

Introduction to Platform-as-a-Service and Cloud Foundry

Manuel Silveyra Senior Cloud Solutions Architect **IBM Open Technologies**

silveyra@us.ibm.com

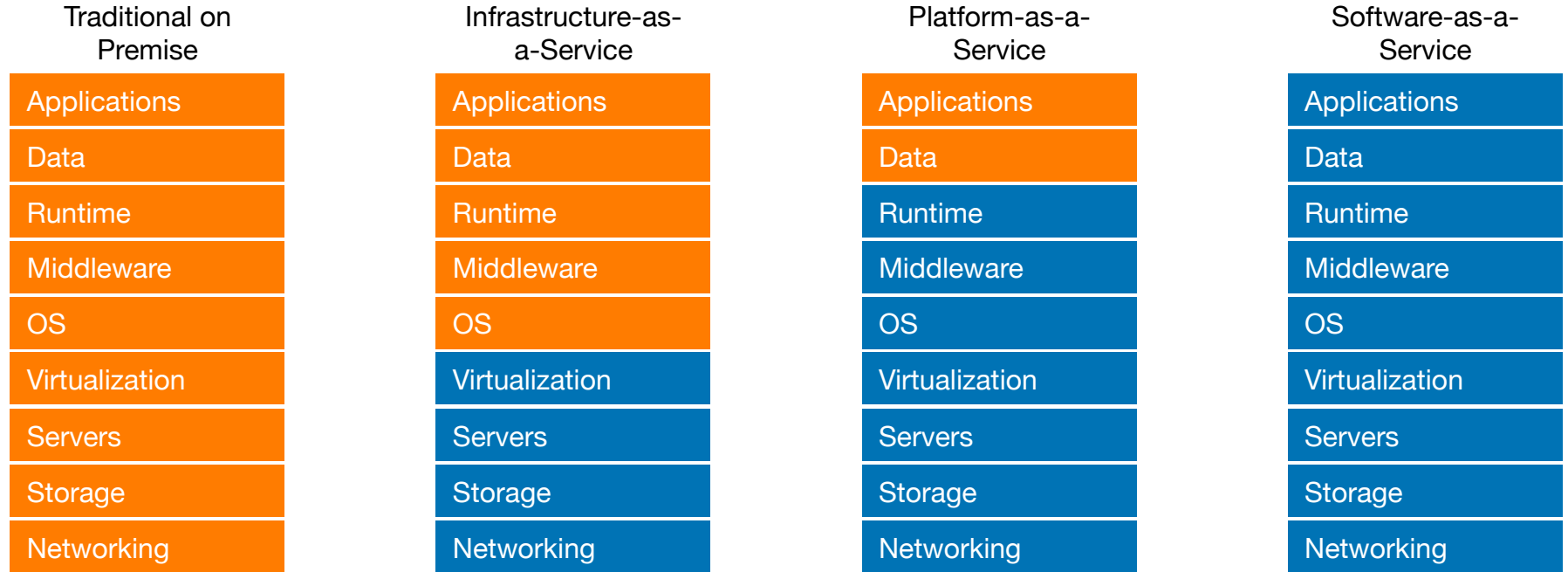
@manuel_silveyra



<https://developer.ibm.com/opentech/>

Agenda

- **What is Platform-as-a-Service?**
- What is Cloud Foundry?
- Sample Cloud Foundry Flow
- The 12 Factor Application and Microservices
- Questions

Cloud Delivery Models



 Client Manages
 Vendor Manages in Cloud

Standardization – Lower Costs –Faster Time to Value

IaaS: The Software Defined Data Center



PaaS: The Cloud Operating Environment



SaaS: The API Economy



The Key Benefits of PaaS for Developers

- There's no need to focus on provisioning, managing, or monitoring the compute, storage, network and software
 - Developers can create working prototypes in a matter of minutes.
 - Developers can create new versions or deploy new code more rapidly
 - Developers can self-assemble services to create integrated applications.
 - Developers can scale applications more elastically by starting more instances.
 - Developers don't have to worry about underlying operating system and middleware security patches.
 - Developers can mitigate backup and recovery strategies, assuming the PaaS takes care of this.

