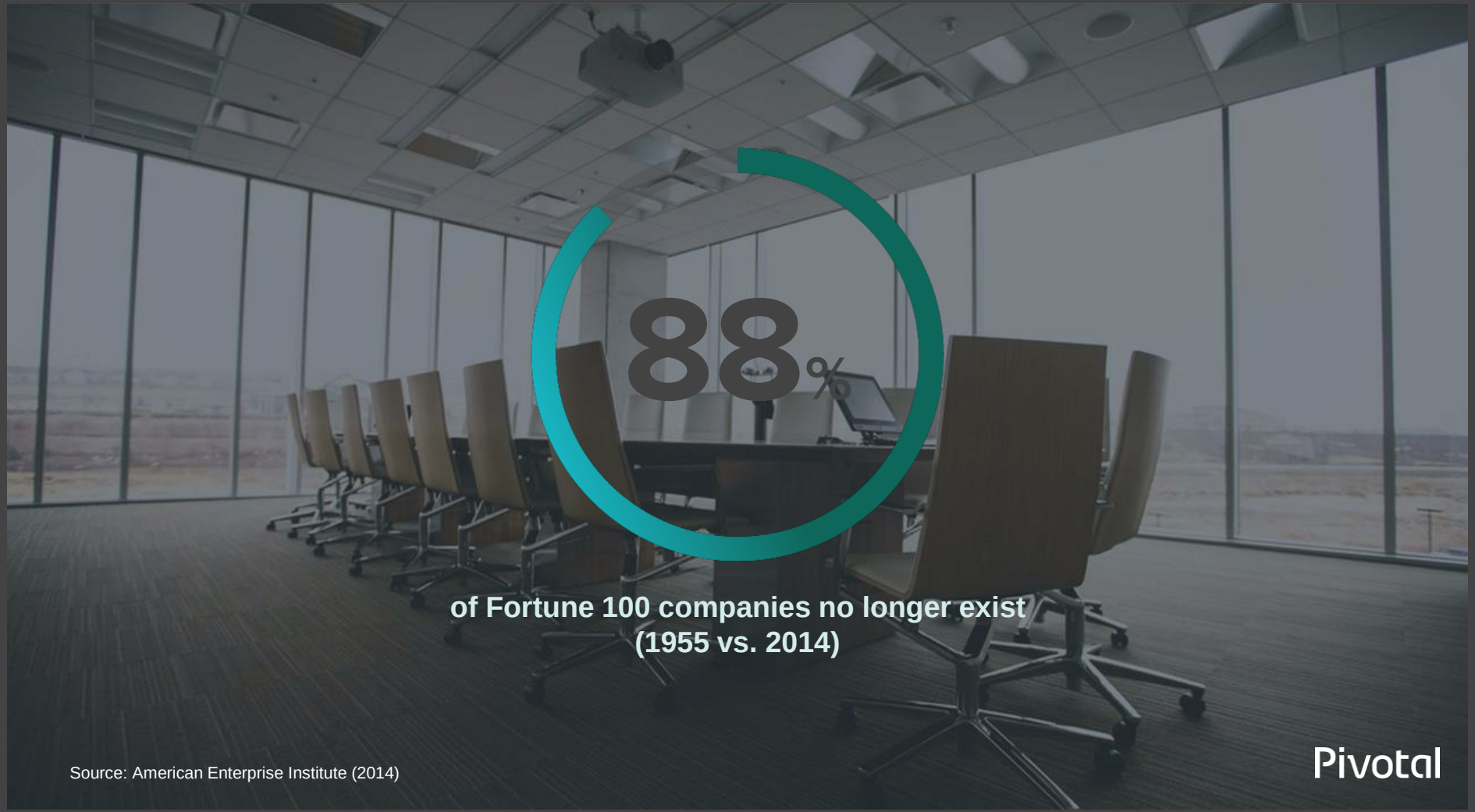
Pivotal

Cloud Native Applications - Going Cloud Native

Derrick Chua
Senior Platform Architect
tchua@pivotal.io
April 2019

You need to be good at software


Disruption is Faceless
“We didn’t do anything wrong but somehow, we lost.”
CEO, Nokia
Pivotal



of Fortune 100 companies no longer exist (1955 vs. 2014)

Source: American Enterprise Institute (2014)

Pivotal

Former Titans of Industry

Software companies have disrupted once untouchable industry leaders

$5BN
Entertainment

$30BN
Technology

$2.2BN
Retail

$1BN
Entertainment

$30BN
Technology

$2.6BN
Retail

Source: Internet

Why do you need to be good at software?

