



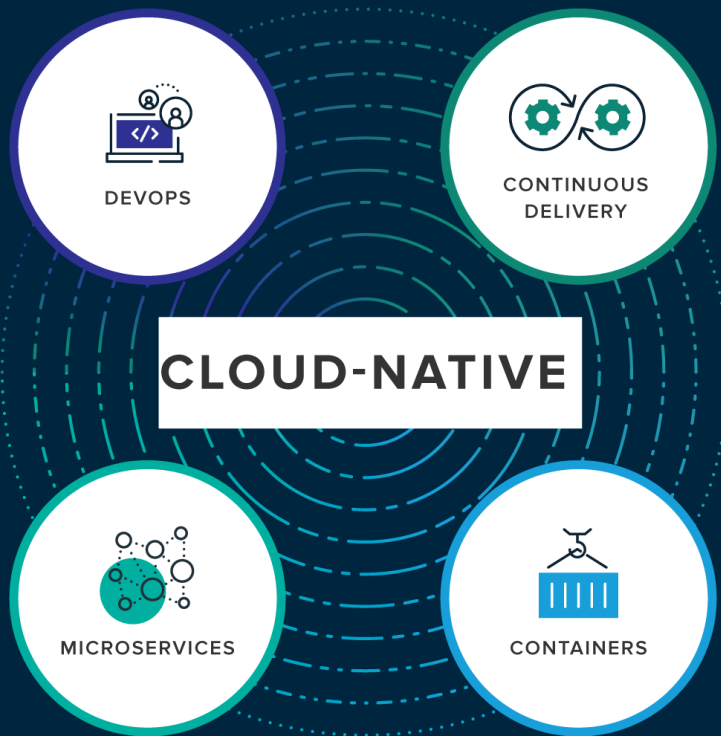
Cloud Native Applications - Going Cloud Native

Derrick Chua
Senior Platform Architect
tchua@pivotal.io
Jan 2020

Going Cloud Native

What is Cloud Native?

Pivotal's definition



Cloud-native is an **approach** to building and running applications that exploits the advantages of the cloud computing delivery model. Cloud-native is about **how** applications are created and deployed, **not where**.

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 Big Differences


CLOUD-NATIVE APPLICATIONS	TRADITIONAL ENTERPRISE APPLICATIONS
<p>Predictable. Cloud-native applications conform to a framework or “contract” designed to maximize resilience through predictable behaviors. The highly automated, container-driven infrastructure used in cloud platforms drives the way software is written. A good example of such a “contract” is illustrated by the 12 principles first documented as the 12-factor app.</p>	<p>Unpredictable. Traditional applications can't realize all of the benefits of running on a cloud-native platform due to the unique way each one is architected or developed. This type of application often takes longer to build, is released in big batches, can only scale gradually, and assumes high availability of dependent services.</p>
<p>OS abstraction. A cloud-native application architecture lets developers use a platform as a means for abstracting away from underlying infrastructure dependencies. Instead of configuring, patching, and maintaining operating systems, teams focus on their software. The most efficient means of abstraction is a formalized platform, for example, Pivotal Platform which is ideal for operating on cloud-based infrastructure such as Google Cloud Platform (GCP), Microsoft Azure, or Amazon Web Services (AWS).</p>	<p>OS dependent. Traditional application architecture allows developers to build close dependencies between the application and underlying OS, hardware, storage, and backing services. These dependencies make migrating and scaling the application across new infrastructure complex and risky, working against the cloud model.</p>
<p>Right-sized capacity. A cloud-native application platform automates infrastructure provisioning and configuration, dynamically allocating and reallocating resources at deploy time based on the ongoing needs of the application. Building on a cloud-native runtime optimizes application lifecycle management, including scaling to meet demand, resource utilization, orchestration across available resources, and recovery from failures to minimize downtime.</p>	<p>Over-sized capacity. Traditional IT designs a dedicated, custom infrastructure solution (“snowflake”) for an application, delaying deployment of the application. The solution is often over-sized based on worst-case capacity estimates with little capability to scale beyond to meet demand.</p>
<p>Collaborative. Cloud-native facilitates DevOps, a combination of people, process, and tools, resulting in a close collaboration between development and operations functions to speed and smooth the transfer of finished application code into production.</p>	<p>Siloed. Traditional IT operates an over-the-wall handoff of finished application code from developers to operations, which then runs it in production. Organizational priorities take precedence over customer value, resulting in internal conflict, slow and compromised delivery, and poor staff morale.</p>