The Key Disadvantages of PaaS for Developers

- Applications require a different architecture mindset
 - This requires developer skill and awareness of best practices and web app limitations.
 - Don't have as much control over the underlying infrastructure. Security, versioning, performance considerations

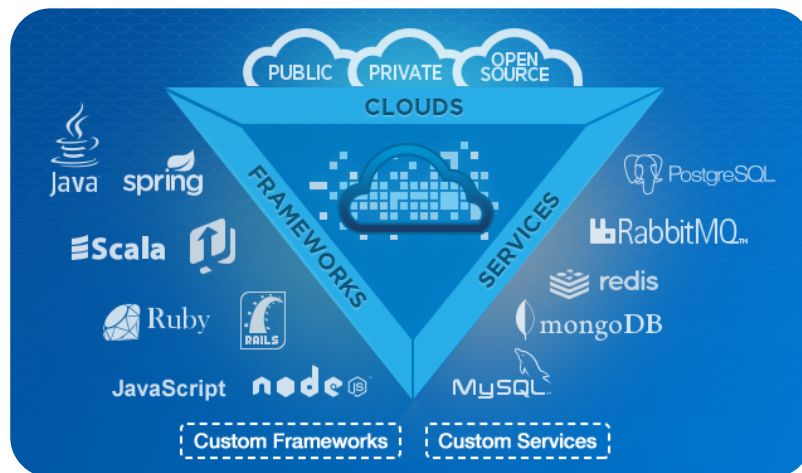
Agenda

- What is Platform-as-a-Service?
- **What is Cloud Foundry?**
- Sample Cloud Foundry flow
- The 12 Factor Application and Microservices
- Questions



Cloud Foundry is the industry's Open PaaS and **provides a choice of clouds, frameworks, and application services**. Its unique vision is to foster contributions from a broad community of developers, users, customers, partners, and ISVs while advancing development of the platform at extreme velocity.

cloudfoundry.org



The Cloud Foundry Foundation

- The Cloud Foundry Foundation's mission is to **establish and sustain** Cloud Foundry as the global industry standard open source PaaS technology with a thriving ecosystem.
- To **deliver continuous quality, value and innovation** to users, operators and providers of Cloud Foundry technology.
- To provide a **vibrant agile experience** for the community's contributors that delivers the highest quality cloud-native applications and software, at high velocity with global scale.
- Its guiding principles are:
 - **Governance By Contribution** - Influence within the Foundation is based on contributions.
 - **IP Hygiene** - IP cleanliness must be preserved at all times.
 - **Equal Opportunity To Participate** - Everyone has an equal opportunity to participate in projects.
 - **No Surprises** - Planning processes and project status are open to all.

