Pivotal

Cloud Native Applications

Introduction to cloud native principles and architecture

Context

What's new at UBS?

- Pivotal Cloud Foundry as the platform of choice
- Program?
- Goals?

Agenda

- Cloud Native - what it is, why it matters, and how to measure value
- Microservices - evolution, principles & benefits
- 12-Factor Apps & the Cloud Native Maturity Model
- Tools
- Managing complexity

Cloud Native

From Outside-in

Cloud Native

Software designed to **run and scale reliably and predictably** on top of potentially unreliable **cloud-based** infrastructure

Pivotal

Source: Duncan C. E. Winn - "Cloud Foundry the Cloud Native Platform"

In the Cloud != Cloud Native

Motivations for Cloud Native

1. **Speed (Agility)**
   - Deliver value as *fast* as possible
2. **Stability**
   - Deliver value as *consistently* as possible
3. **Scalability**
   - Deliver more value when demand increases
4. **Savings**
   - Maximizing value relative to cost
5. **Security**
   - Protect value from unpredictable loss

Cloud Native User

Users of our software naturally think like this...

As a User

- ... I need to get things done quickly.
- ... I must deliver consistently and fail gracefully.
- ... I must deliver most when I'm needed most.
- ... I must get the best return for my effort.
- ... I must trust and be trusted to be effective.

Speed
Stability
Scalability
Savings
Security

Cloud Native Application

… so our applications must do the same

As an Application

- … I always respond within 30ms.
- … I gracefully degrade when services are down.
- … I discover new services automatically.
- … My code is only business logic & shared libraries.
- … I rotate credentials and encrypt end-to-end.

Speed
Stability
Scalability
Savings
Security

Cloud Native Platform

… and so must the platform

As a Platform

- … I start apps within 10 seconds.
- … I deploy with zero downtime.
- … I add instances when CPU > 80%.
- … I drop instances when CPU < 10%.
- … I encrypt all connections and credentials.

Speed
Stability
Scalability
Savings
Security

Cloud Native - Benefits & Possibilities

	Speed	Stability	Scalability	Savings	Security
User	✓	✗	✓	✓	✓
Application	✓	✗	✓	✓	✗
Services	✓	✗	✓	✗	✓
Platform	✓	✗	✗	✓	✓
Infrastructure	✓	✗	✓	✓	✓

Pivotal

How a User Sees Cloud Native

amazon Try Prime | Books | cloud native | Shop Back to School

Departments | Your Amazon.com | Today's Deals | EN | Hello. Sign in Account & Lists | Orders | Try Prime | 5 Cart

Books | Advanced Search | New Releases | NEW! Amazon Charts | Best Sellers & More | The New York Times® Best Sellers | Children's Books | Textbooks

Back to search results for "cloud native"

Cloud Native Java and over one million other books are available for **Amazon Kindle**. Learn more

Cloud Native Java: Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry 1st Edition

by Josh Long (Author), Kenny Bastani (Author)

Be the first to review this item

#1 New Release in Java Programming

Look inside

Kindle	Paperback	Other Sellers
$29.99	$37.49	from $36.03

Buy new $37.49

May take an extra 1-2 days to ship.

Ships from and sold by Amazon.com. Gift-wrap available.

List Price: ~~$59.99~~ Save: $22.50 (38%)

23 New from $37.49

How a User Sees Cloud Native

amazon Try Prime

Shop Back to School

Your Amazon.com Today's Deals Orders Try Prime

Books Advanced Search New Releases NEW! Amazon Charts Best Sellers & More The New York Times® Best Sellers Children's Books Textbooks

Back to search results for "cloud native"

Cloud Native Java and over one million other books are available for **Amazon Kindle**. Learn more

Cloud Native Java: Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry 1st Edition

Be the first to review this item

#1 New Release in Java Programming

Kindle $29.99 | **Paperback $37.49** | Other Sellers from $36.03

Buy new $37.49

May take an extra 1-2 days to ship.

Ships from and sold by Amazon.com. Gift-wrap available.

List Price: ~~$59.99~~ Save: $22.50 (38%)

23 New from $37.49

Cloud Native in Depth: Speed (Agility)

We need to measure to see changes in value

- **Measure**
  - ↑ features delivered (velocity), # of deployments, automation
  - ↓ interruptions to value delivery: bug fixes, outages, optimization
- **Tools**
  - **Framework:** Spring Boot
  - **Platform:** Pivotal Cloud Foundry
  - **Tracking features and Velocity:** JIRA
  - **CI/CD Pipelines:** Jenkins / SDP
  - **Code Quality and Safety:** BlackDuck, Fortify, SonarQube
- **Process**
  - **Agile development:** rapid iteration, constant improvement, concurrent user-research, design and engineering
  - **Test Driven Development:** Codifying behavior of software before implementation and throughout lifecycle

Cloud Native in Depth: Stability

We need to measure to see changes in value

- **Measure**
  - ↑ uptime: average, during deployments, during traffic spikes, code quality
  - ↓ downtime
- **Tools**
  - **Circuit-Breaker:** Spring Cloud Hystrix
  - **Platform:** Pivotal Cloud Foundry
- **Process**
  - **Bulkhead and Circuit-breaker patterns:** Application are written to be stable when internal and external functionality fails.
  - **DevOps:** Developer ownership of application stability

Cloud Native in Depth: Scalability

We need to measure to see changes in value

- **Measure**
  - ↑ transactions per second
  - ↓ latency, cost per transaction
- **Tools**
  - **Service Discovery:** Spring Cloud Eureka
  - **Distributed Config:** Spring Cloud Config
  - **Platform Autoscaling:** Cloud Foundry Autoscaler
- **Process**
  - **Microservice app patterns:** Independently deployable and scalable application components
  - **12 Factor:** Application development patterns for externalizing configuration, state, and build-time/deploy-time/runtime concerns

Cloud Native in Depth: Savings

We need to measure to see changes in value

- **Measure**
  - ↑ utilization of hardware, automation
  - ↓ person-time spent on non-value-add activities, unused infrastructure
- **Tools**
  - **Platform usage reporting:** AppDynamics, Pivotal Cloud Foundry Metrics
  - **Platform Autoscaling:** Cloud Foundry Autoscaler
- **Process**
  - **Continuous Integration / Continuous Delivery:** Automate integration, testing and deployment

Cloud Native in Depth: Security

We need to measure to see changes in value

- **Measure**
  - ↑ quality of security reviews, dev security knowledge, patch rate
  - ↓ custom code, unmaintained dependencies, attack vectors
- **Tools**
  - **Dependency scanning:** Black Duck
  - **Code Scanning**: SonarQube, Fortify
  - **Platform:** Lifecycle-tools, Chaos monkey
  - **SSO:** Cloud Foundry UAA with Identity Provider integration
  - **Framework:** Spring Security
- **Process**
  - **Security Triage:** First-class policies for vulnerability reporting, evaluation and mitigation.
  - Regularly scheduled security reviews with Cybersecurity teams

Key Takeaways

- Running and scaling reliably and predictably depends on:
  - Speed (Agility)
  - Stability
  - Scalability
  - Savings
  - Security
- Our users depend on multiple layers working in concert:
  - Application
  - Services
  - Platform
  - Infrastructure
- We optimize working with these layers by using Cloud Native tools and processes