The Big Differences

CLOUD-NATIVE APPLICATIONS	TRADITIONAL ENTERPRISE APPLICATIONS
<p>Continuous delivery. IT teams make individual software updates available for release as soon as they are ready. Organizations that release software rapidly get a tighter feedback loop and can respond more effectively to customer needs. Continuous delivery works best with other related approaches including test-driven development and continuous integration.</p>	<p>Waterfall development. IT teams release software periodically, typically weeks or months apart, when code has been built into a release despite the fact that many of the components of the release are ready earlier and have no dependency other than the artificial release vehicle. Features that customers want or need are delayed and the business misses opportunities to compete, win customers, and grow revenue.</p>
<p>Independent. Microservices architecture decomposes applications into small, loosely coupled independently operating services. These services map to smaller, independent development teams and make possible frequent, independent updates, scaling, and failover/restart without impacting other services.</p>	<p>Dependent. Monolithic architectures bundle many disparate services into a single deployment package causing unnecessary dependencies between services and leading to a loss of agility during development and deployment.</p>
<p>Automated scalability. Infrastructure automation at scale eliminates downtime due to human error. Computer automation faces no such challenge, consistently applying the same set of rules across any size of deployment. Cloud-native also goes beyond the ad-hoc automation built on top of traditional virtualization-oriented orchestration. A fully cloud-native architecture is about automating systems, not servers.</p>	<p>Manual scaling. Manual infrastructure includes human operators that manually craft and manage server, network, and storage configurations. At scale, operators are slow to correctly diagnose issues and easily fail to correctly implement at scale due to the level of complexity. Hand-crafted automation recipes have the potential to hard-code human errors into the infrastructure.</p>
<p>Rapid recovery. The container runtime and orchestrator provides a dynamic, high-density virtualization overlay on top of a VM, ideally matched to hosting microservices. Orchestration dynamically manages placement of containers across a cluster of VMs to provide elastic scaling and recovery/restart in the event of app or infrastructure failure.</p>	<p>Slow recovery. VM-based infrastructure is a slow and inefficient foundation for microservice-based applications because individual VMs are slow to startup/shutdown and come with large overhead even before deploying application code to them.</p>

Going Cloud Native

Why Cloud Native?



“One of the things we've learned is that if you can't get it to market more quickly, there is no doubt that the market will have changed and no matter how well you've engineered it or built it or deployed it or trained your folks, it's not going to be quite right because it's just a little too late.”

James McGlennon

Executive VP and CIO, **Liberty Mutual Insurance Group**

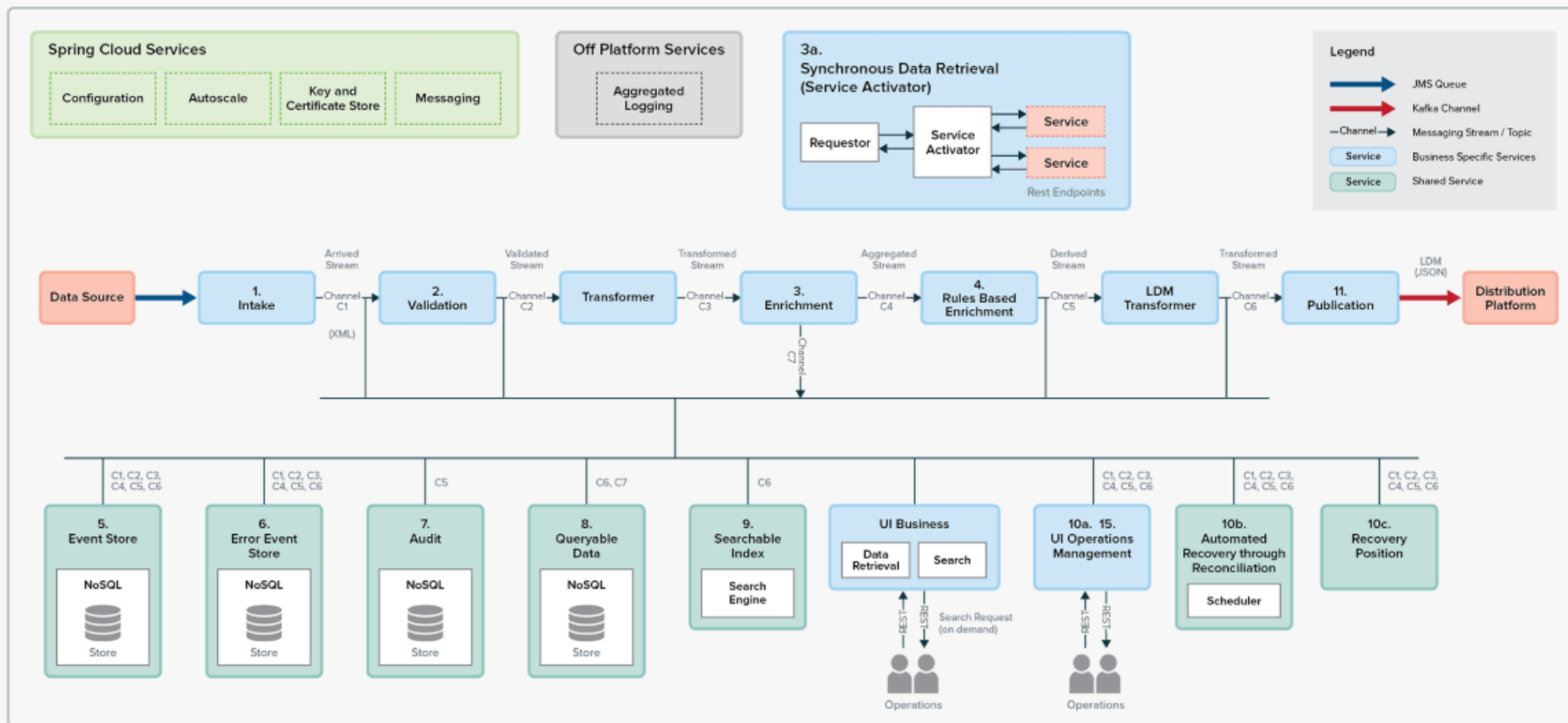
21 applications submitted for up to 5 Singapore digital bank licences: MAS