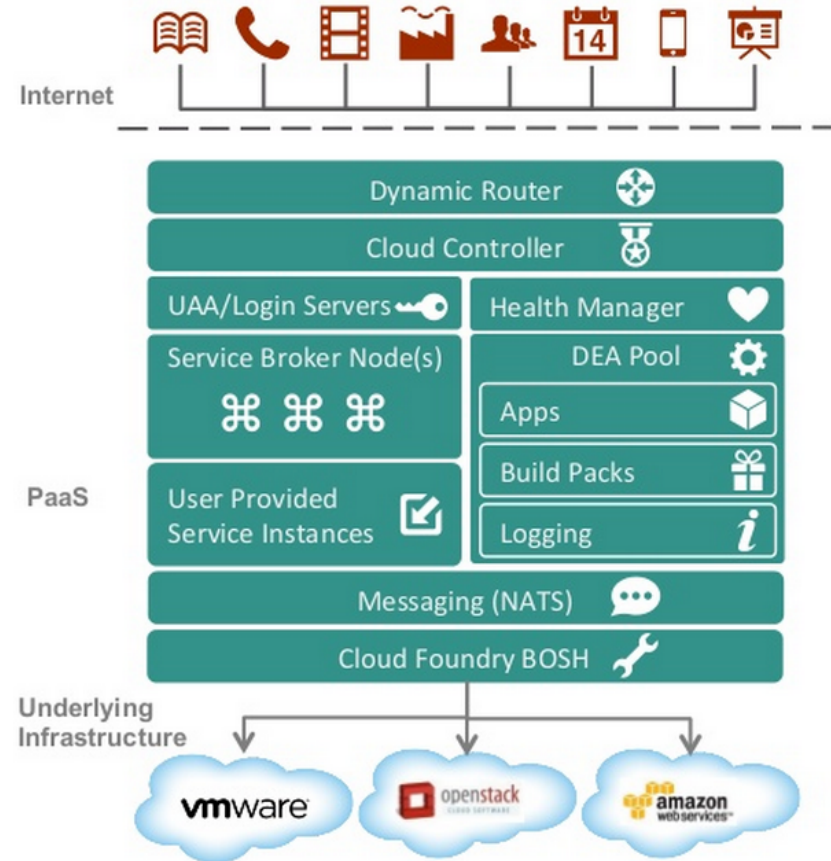
The Cloud Foundry Foundation



<http://www.cloudfoundry.org/about/index.html>

Cloud Foundry Architecture

- The platform is abstracted as a set of large-scale distributed services.
- It uses Cloud Foundry Bosh to operate the underlying infrastructure from the IaaS providers.
- Components are dynamically discoverable and loosely coupled.
- Health is exposed through HTTP endpoints so agents can collect state information and act on it.



Cloud Foundry Components: Dynamic Router

How It Works

- The router shapes and routes all external system traffic (HTTP/API) and application traffic from the internet/intranet.
- It maintains a dynamic routing table for each load-balanced app instance with IP addresses and ports.

Responsible For

- Load balancing
- Maintaining an active routing table
- Access logs
- Supports web-sockets

Cloud Foundry Components: Cloud Controller

How It Works

- The Cloud Controller maintains command and control systems, including interface with clients (CLI, Web UI, Spring STS), account and provisioning control.
- It also provides RESTful interface to domain objects (apps, services, organizations, spaces, service instances, user roles, and more).

Responsible For

- Expected App state, state transitions, and desired convergence
- Permissions/Auth Orgs/Spaces/Users
- Services management
- App placement
- Auditing/Journaling and billing events
- Blob storage

Cloud Foundry Components: UAA and Login Servers

How It Works

- “User Authorization and Authentication” provides identity, security and authorization services.
- It manages third party OAuth 2.0 access credentials and can provide application access and identity-as-a-service for apps running on Cloud Foundry.
- Composed of: UAA Server, Command Line Interface, Library.

Responsible For

- Token Server
- ID Server (User management)
- OAuth Scopes (Groups) and SCIM
- Login Server
 - UAA Database
 - SAML support (for SSO integration) and Active Directory support with the VMware SSO Appliance
- Access auditing

Cloud Foundry Components: Health Manager

How It Works

- Health Manager monitors application uptime by listening to the NATS message bus for mismatched application states (expected vs. actual).
- The Cloud Controller publishes expected state and the DEAs publish actual state.
- State mismatches are reported to the Cloud Controller.

Responsible For

- Maintains the actual state of apps
- Compares to expected state
- Sends suggestions to make actual match expected (cannot make state changes itself – only CC can do that!)

Cloud Foundry Components: DEA

How It Works

- “Droplet Execution Agents” are secure and fully isolated containers.
- DEAs are responsible for an Apps lifecycle: building, starting and stopping Apps as instructed.
- They periodically broadcast messages about their state via the NATS message bus.

Responsible For

- Managing Linux containers (Warden)
- Monitoring resource pools
 - Process
 - File system
 - Network
 - Memory
- Managing app lifecycle
- App log and file streaming
- DEA heartbeats (NATS to CC, HM)

Cloud Foundry Components: Buildpacks

How It Works

- Buildpacks are Ruby scripts that detect application runtimes/frameworks/plugins, compile the source code into executable binaries, and release the app to an assigned DEA.
- Runtime components can be cached for faster execution of subsequent app pushes.

Responsible For

- Staging*
 - /bin/detect
 - /bin/compile
 - /bin/release
- Configure droplet
 - Runtime (Ruby/Java/Node/Python)
 - Container (Tomcat/Liberty/Jetty)
 - Application (.WAR, .rb, .js, .py)

(*) Cloud Foundry Buildpacks are compatible with Heroku

Cloud Foundry Components: Messaging (NATS)

How It Works

- NATS is a fast internal messaging bus to manage system wide communication via a publish-and-subscribe mechanism.

Responsible For

- Non-Persistent messaging
- Pub/Sub
- Queues (app events)
- Directed messages (INBOX)

Cloud Foundry Components: Service Broker

How It Works

- Service Brokers provide an interface for native and external 3rd party services.
- Service processes run on Service Nodes or with external as-a-service providers (e.g., email, database, messaging, etc.).

Responsible For

- Advertising service catalog.
- Makes create/delete/bind/unbind calls to service nodes.
- Requests inventory of existing instances and bindings from cloud controller for caching, orphan management.
- SaaS marketplace gateway.
- Implemented as HTTP endpoint, written in any language.