Pivotal

Microservices

Microservices

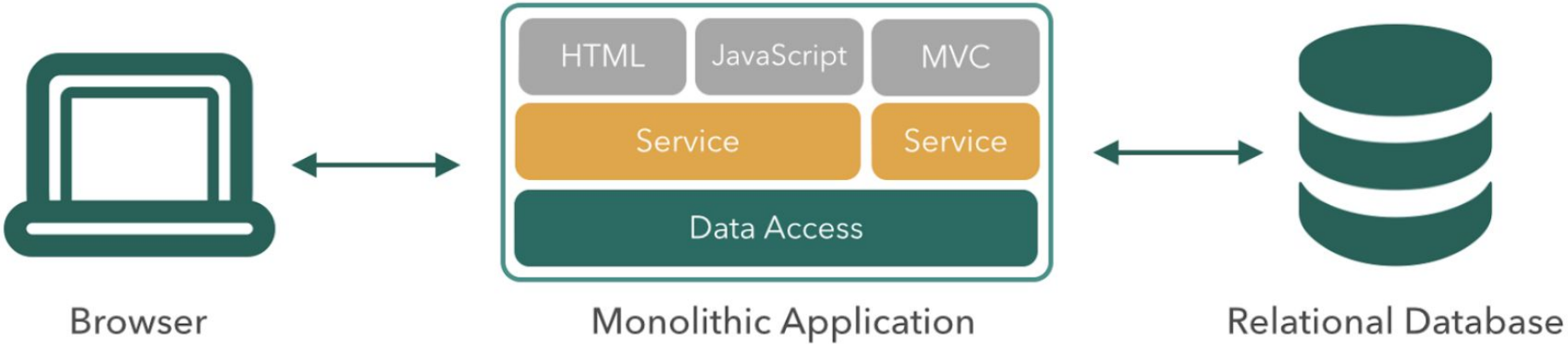
If services have to be updated together, they're not loosely coupled!