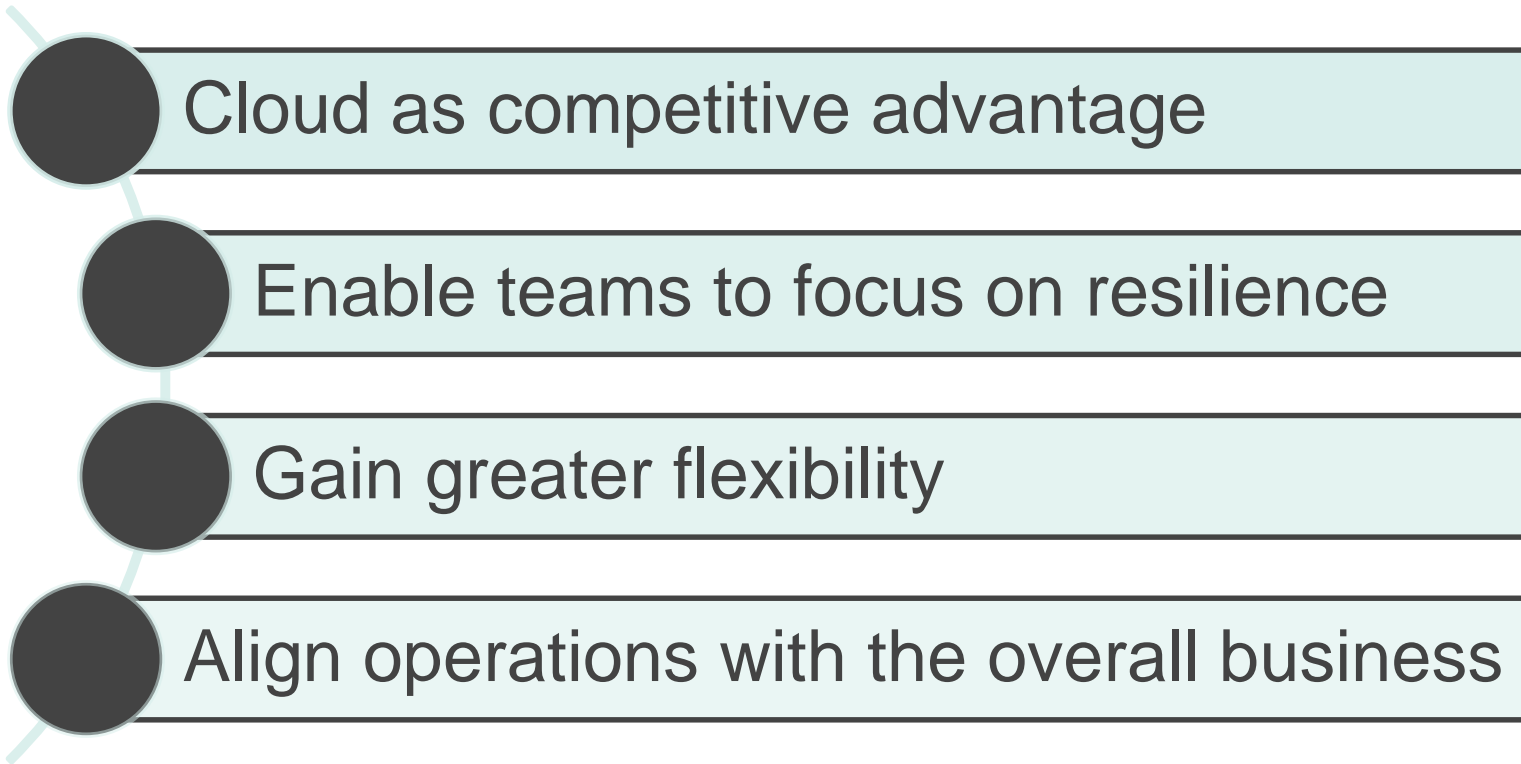
The Monetary Authority of Singapore is issuing up to five digital banking licences by the middle of this year - two full-bank licences that permit retail banking, and three for wholesale banking. PHOTO: ST FILE