Customers expect it.

Meet the demands to operate at scale.

Gives you more business options.

Your competitors are improving.

It makes your life better.

Ok, but how do I know that I'm getting better at software?

Improved savings. Lower cost per unit of deployed compute.

Improved security. Achieving 100% patch coverage.

Improved speed. Faster cycle time, more frequent deployments.

Improved scale. More requests per second to apps and services.

Improved stability. Greater uptime of customer-facing service.

What keeps you from being good at software?

It's hard to experiment and quickly incorporate what we learn.

Stuck with incomplete or outdated application platforms.

Hostile processes and procedures make it painful to ship software.

Organization silos have competing priorities.

Going Cloud Native

What is Cloud Native?

CNCF's definition

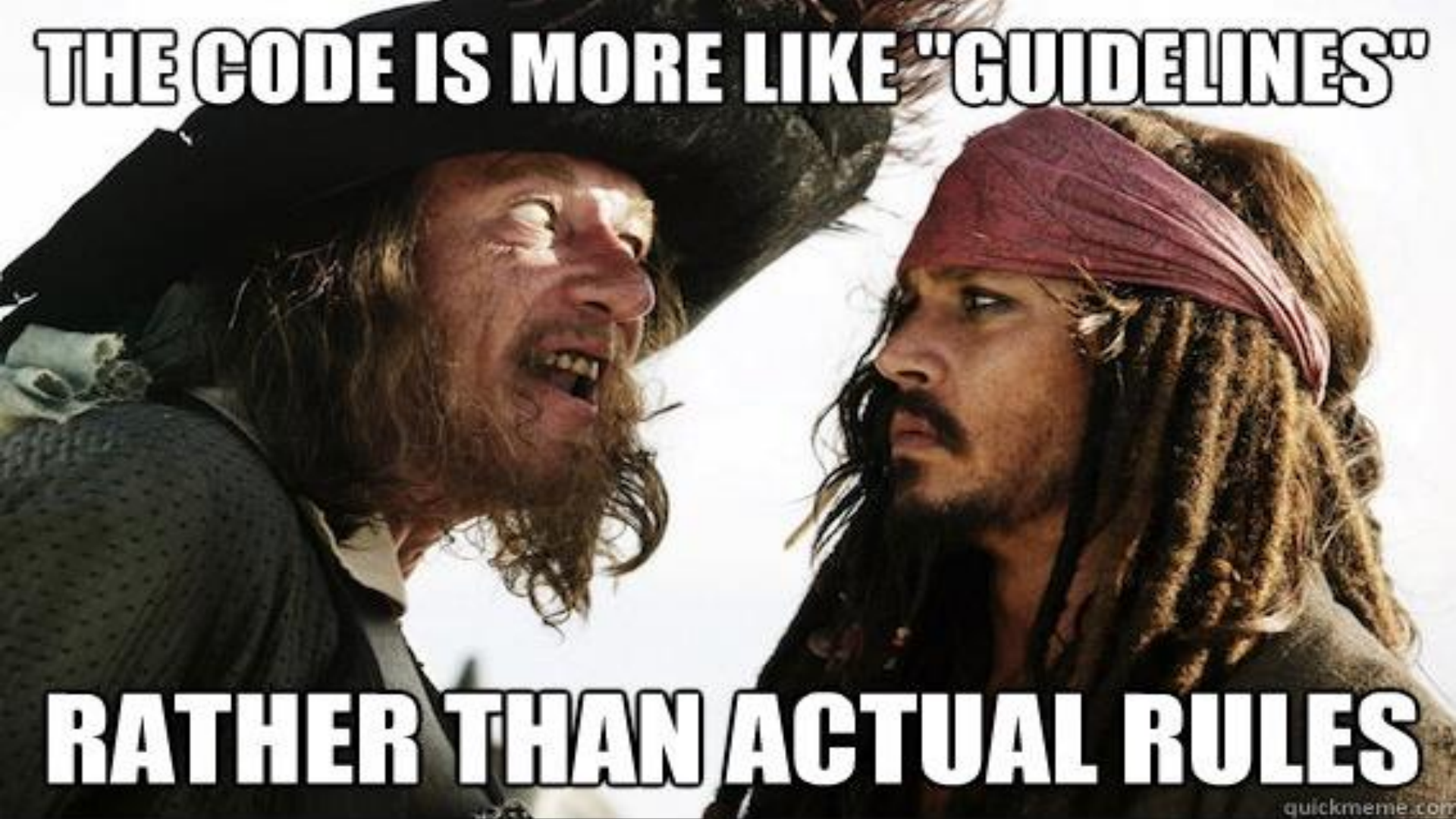
Cloud native technologies empower organizations to build and run **scalable applications** in modern, dynamic environments such as **public, private, and hybrid clouds**. **Containers, service meshes, microservices, immutable infrastructure, and declarative APIs** exemplify this approach.