Loosely coupled service oriented architecture with bounded contexts

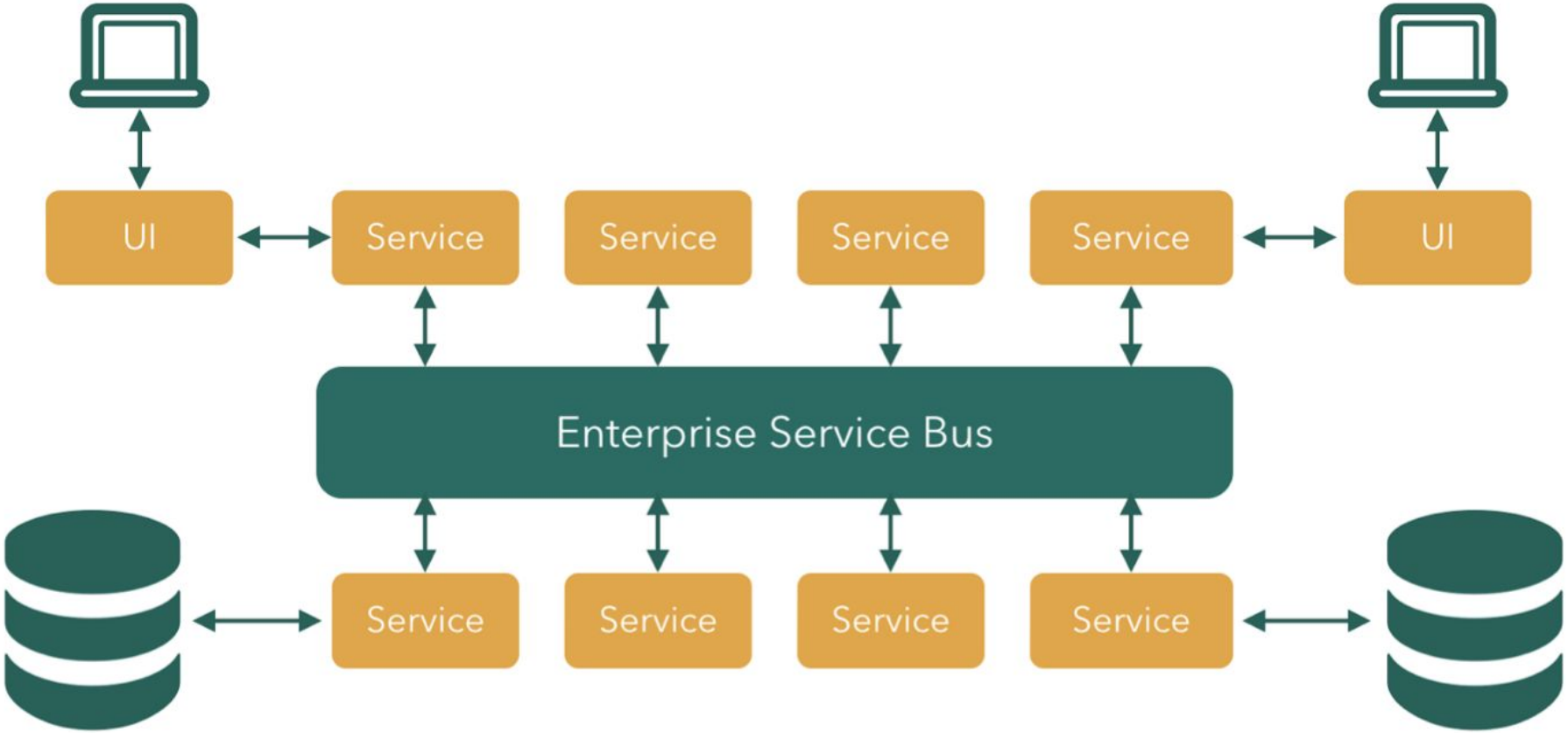
- Adrian Cockcroft

If you have to know about surrounding services, you don't have a bounded context

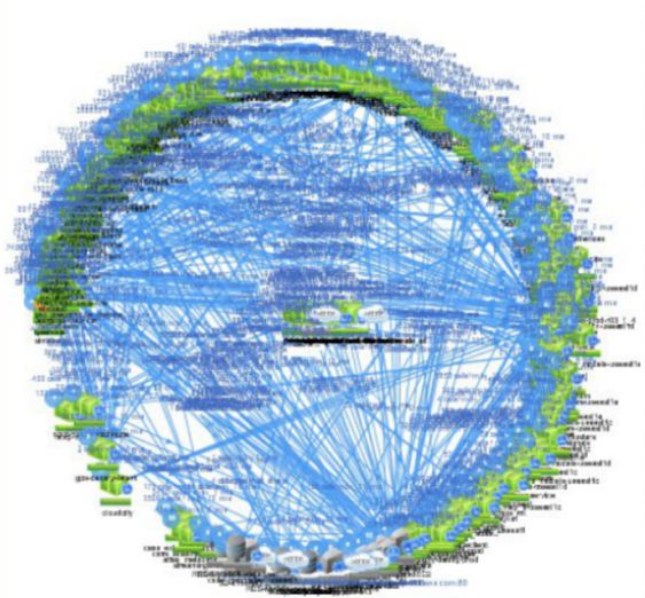
Not Monoliths



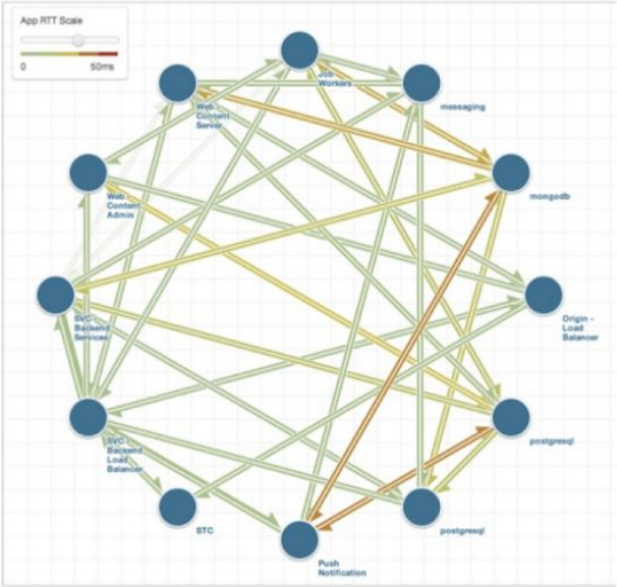
Not ESB-centric SOA



Microservice Pioneers



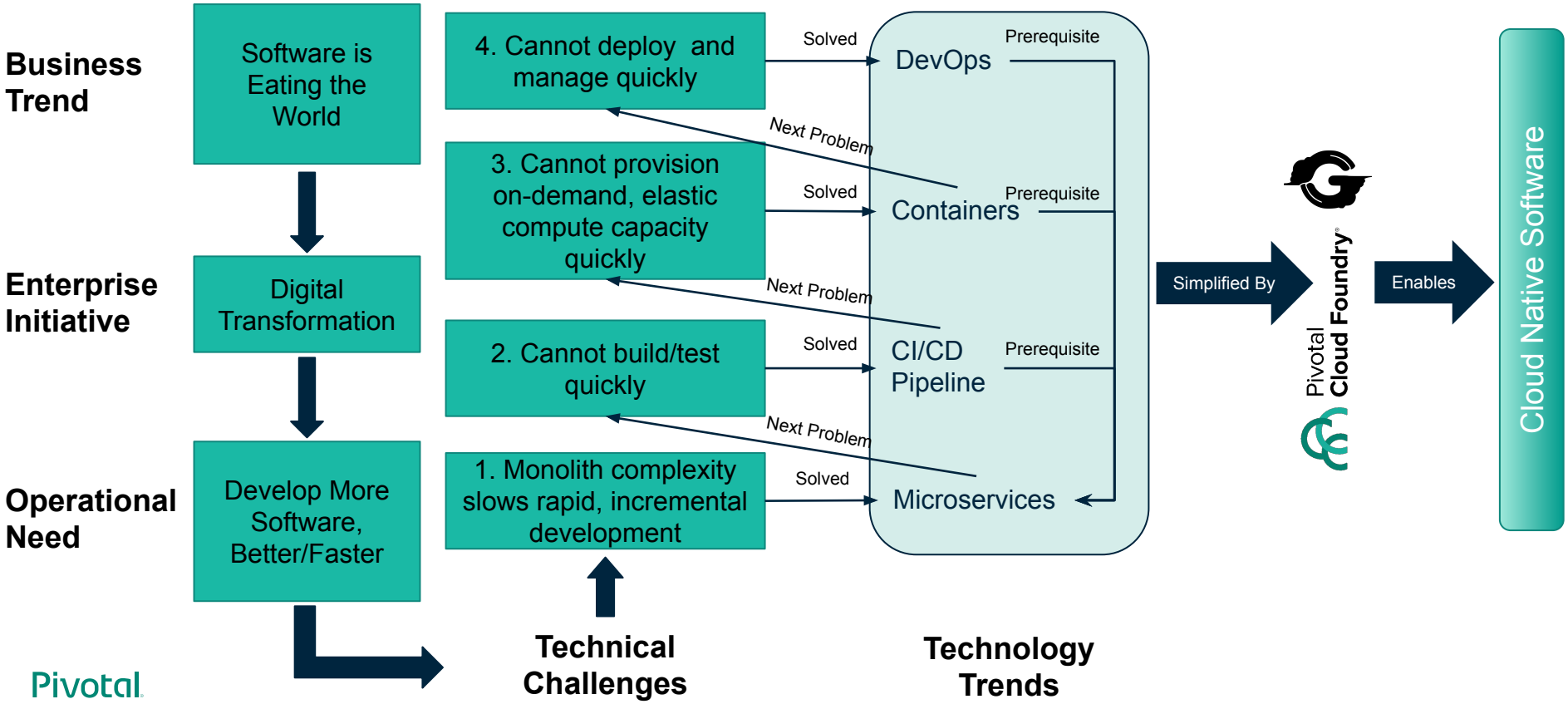
Netflix



Gilt Groupe (12 of 450)

Twitter

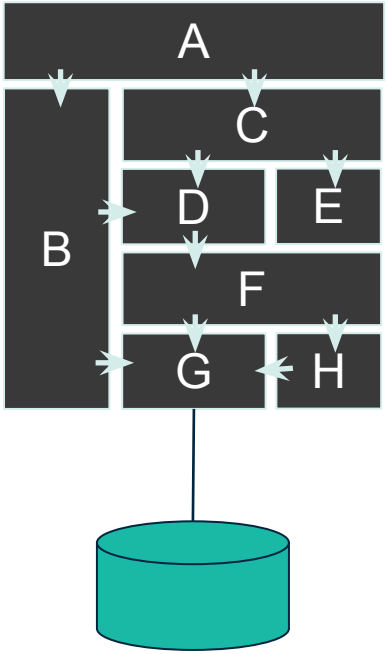
Converging Trends for Shorter Cycle Times



Pivotal

Monolith vs. Microservice

Interface

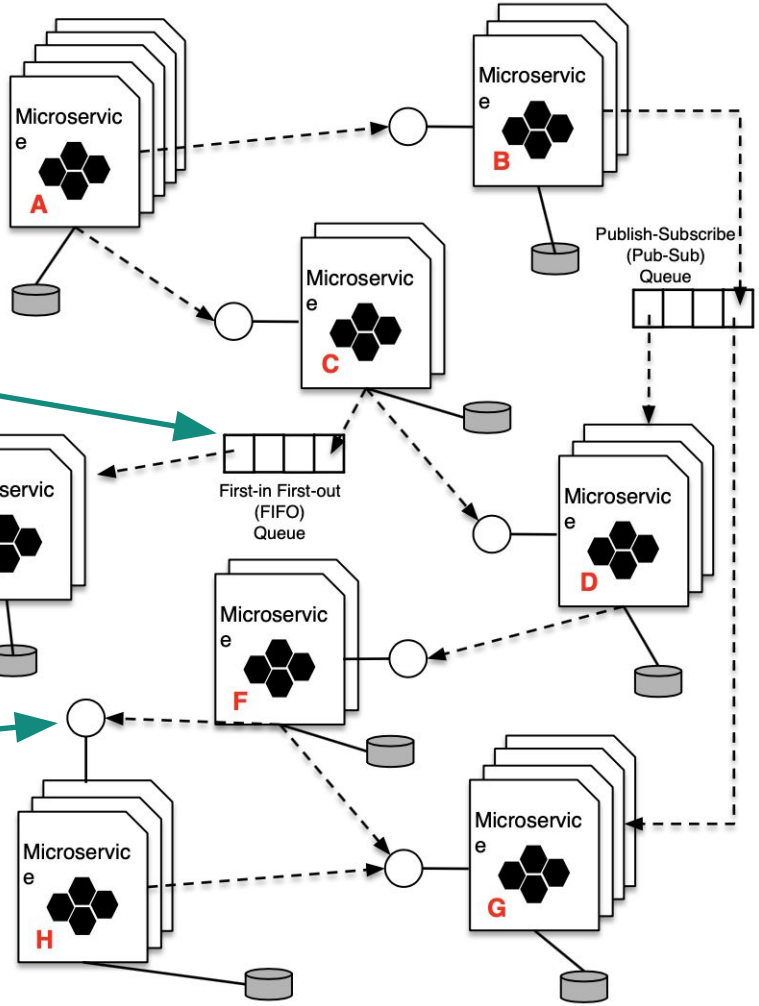


Queues, Pub-Sub and other messaging paradigms enable asynchronous processing and independent scaling.