Industry Trend	Cloud-Native Advantage
<p>Rising consumer expectations. Consumers now expect always-on access to their financials. They want it delivered in real-time, across many different devices. They want online banking, via self-service. This can be a challenge for banks, especially those that still depend on legacy tech up and down the stack. Investment banks face many of these same challenges.</p>	<p>With a cloud-native application, the priority is an API-first design. Developers can deliver features to users across across many devices. Great APIs also deliver a zippy user experience.</p>
<p>Increased competition from FinTech. Software is eating the financial services industry. Startups are gaining market share on the strengths of their software. They can rapidly improve it based on customer feedback. These startups have many advantages compared to incumbents. A better culture, leaner processes, and a lack of technical debt to name a few. We've seen this disruption pattern play out in many industries the last 5 years, and now it's happening in FinServ.</p>	<p>Three things help you become a software-defined business. First, a microservices architecture. This encourages rapid iteration, and small changes to discrete components of a system. Next, continuous delivery gives developers a frictionless path to production. This leads to frequent deployments of small changes that add up to big benefits over time. Finally, a DevOps culture eschews traditional developer and operator roles and objectives. Instead, the goal is to deliver value to the customer.</p>
<p>Regulatory requirements & compliance. IT systems, processes, and applications need to keep pace with a tidal wave of industry regulations and compliance standards. That demands significant time, attention, and budget.</p>	<p>Cloud-native approaches prize automation and a "securityfirst" approach to feature design. These two points are crucial to compliance. Highly automated systems log every activity. It's easy to see what changed, when, and who triggered the update. Further, automated systems can be rapidly patched and updated with new bits often. And automated systems generally need much less day-to-day human involvement, boosting security.</p> <p>"Security first" means that engineers design security into features from the start. They are not bolted on later.</p>
<p>Security. The threat landscape is constantly evolving and changing. Banks must contend with malware, advanced persistent threats (APTs), and the specter of leaked credentials.</p>	<p>Conventional wisdom within enterprise security circles advises "go slow to reduce risk. In the cloud-native world, it's "go fast to reduce risk." Why? Systems that change often are far less vulnerable to malware and other threats. Static environments are where bad actors thrive, and wreak havoc on your enterprise.</p> <p>Use automation to rapidly apply the latest patches to systems. Embrace immutable infrastructure concepts and re-deploy your stack to a "last known good state" often. And rotate credentials often, so that any leaked creds quickly expire and become worthless.</p>

Reference Architecture for Data Ingestion based on Spring Cloud Stream



Why cloud-native applications matter



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

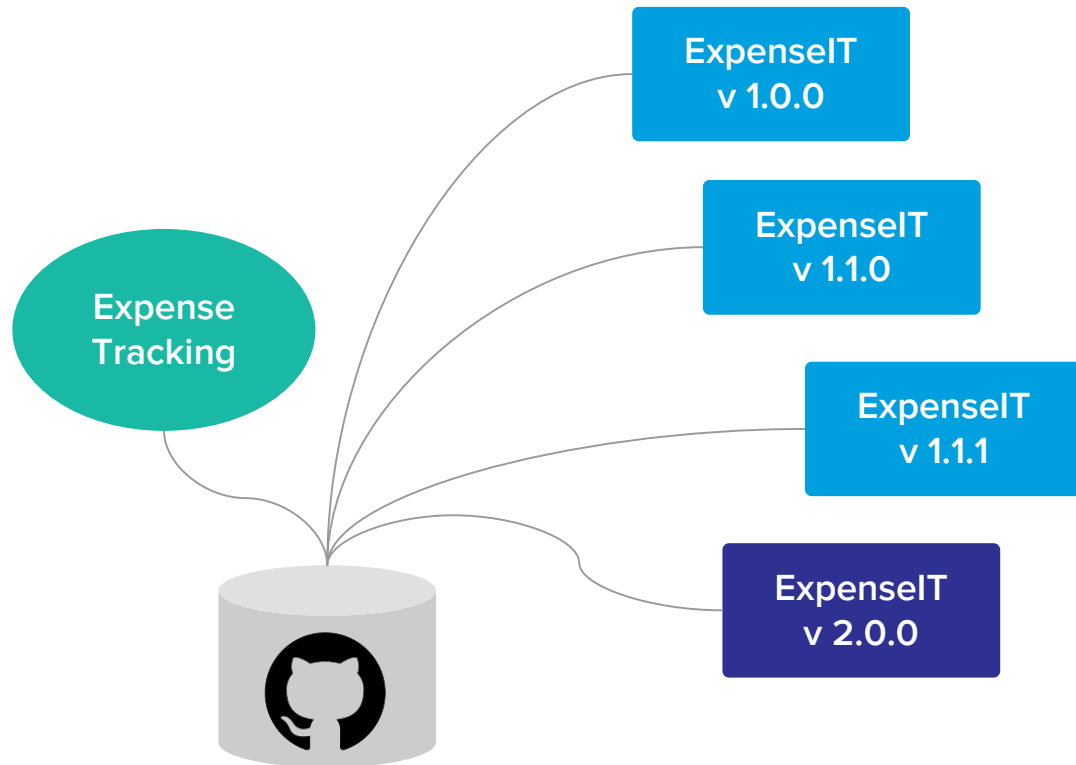
#10 Dev/Prod Parity

#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

```
<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>
```