These techniques enable **loosely coupled systems** that are **resilient, manageable, and observable**. Combined with **robust automation**, they allow engineers to **make high-impact changes frequently** and **predictably with minimal toil.**

Another definition

1. Packaged as lightweight containers
2. Developed with the 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

https://thenewstack.io/10-key-attributes-of-cloud-native-applications/


THE CODE IS MORE LIKE "GUIDELINES"
RATHER THAN ACTUAL RULES
quickmeme.com

Going Cloud Native

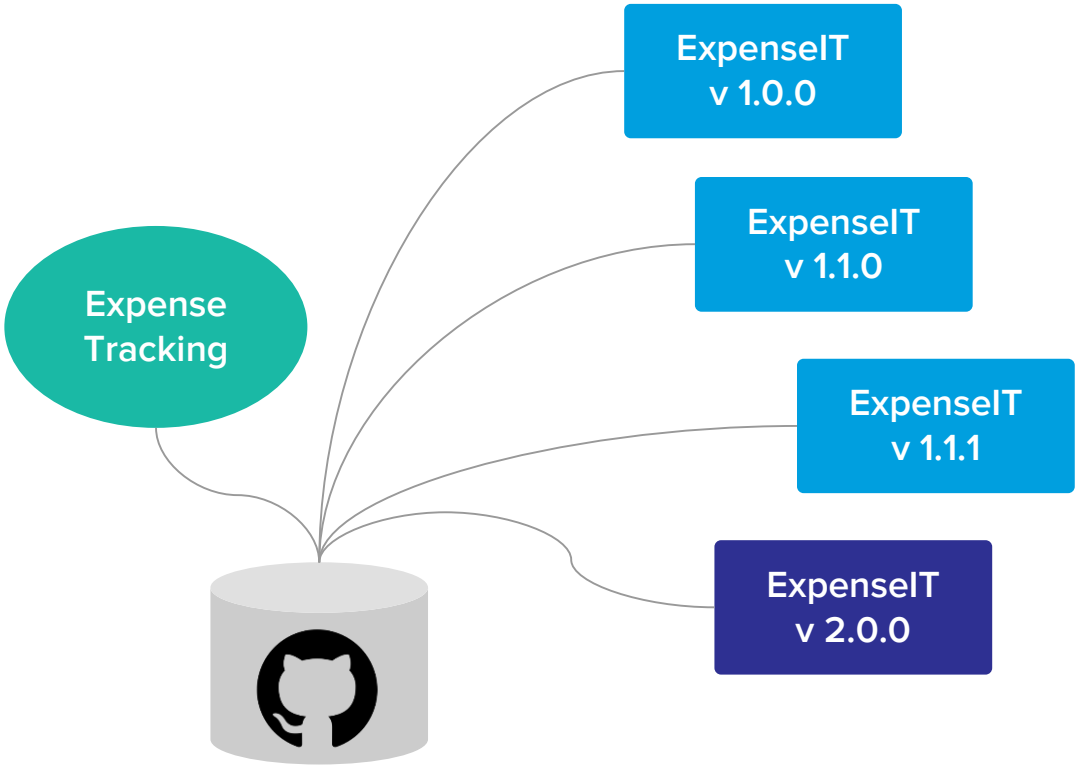
12 Factor App

12 Factors – Methodology for Cloud Native Implementation

#1 Codebase	#2 Dependencies	#3 Configuration	#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