Independent state

Explicit interface boundary

Independently scalable, e.g. Microservice H may have 3 container instances



Pivotal

Principles of Microservices

- Distributed
- Improved Resiliency
- Independent Lifecycle / Decoupled
- Facade External Dependencies
- Multiple Rates of Change
- Multiple Business Owners
- Multiple Teams
- Enable Independent Scalability
- Automation Centric
- Independent State/Storage

Pivotal

Benefits of Microservices

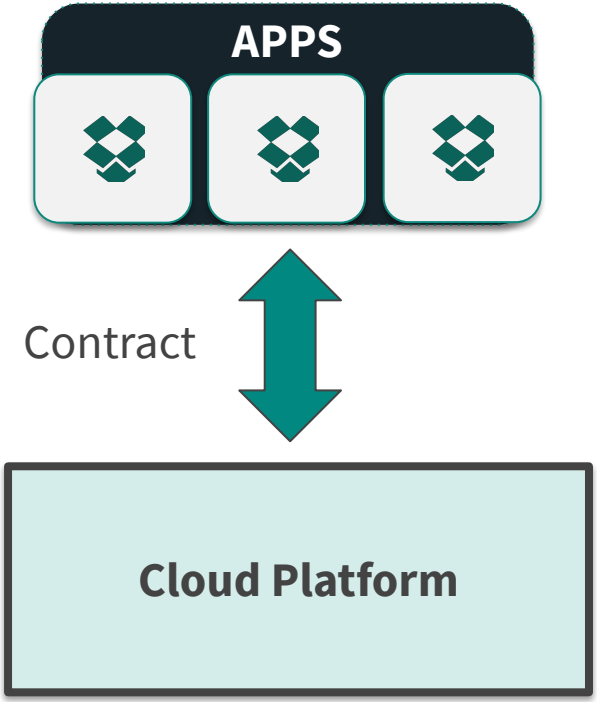
- Strong Module Boundaries
- Independent Deployment
- Technology Diversity
- High Scalability
- Resilience/Flexibility
- Easy to understand
- Follow the Single Responsibility Principle
- Easy to enhance

Pivotal

Principles of Cloud Native Applications

Principles establish a **contract between** cloud native **apps** and the underlying **platform**

- Cloud native apps recognize they are **ephemeral**
- Cloud native apps **minimize dependencies** on the underlying platform
- Twelve-Factor (http://12factor.net) is a popular and important set of principles that includes Dependencies, Config, Processes, and Disposability



12 Factor Applications

Cloud Native Design - 12 factors

Codebase
One codebase tracked in revision control, many deploys

Dependencies
Explicitly declare and isolate dependencies

Configuration
Store config in the environment

Backing services
Treat backing services as attached resources

Build, release, run
Strictly separate build and run stages

Processes
Execute the app as one or more stateless processes

Port Binding
Export services via ports

Concurrency
Scale out via the process model

Disposability
Maximize robustness with fast startup and graceful shutdown

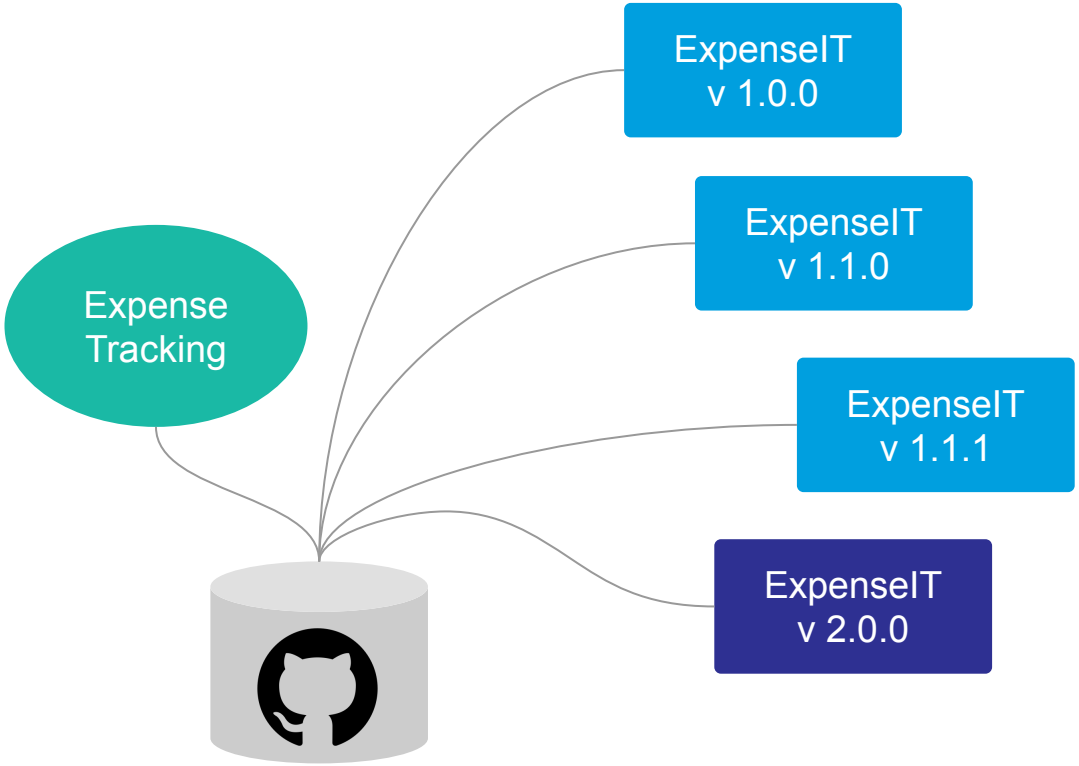
Dev/Prod Parity
Keep dev to prod as close as possible

Logs
Treat logs as event streams

Admin Processes
Run admin and management tasks as one-off processes

Cloud Native Design - Codebase

One codebase tracked in revision control, many deploys



Cloud Native Design - Dependencies

Explicitly declare and isolate dependencies

```
<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>
```