Cloud Foundry Components: UPSI

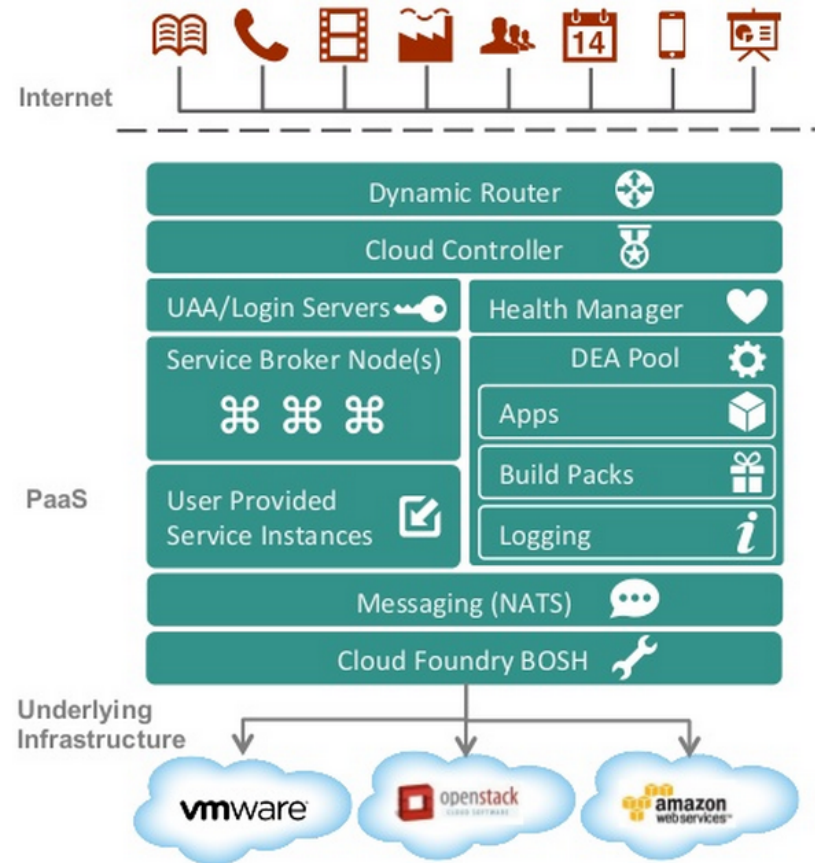
How It Works

- User Provided Service Instances (formerly “Service Connectors”) store meta-data in the Service Broker to enable Cloud Foundry to connect to local services that are NOT managed by Cloud Foundry (e.g., OracleDB, DB2, SQLServer, etc.)