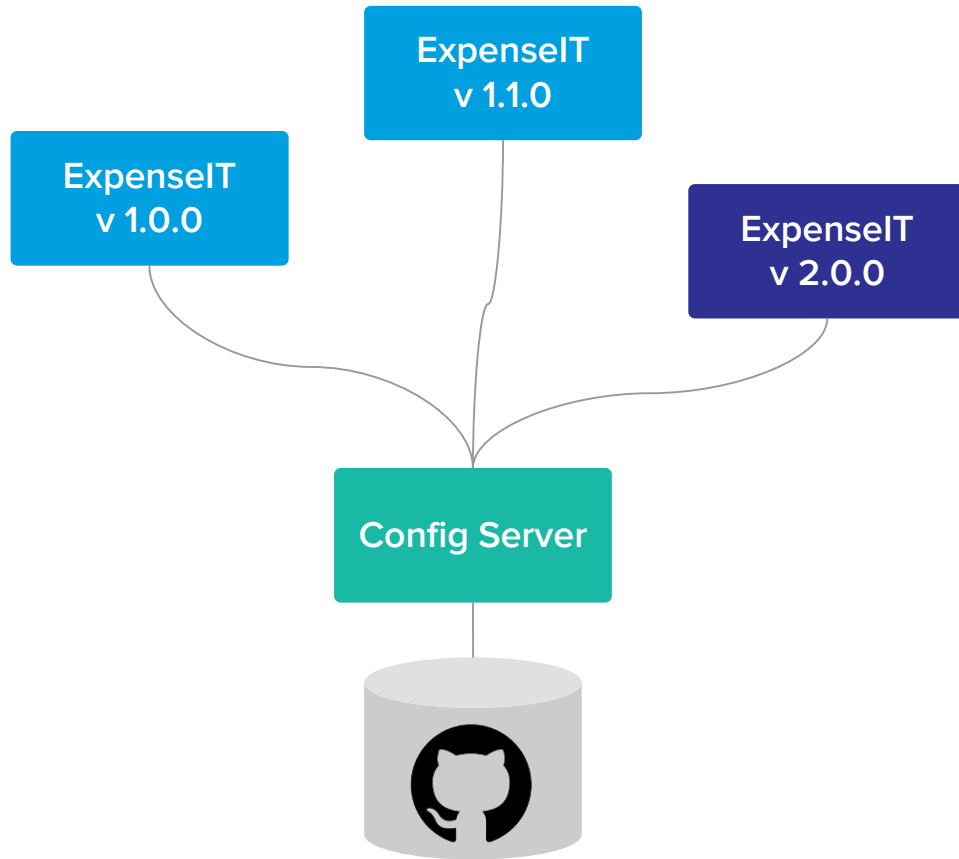
Declaration

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
```

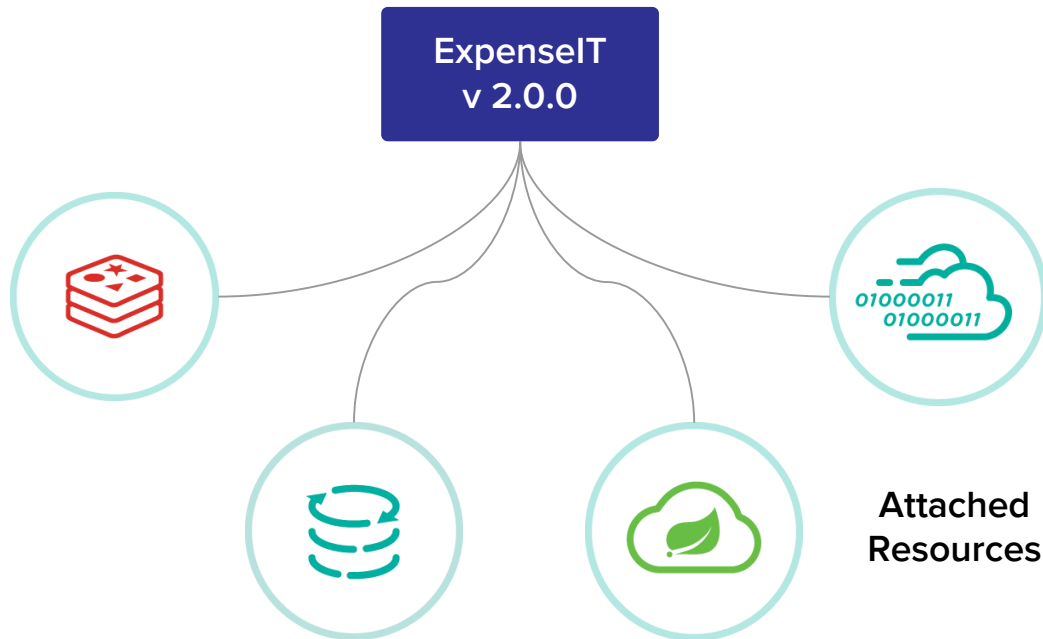
Cloud Native Design - Configuration

Store configuration external to
application



Cloud Native Design - Backing Services

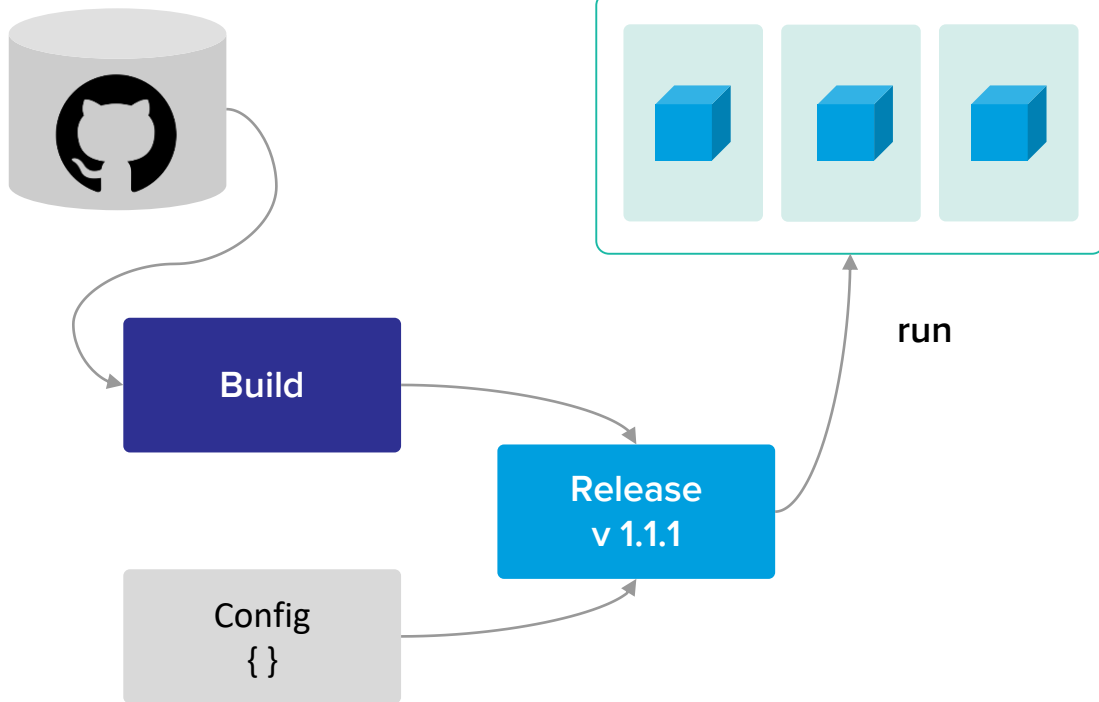
Treat backing services as attached resources



Cloud Native Design - Build, Release, Run

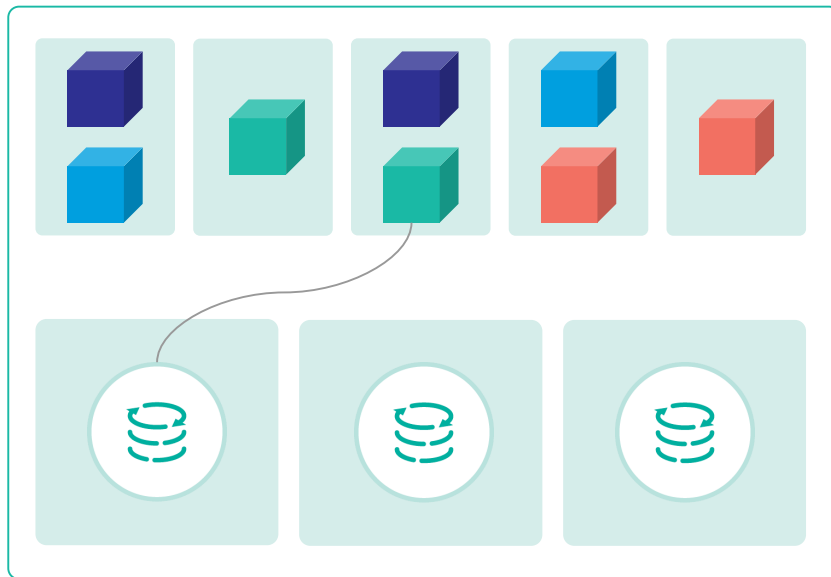
Strictly separate build, release and