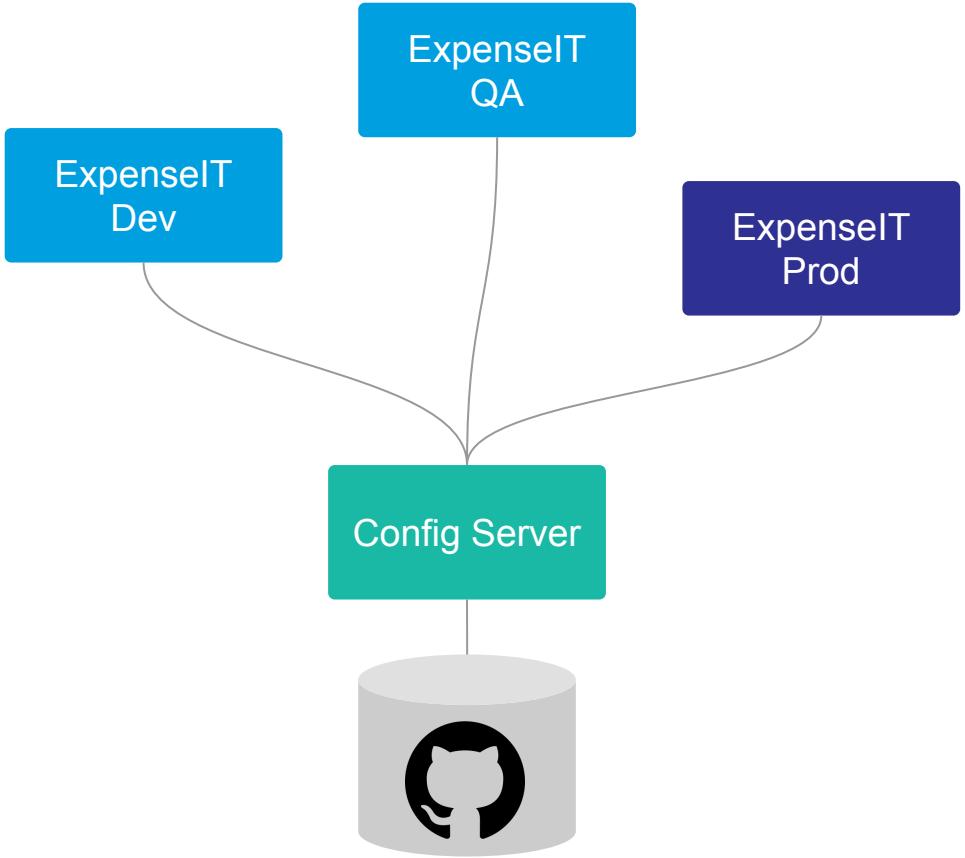
Declaration

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

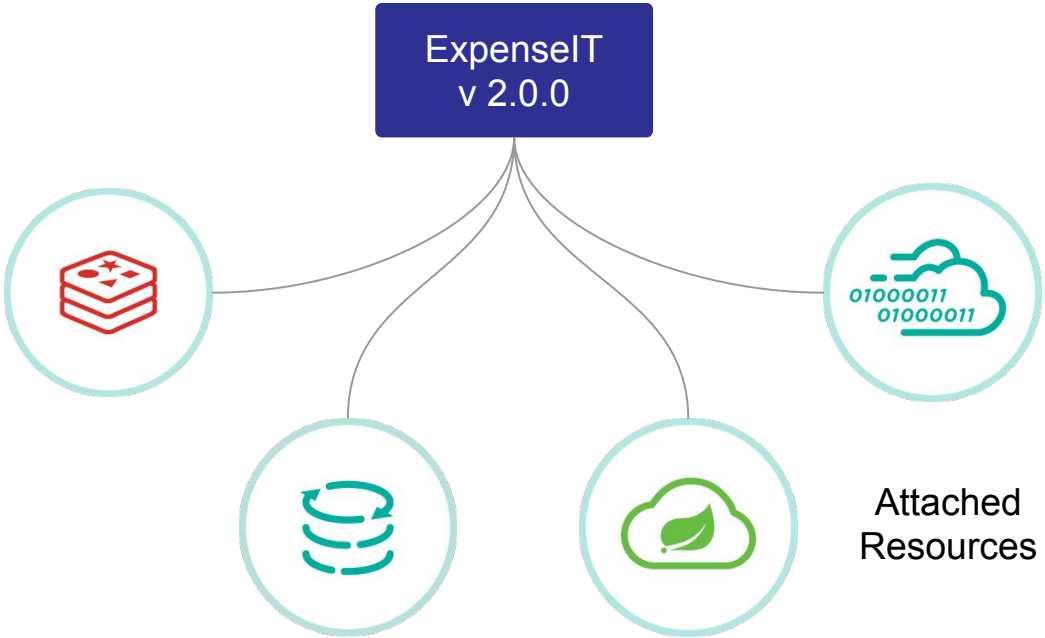
Store configuration external to application



Pivotal

Cloud Native Design - Backing Services

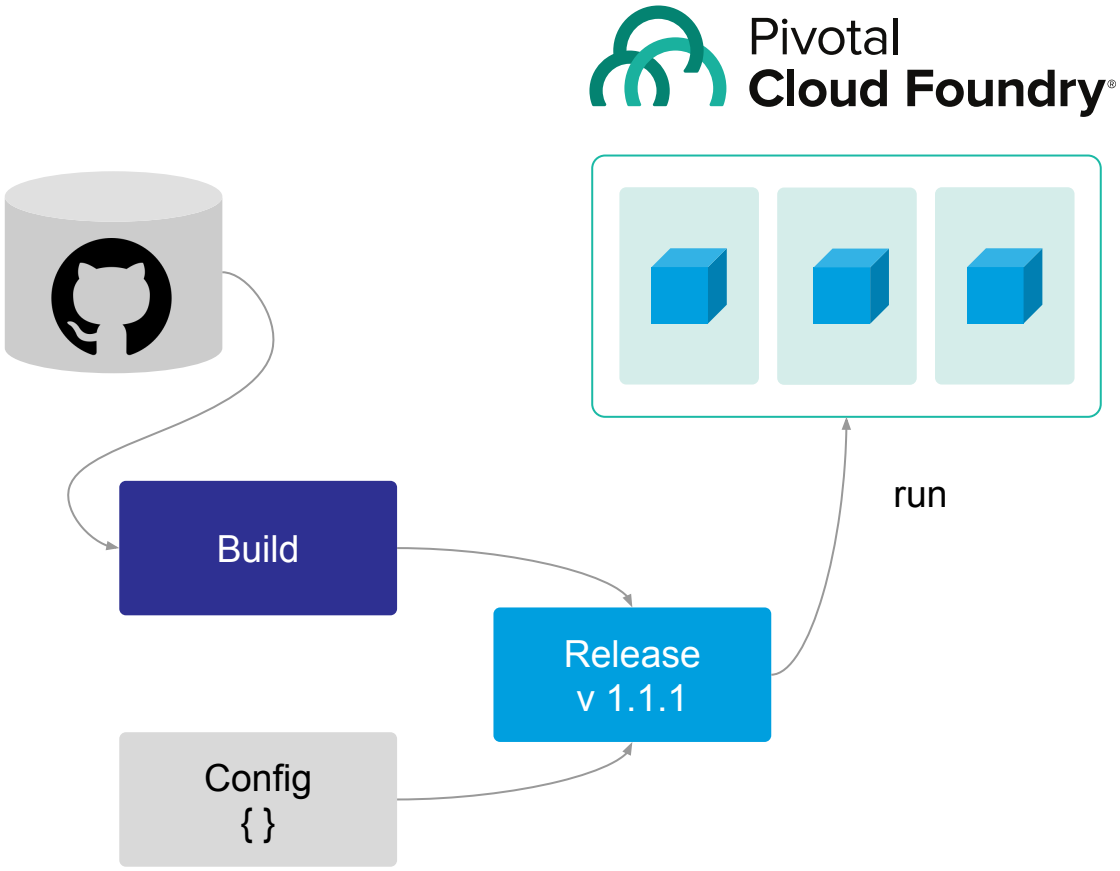
Treat backing services as attached resources



