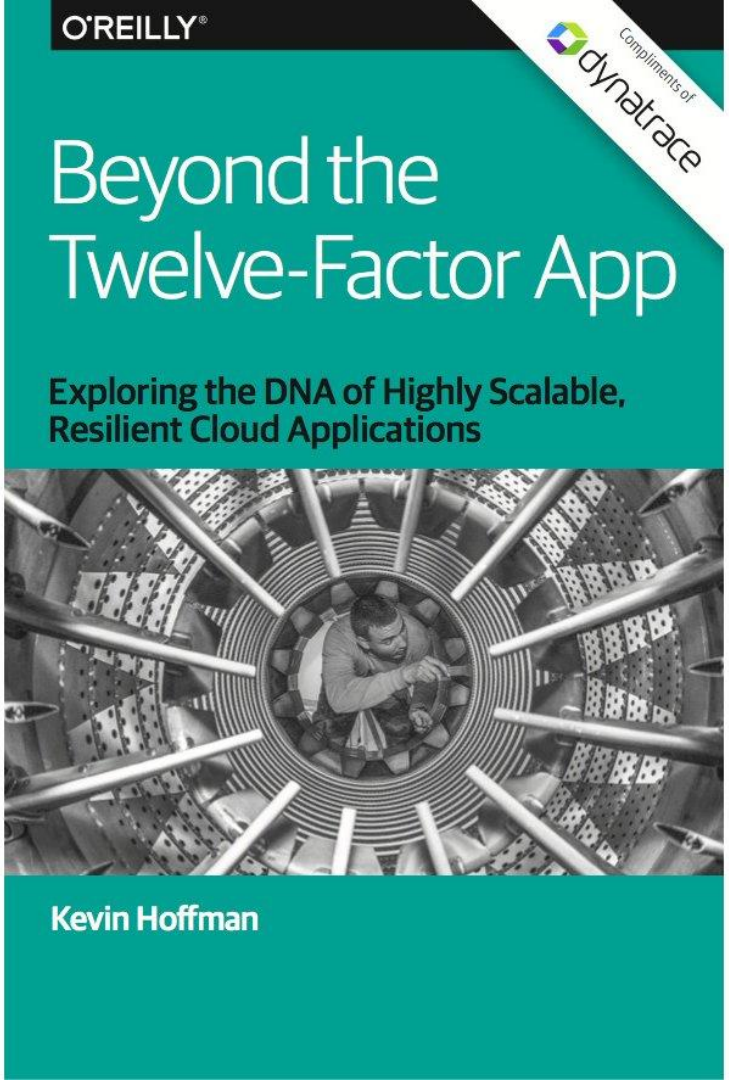
So, now what?

It’s 15-factor app for now

Because 12 is not enough

New factors

API first
Telemetry
AuthN & AuthZ



13
API First

http://www.api-first.com/

- API as a first-class artifact
- API first → easy collaboration
- Mobile first

14
Telemetry

APM, tracing, events, metrics, monitoring, logging

- Application performance monitoring (APM)
- Domain-specific telemetry
- Health and system logs

15
AuthN & AuthZ

cloud-native application
has to be a secure application

- Role-based access control (RBAC)
- OAuth2, OpenID, SSO
- SSO in Viettel: Apereo CAS