run phases

Pivotal



Cloud Native Design - Processes

Execute the app as one or more
stateless processes



Stateless
Processes

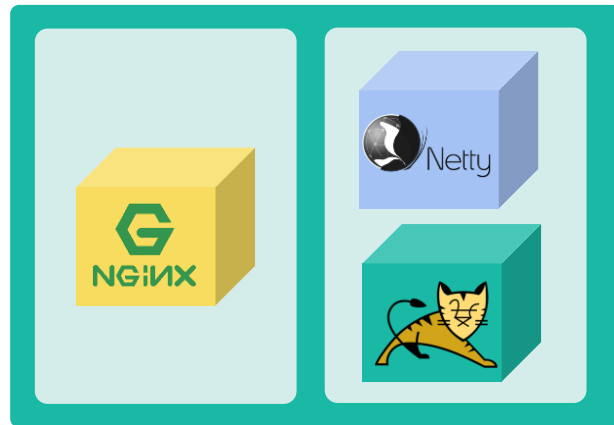
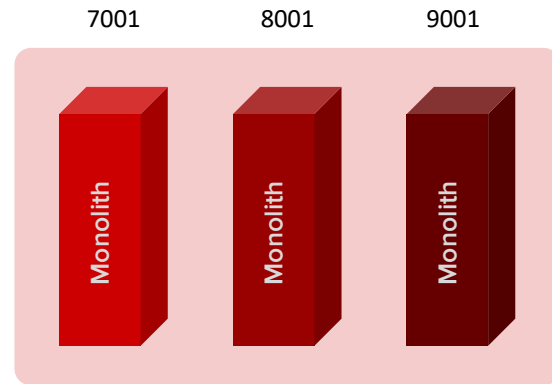
Stateful
Backing
Services

Cloud Native Design - Port binding

Export Services via Port binding

Pivotal

ORACLE[®]
WebLogic

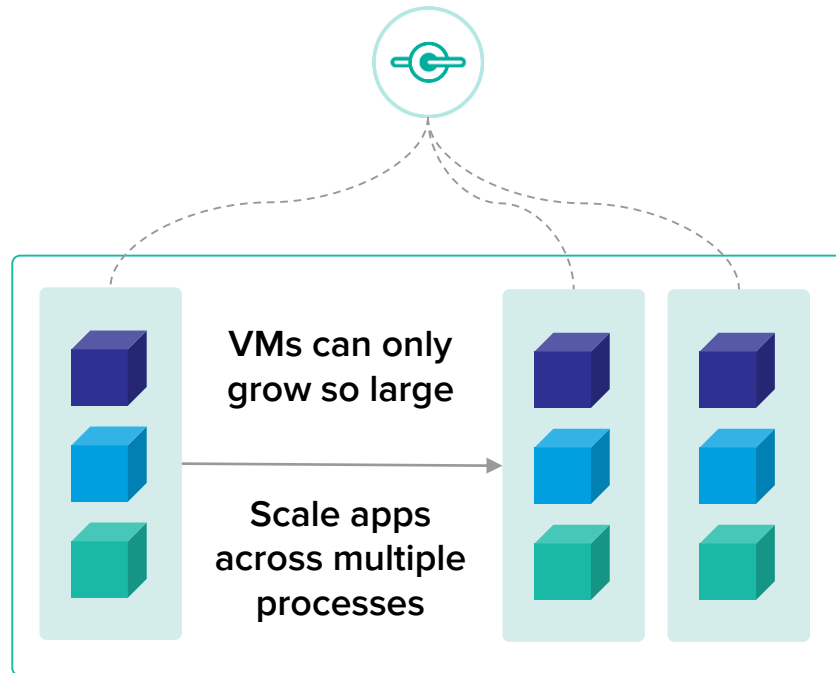


Pivotal
Cloud Foundry[®]

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