Cloud Native Design - Codebase

One codebase tracked in revision control, many deploys



Cloud Native Design - Dependencies

Explicitly declare and isolate dependencies

Declaration

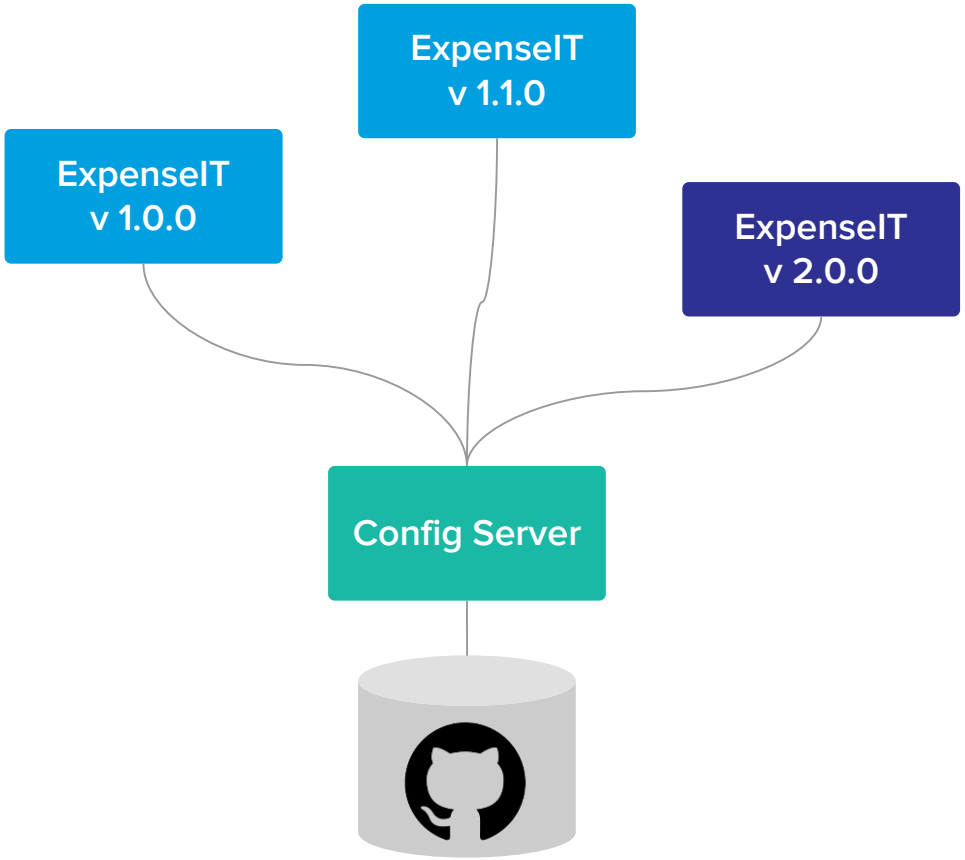
```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>
</dependencies>
```

Isolation

```
BOOT-INF/lib/
BOOT-INF/lib/spring-boot-starter-actuator-1.5.9.RELEASE.jar
BOOT-INF/lib/spring-boot-starter-1.5.9.RELEASE.jar
BOOT-INF/lib/spring-boot-starter-logging-1.5.9.RELEASE.jar
BOOT-INF/lib/logback-classic-1.1.11.jar
BOOT-INF/lib/logback-core-1.1.11.jar
BOOT-INF/lib/jul-to-slf4j-1.7.25.jar
BOOT-INF/lib/log4j-over-slf4j-1.7.25.jar
BOOT-INF/lib/snakeyaml-1.17.jar
```