Pivotal

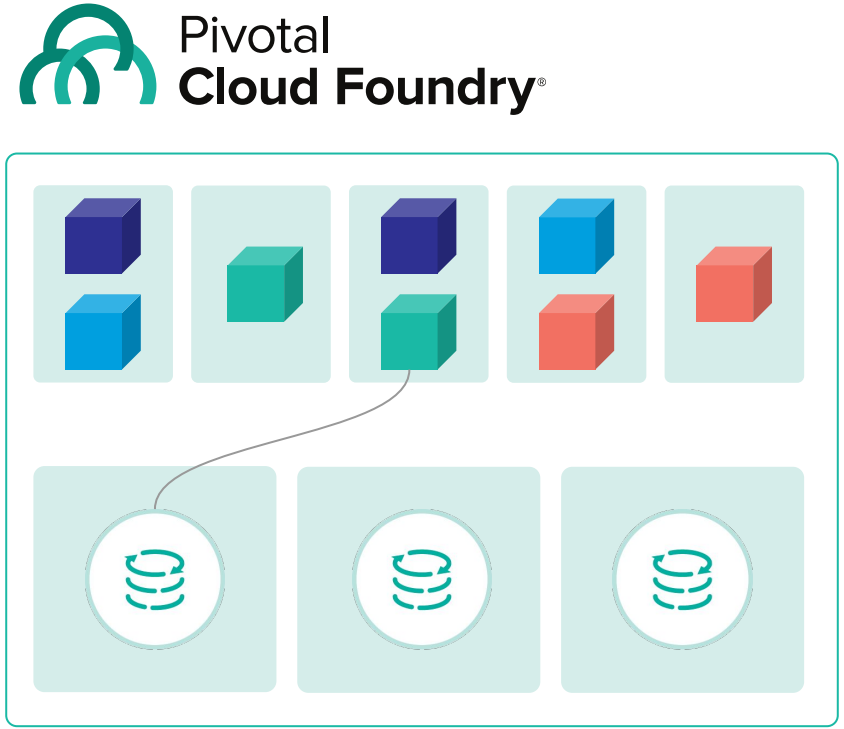
Cloud Native Design - Build, Release, Run

Strictly separate build, release and run phases



Cloud Native Design - Processes

Execute the app as one or more stateless processes

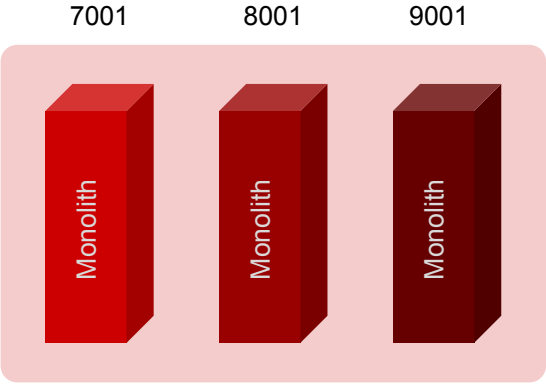


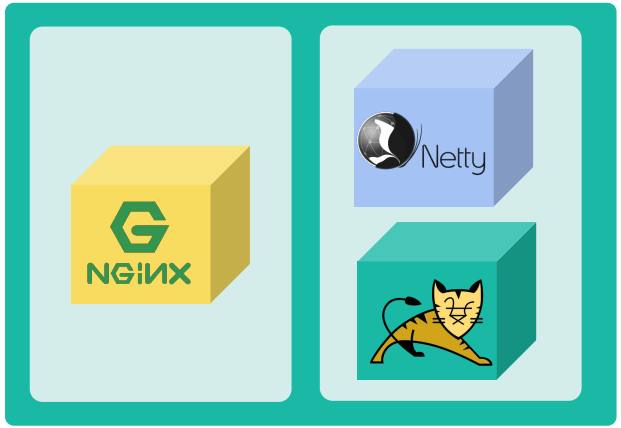
Stateless Processes

Stateful Backing Services

Cloud Native Design - Port binding

Export services via port binding

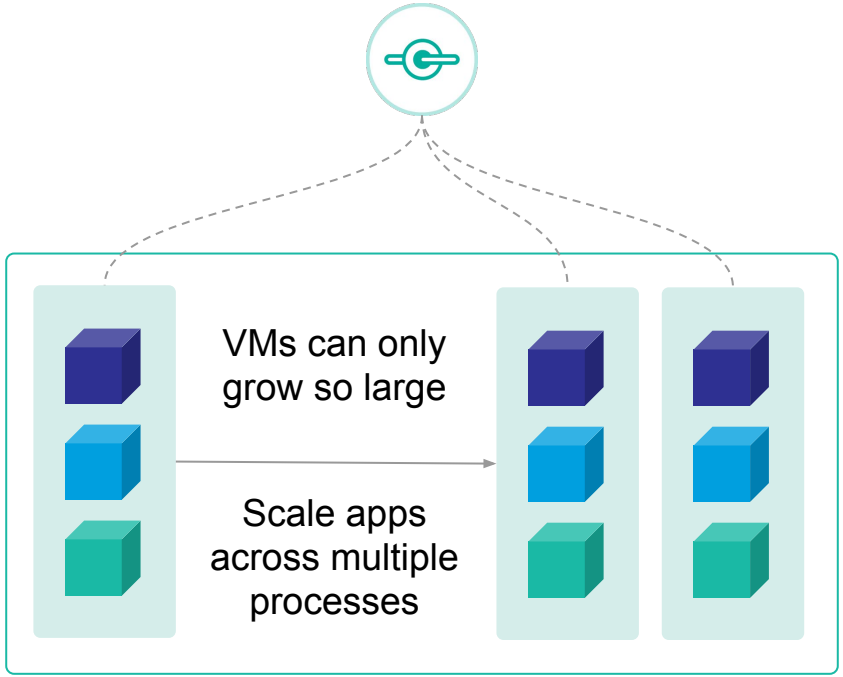




- Self contained
- Inside out export services
- Apps can become backing services for other apps via port binding