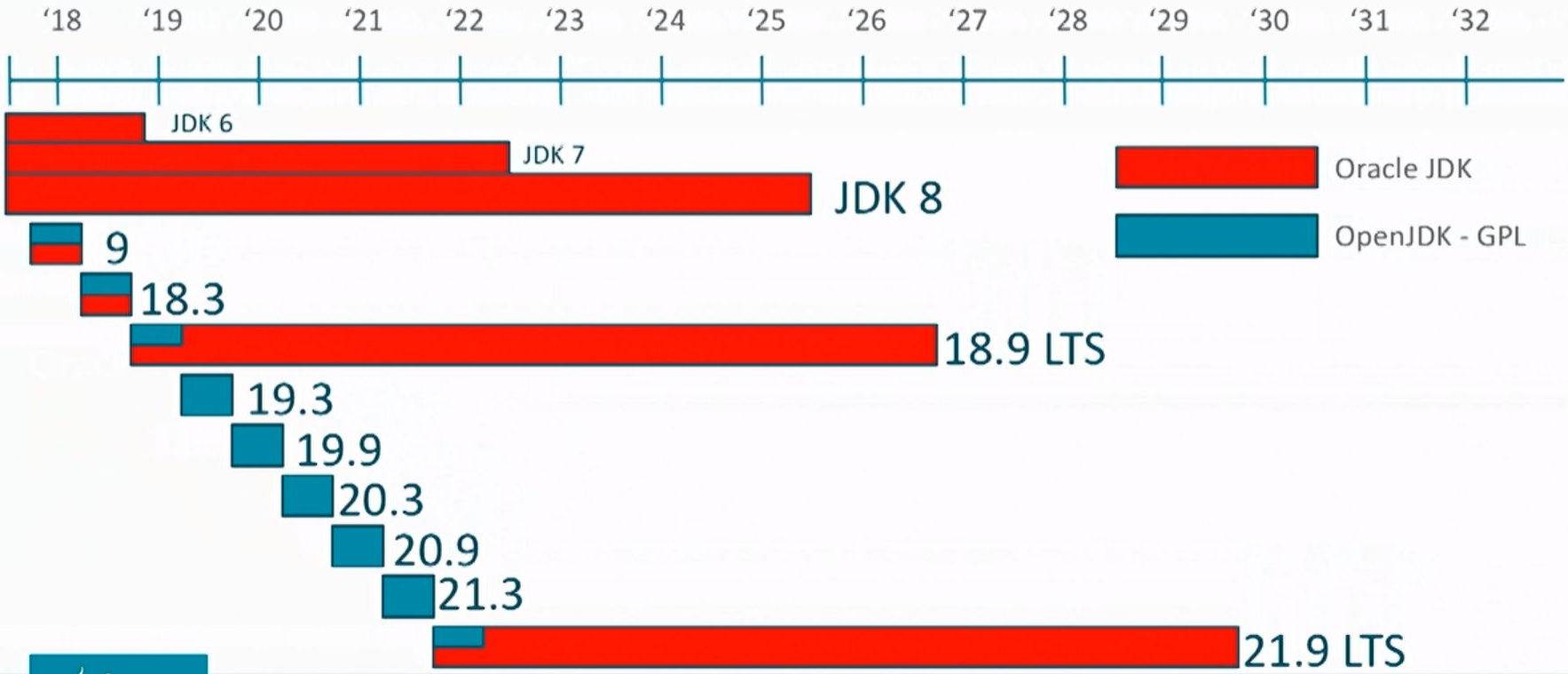
Nobody puts Java in a container

TL;DR

- Java and Docker aren’t friends out of the box
- Docker can set memory/CPU limitations that Java can’t automatically detect
- Old Java: workaround with `Xmx` flags and some docker tricks
- Java 10:
  - JDK-8146115 Improve docker container detection and resource configuration usage
  - JDK-8186248 Allow more flexibility in selecting Heap % of available RAM

Java 10, ftw

Oracle JDK & OpenJDK



Java 10 ❤ Container

JDK-8146115 Improve docker container detection/resource configuration usage

- `-XX:-UseContainerSupport`

JDK-8186248 Allow more flexibility in selecting Heap % of available RAM

- `-XX:InitialRAMPercentage`
- `-XX:MaxRAMPercentage`
- `-XX:MinRAMPercentage`

Java LTS

Java has new naming convention for version from Mar 2018

- Java 10, actually named 18.3
- Java 11 will be released this Sept 2018 with version 18.9 - a LTS version

References

Twelve-factor application

1. The twelve-factor app
2. Running process
3. The New Heroku (Part 4 of 4): Erosion-resistance & Explicit Contracts
4. Pattern of Enterprise application architecture
5. Refactoring - Martin fowler
6. Beyond twelve-factor app
7. Shared-nothing architecture
8. Immortal - *nix cross platform supervisor
9. MariaDB Galera cluster
10. Should the Twelve-Factor App now be Fifteen-Factor?
11. Java hipster tech stack
12. The 12-Factor App: A Java Developer's Perspective
13. NGINX - Building microservices series (7 posts)

References

Cloud-native

1. CNCF Blog - Developing Cloud Native Applications (DevNetCreate.io)
2. Cisco - Developing Cloud Native Applications
3. Cisco - The Journey to Cloud Native

Java

1. Nobody puts Java in a container
2. Java inside Docker
3. Java and Docker memory limit
4. Docker Support in Java 10
5. DZone - Improved Docker container integration with Java 10
6. Docker Blog - Improved Docker container integration with Java 10
7. Better containerized JVMs in JDK 10
8. Java inside docker: What you must know to not FAIL
9. YCombinator - Improved Docker Container Integration with Java 10

Thank you


Uppsala universitet 1852–1916
Studenten, Staden och Samingen
UPPSALA UNIVERSITE

Cloud Native Landscape

v20180525

See the interactive landscape at l.cncf.io

Greyed logos are not open source

App Definition and Development

Database and Data Warehouse

Vitess (CNCF Incubating)

Streaming

NATS (CNCF Incubating), cloudevents (CNCF Sandbox)

Source Code Management

Bitbucket, GitHub, GitLab

Application Definition and Image Build

Kubernetes (CNCF Sandbox)

Continuous Integration / Continuous Delivery (CI/CD)

Orchestration & Management

Scheduling & Orchestration

kubernetes (CNCF Graduated)

Coordination & Service Discovery

CoreDNS (CNCF Incubating)

Service Management

gRPC (CNCF Incubating), envoy (CNCF Incubating), LINKERD (CNCF Incubating), Open Policy Agent (CNCF Sandbox)

Runtime

Cloud-Native Storage

ROOK (CNCF Sandbox)

Container Runtime

containerd (CNCF Incubating), rkt (CNCF Incubating)

Cloud-Native Network

CNI (CNCF Incubating)

Provisioning

Host Management / Tooling

Infrastructure Automation

Container Registries

Secure Images

Notary (CNCF Incubating), TUF (CNCF Incubating)

Key Management

spiffe (CNCF Sandbox), SPIRE (CNCF Sandbox)

Cloud

Public

Private

Platforms

Certified Kubernetes - Distribution

Certified Kubernetes - Hosted

Certified Kubernetes - Installer

Non-Certified Kubernetes

PaaS/Container Service

Observability & Analysis

Monitoring

Prometheus (CNCF Incubating)

Logging

fluentd (CNCF Incubating)

Tracing

JAEGER (CNCF Incubating), OPENTRACING (CNCF Incubating)

Serverless

Kubernetes Certified Service Provider

Kubernetes Training Partner

Special

This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.

l.cncf.io

CLOUD NATIVE Landscape

CLOUD NATIVE COMPUTING FOUNDATION

Redpoint Amplify