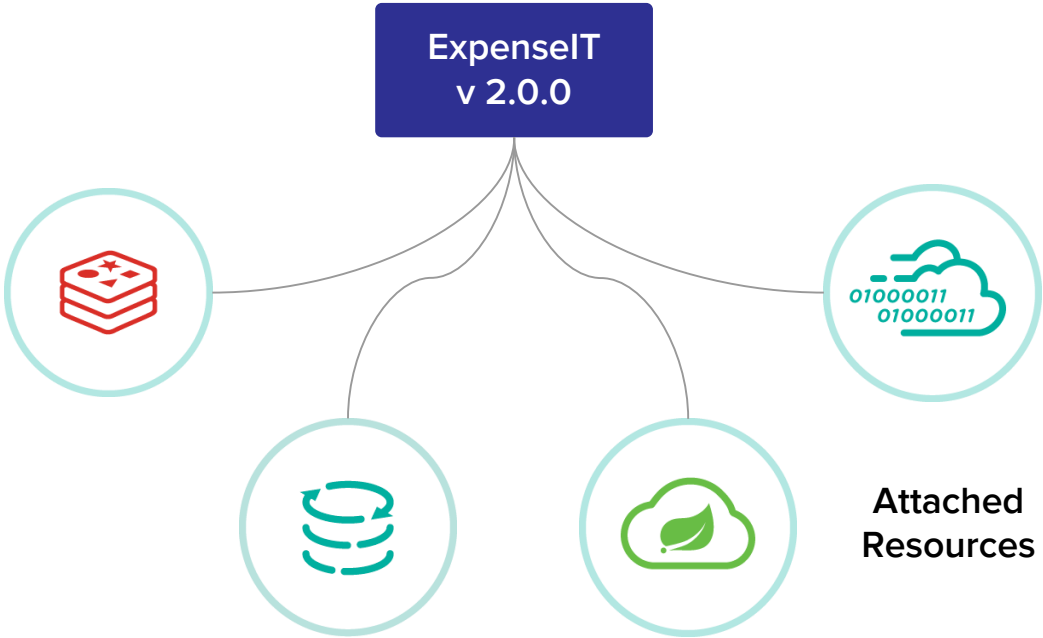
Cloud Native Design - Configuration

Store configuration external to application



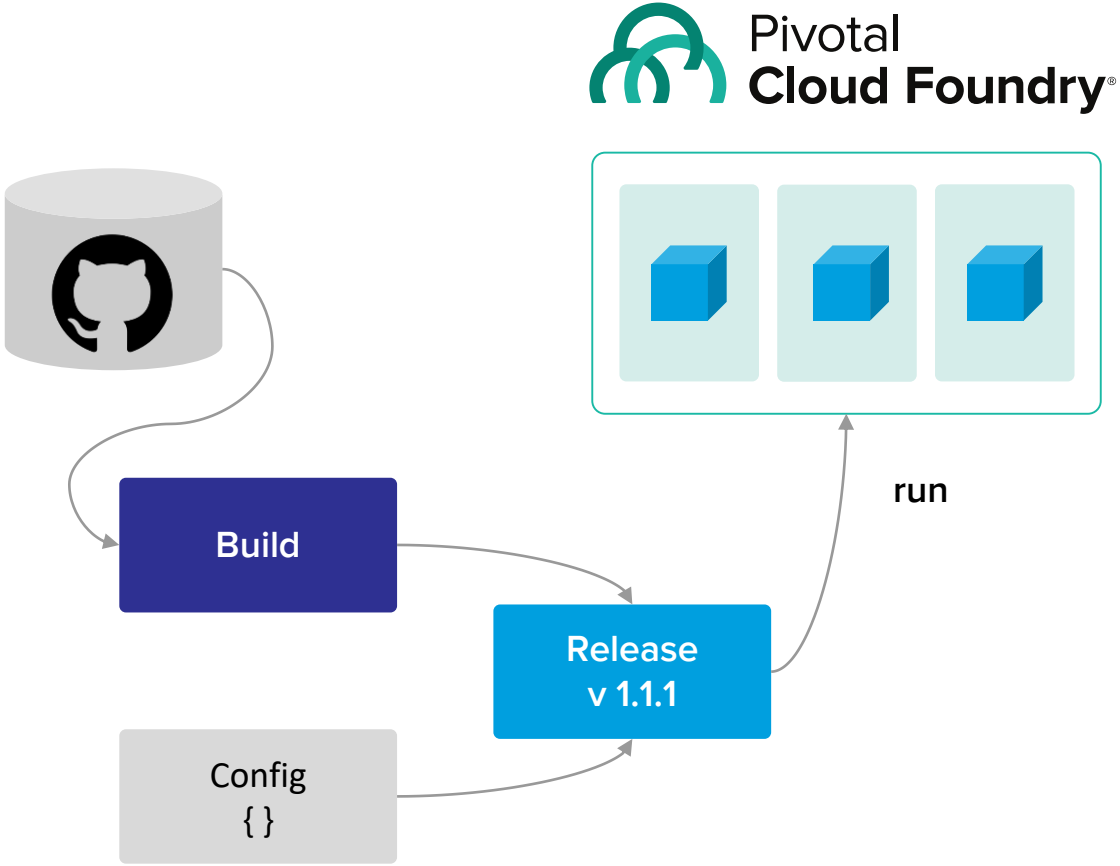
Cloud Native Design - Backing Services

Treat backing services as attached resources