Cloud Native Design - Concurrency

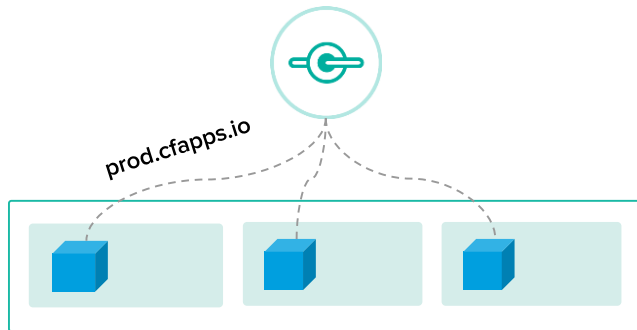
Scale out via the process model



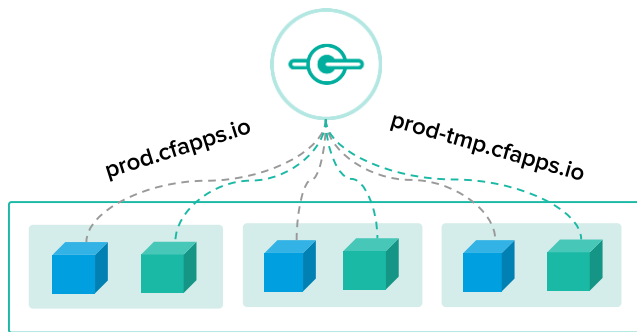
Cloud Native Design - Disposability

Maximum robustness with fast startup and graceful shutdown

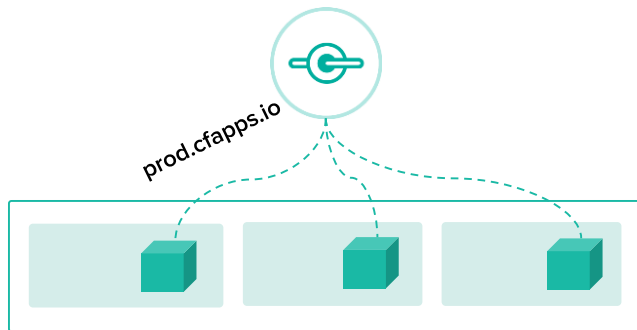
Pivotal



Apps can start or stop at a moment's notice



Strive for fast startup



And graceful shutdown

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.



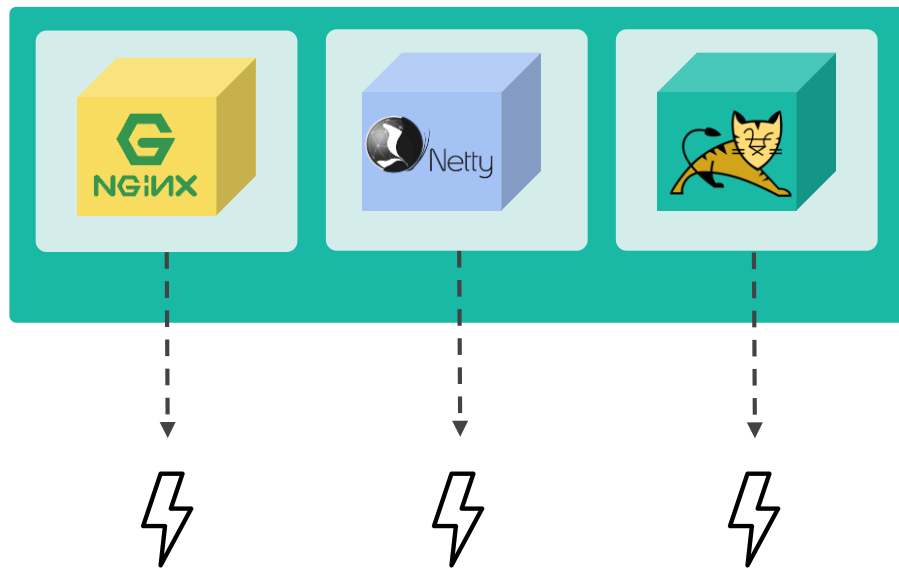