Cloud Native Design - Concurrency

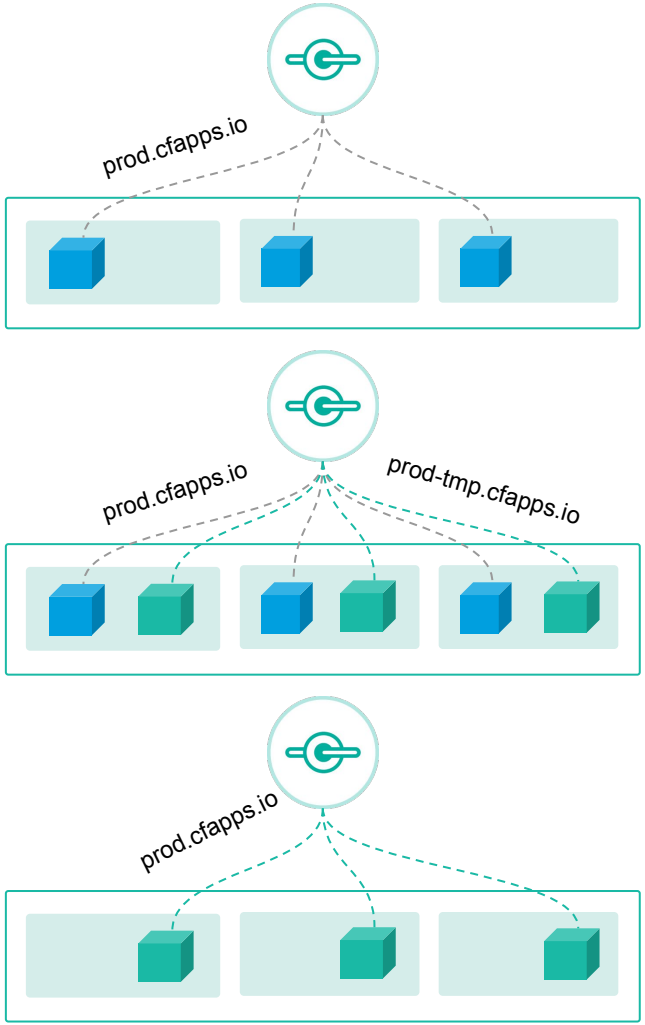
Scale out via the process model



Pivotal

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

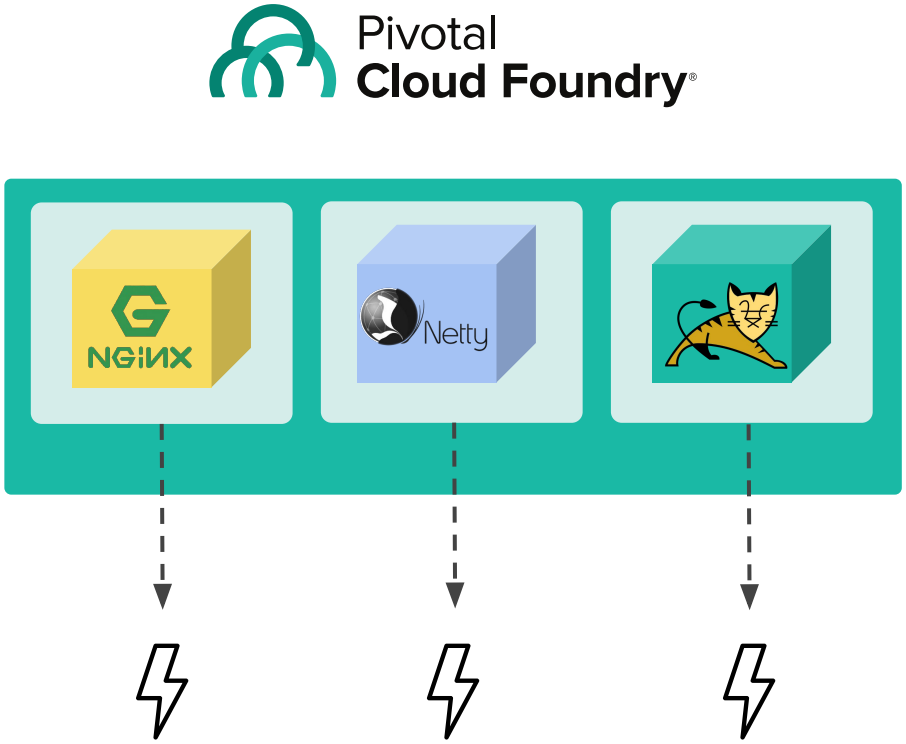
Developers use one technology in lower environments and company uses another in prod (i.e windows to linux)

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

Pivotal

Cloud Native Design - Logs

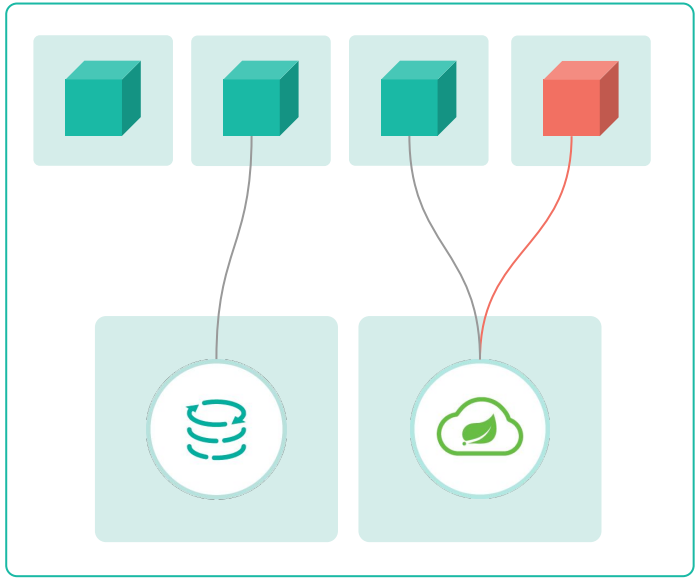
Treat logs as event stream



- 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

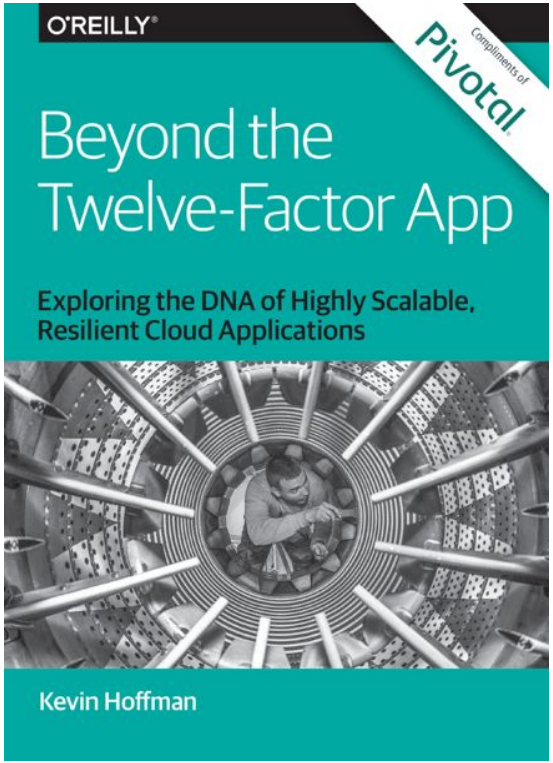
Cloud Native Maturity Model

Replatforming

VALUE & APPROACH

I. One Codebase, One App*	= **Time to Market**; find the seams; use good SDLC practices
II. Dependency Management*	= **Dev Productivity**; standardize & remove surprises
V. Build, Release, Run*	= **Release Mgmt Hygiene**; use CI/CD automation /w PCF
III. Configuration*	= **Release Mgmt Hygiene**; move to environment vars
XI. Logs*	= **Real-Time Metrics**; use PCF features; stdout / stderr
IX. Disposability	= **Auto-Scale**; move slow processes to backing services
IV. Backing Services	= **Resiliency / Agility**; use circuit breaker; loose binding
X. Environmental Parity*	= **Reliability**; use well architected PCF, get parity
XII. Administrative Process	= **Reliability**; move to backing service(s), expose as REST
VII. Port Binding*	= **Ops Efficiency**; use PCF features like routing, scaling, etc.
VI. Process	= **Cloud Compatibility**; move state to backing service(s)
VIII. Concurrency	= **Auto-Scale, ZDD**; design for cloud, use PCF features

Looking Beyond 12-Factors



- **12-Factor Published in 2012**
  - In context of Heroku
  - A LOT has changed
- **New Guidance**
  - Emphasis on Enterprise Java & PCF
  - 3 new "factors"
    - API First
    - Telemetry – APM, Logs, Domain-Specific
    - Authn / Authz – Security First Design
- **Must Read for Application Architects**

Tools

Structured Automation

- Groundwork for devops
- Self-service
- Rapid, automated provisioning
- Predictability and consistency
  - not ad-hoc!
- Visibility (monitoring & metrics)
- Continuous delivery on Day 2, too!