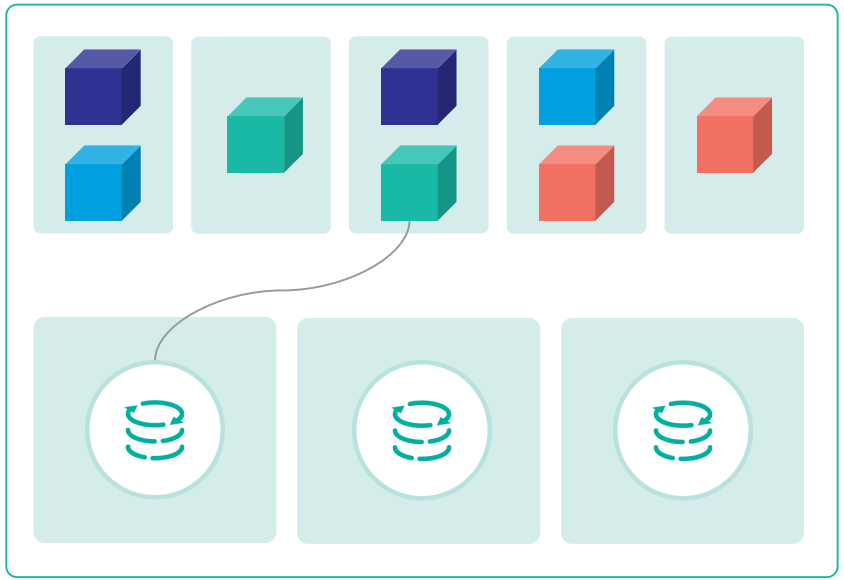
Cloud Native Design - Build, Release, Run

Strictly separate build, release and run phases



Cloud Native Design - Processes

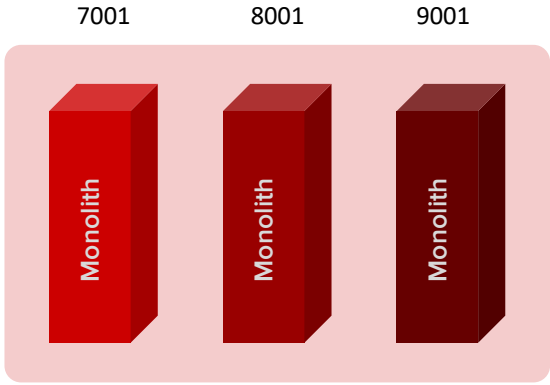
Execute the app as one or more stateless processes