Technology

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

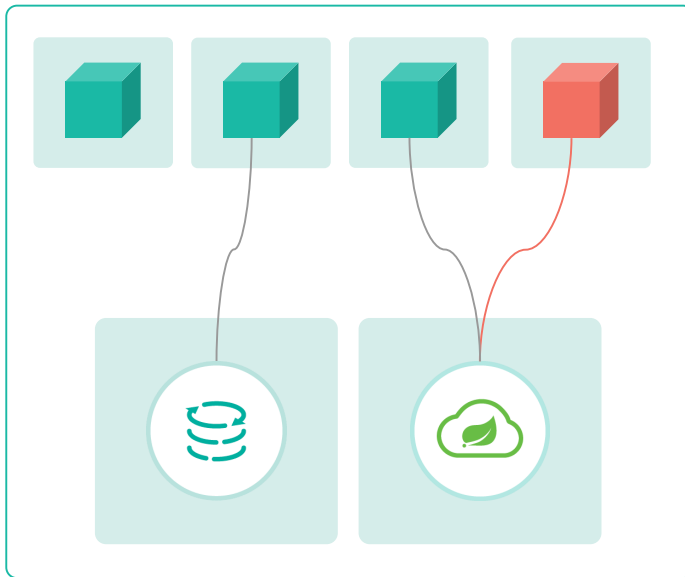
Treat logs as event stream



- 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



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

The background of the slide is a teal-colored image of the Golden Gate Bridge, viewed from a low angle looking up at the tower and the bridge deck stretching into the distance.

Pivotal®



Transforming How The World Builds Software