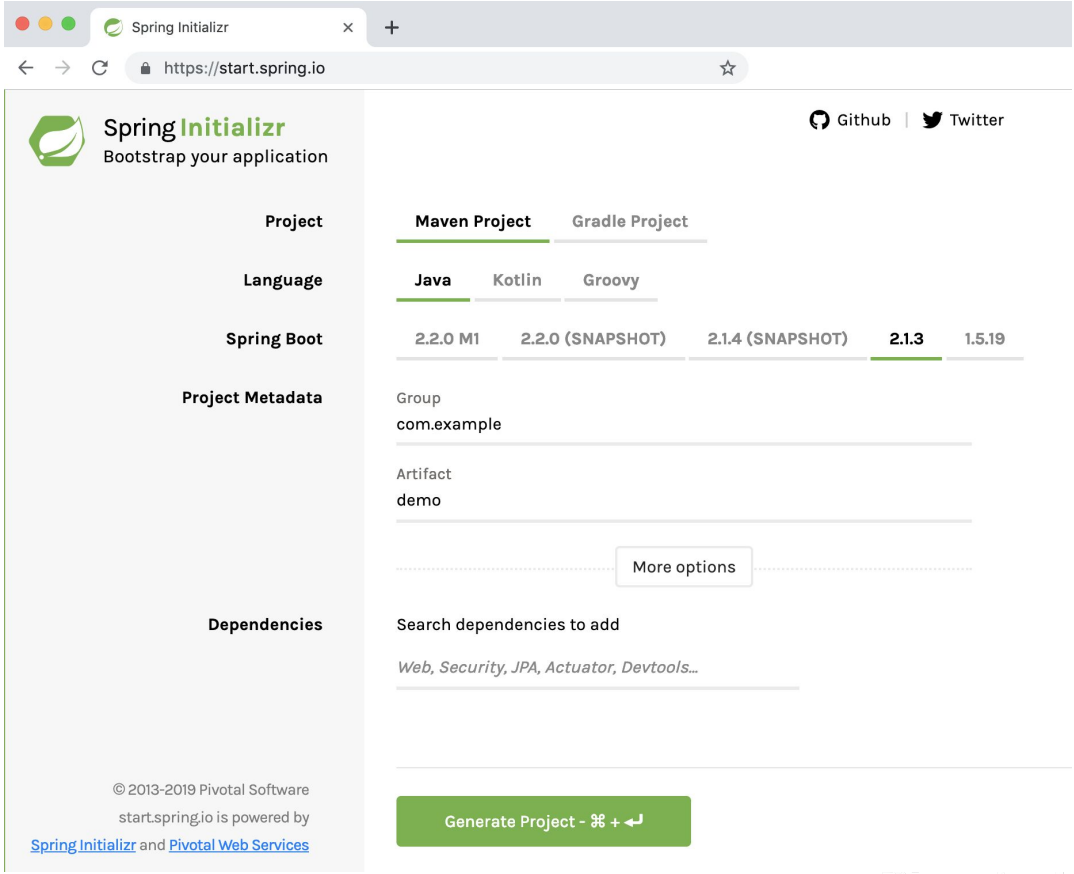
Spring Boot

- “Microframework”
- 12-Factor style
- Opinionated
- Convention over configuration
- Production ready
- Ops Friendly
  - Self contained
  - Health and metrics
  - Externally configurable

Pivotal

Spring Boot

- "Microframework"
- 12-Factor style
- Opinionated
- Convention over configuration
- Production ready
- Ops Friendly
  - Self contained
  - Health and metrics
  - Externally configurable



What’s the Catch?

Who Moved My Complexity?

- Microservices are individually simple
- Complexity is transferred to the ecosystem

Pivotal

Challenges in a Distributed System

- Configuration management
- Registration and discovery
- Routing and load balancing
- Fault tolerance and isolation
- Aggregation and transformation
- Monitoring and distributed tracing
- Process management

Pivotal

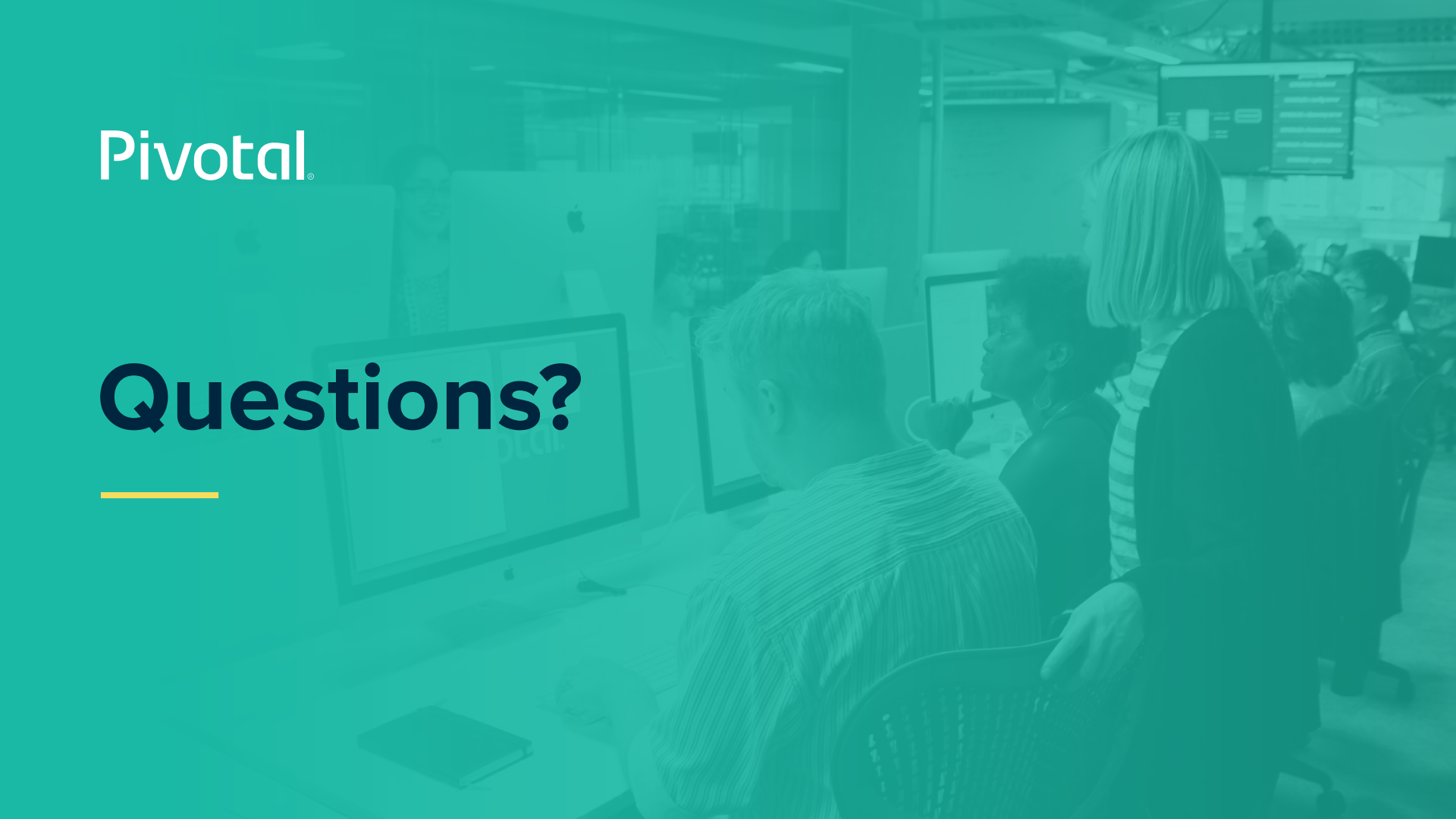
Next Session!

Cloud Native Architecture

...on the complexities
associated with microservice architectures and the
patterns and solutions we can use to resolve them!

Pivotal

Questions?



Pivotal

Thank You!