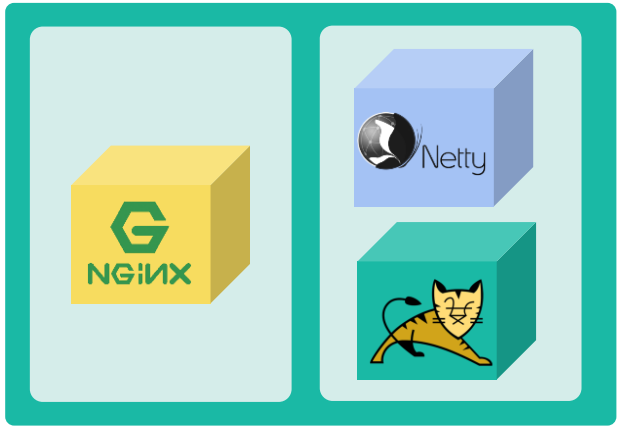


Cloud Native Design - Port binding

Export Services via Port binding

ORACLE WebLogic



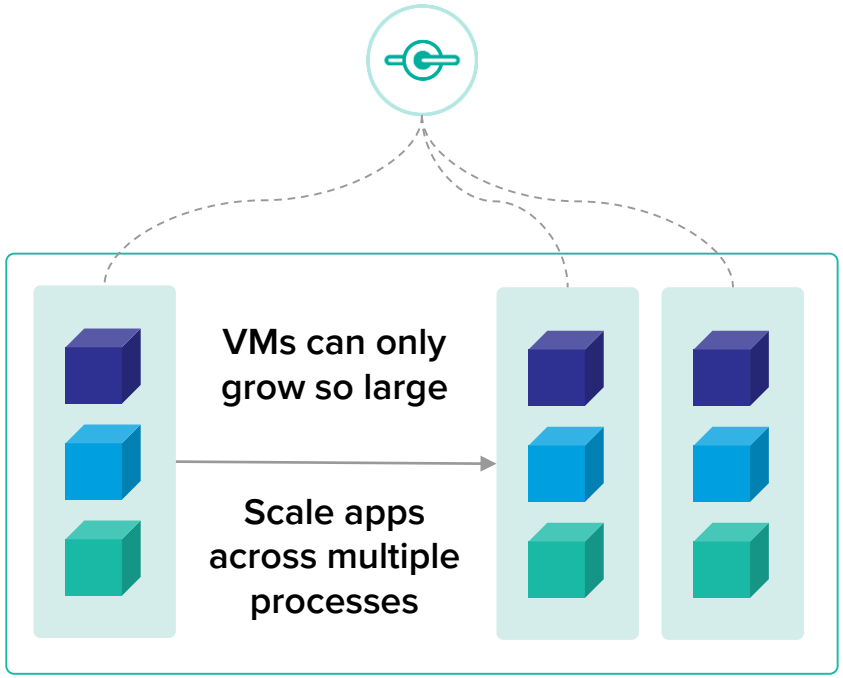


Pivotal Cloud Foundry®

- Self contained
- Inside out export services
- Apps can become backing services for other apps via Port binding

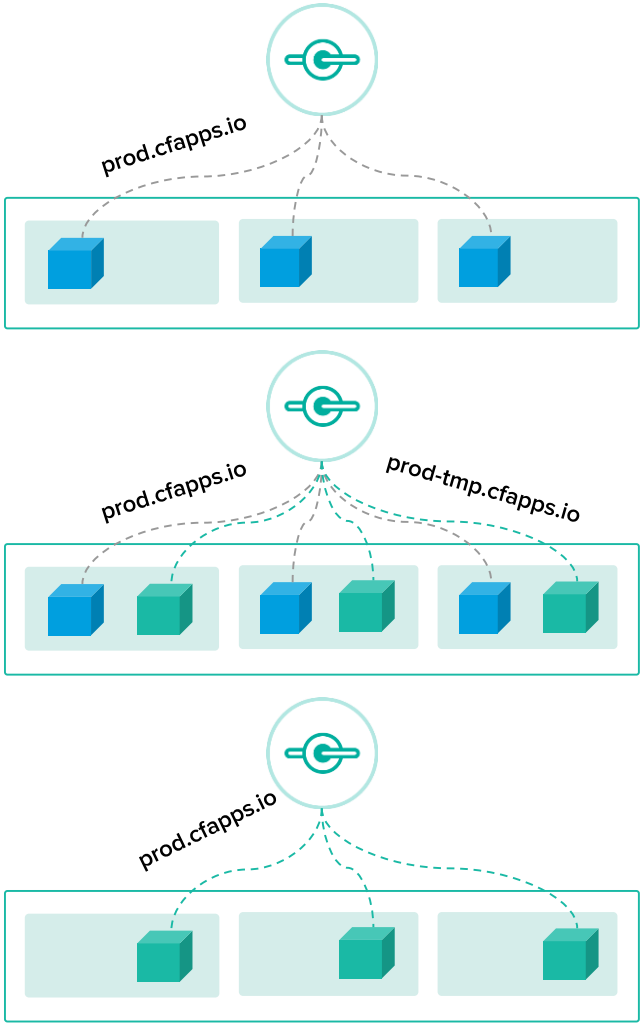
Cloud Native Design - Concurrency

Scale out via the process model



Cloud Native Design - Disposability

Maximum robustness with fast startup and graceful shutdown



Apps can start or stop at a moment's notice

Strive for fast startup

And graceful shutdown

Pivotal

Cloud Native Design - Dev/Prod Parity

Keep dev, staging and production as similar as possible

Time

Developer changes takes day, weeks or months to get into production.

Personnel

Developers develop code and Ops deploys code in Silos.

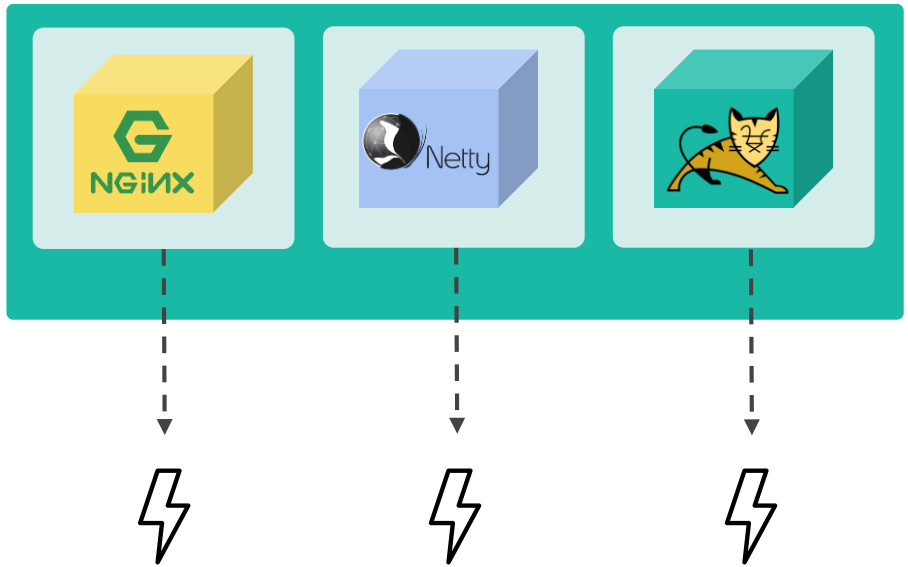
Technology

Developers use one technology in lower environments and company uses another in prod (i.e windows to linux)

	Traditional App	12-factor App
Time between deploys	days/weeks	mins/hours
Dev vs. Ops	different folks	same folks
Dev & Prod Environments	Divergent	Similar

Cloud Native Design - Logs

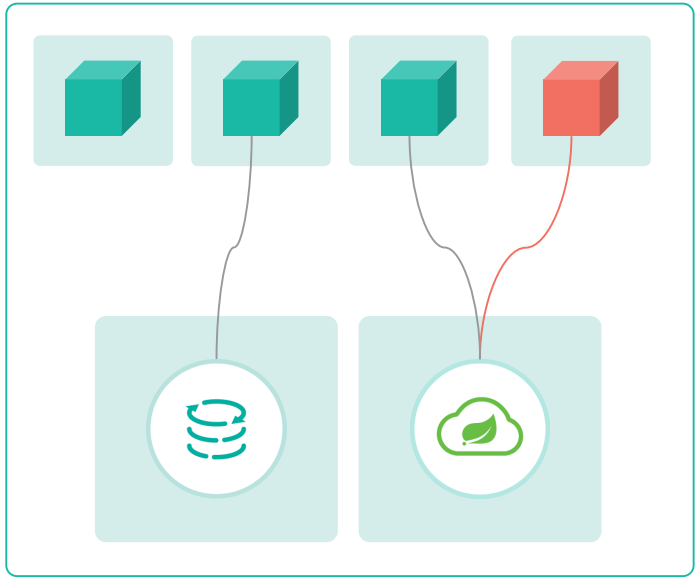
Treat logs as event stream



- Apps shouldn't manage logs
- No routing or storage for logs
- Unbuffered event stream to stdout

Cloud Native Design - Admin Processes

Run admin/management tasks as one-off processes



Admin Tasks

- DB Migrations
- Running a console
- Clean-up scripts
- Versioned with App
- Boot Actuators
- Boot DevTools

Cloud Native Implementation

#1 Codebase	#2 Dependencies	#3 Configuration	#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

Cloud Native Implementation

#1 Codebase 	#2 Dependencies 	#3 Configuration 	#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 

Pivotal

Cloud Native Evaluation

Cloud Native

- Microservice Architecture
- API first design

Cloud Resilient

- Design for failure
- Apps are unaffected by dependent service failure
- Proactive testing for failure
- Metrics and monitoring baked in
- Cloud agnostic runtime implementation

Cloud Friendly

- 12 Factor apps
- Horizontally scalable
- Leverage platform for HA

Cloud Ready

- No file system requirements
- Containerized
- Platform managed addresses and ports
- Consume Platform services

Transforming How The World Builds Software