Responsible For

- Metadata management

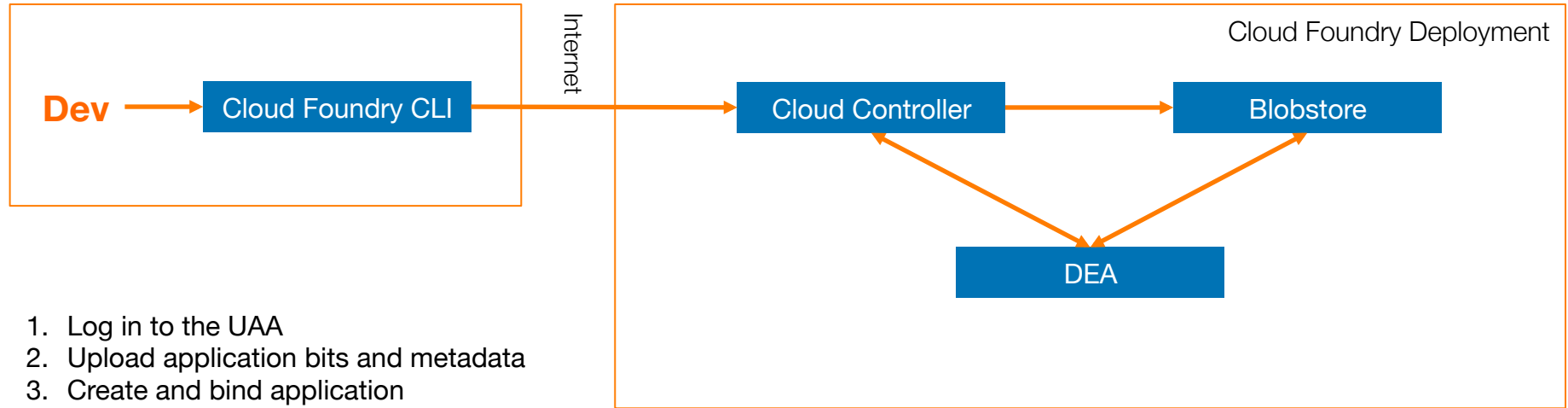
Cloud Foundry Architecture



Agenda

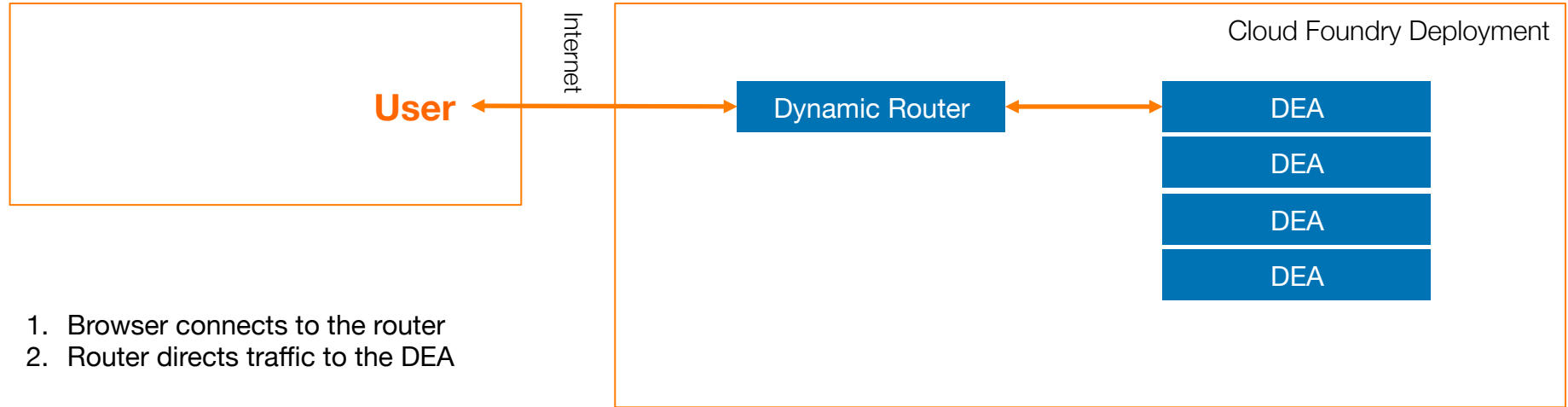
- What is Platform-as-a-Service?
- What is Cloud Foundry?
- **Sample Cloud Foundry Flow**
- The 12 Factor Application and Microservices
- Questions

Pushing an App to Cloud Foundry



1. Log in to the UAA
2. Upload application bits and metadata
3. Create and bind application
4. Stage application
5. Deploy application
6. Manage application health

Accessing an App in Cloud Foundry



Agenda

- What is Platform-as-a-Service?
- What is Cloud Foundry?
- Sample Cloud Foundry Flow
- **The 12 Factor Application and Microservices**
- Questions

Best Practices for Microservice Architectures

12factor.net

The 12 factor app is a methodology for building apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project.
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments.
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration.
- **Minimize divergence** between development and production, enabling continuous deployment for maximum agility.
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The 12 factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc.)

1. **Codebase**
One codebase tracked in revision control, many deploys.
2. **Dependencies**
Explicitly declare and isolate dependencies.
3. **Config**
Store config in the environment.
4. **Backing Services**
Treat backing services as attached resources.
5. **Build, Release, Run**
Strictly separate build and run stages.
6. **Port Binding**
Export services via port binding.
7. **Concurrency**
Scale out via the process model.
8. **Disposability**
Maximize robustness with fat startup and graceful shutdown.
9. **Dev/Prod Parity**
Keep development, staging, and production as similar as possible.
10. **Logs**
Treat logs as event streams.
11. **Admin Processes**
Run admin/management tasks as one-off processes.

Designing Apps for the Cloud: PaaS Best Practices

Following these guidelines makes an application cloud-native, and facilitates deployment to Cloud Foundry and other cloud platforms.

- **Avoid writing to the local file system**
 - Local file system storage is short-lived
Your application can write local files while it is running, the files will disappear after the application restarts.
 - Instances of the same application do not share a local file system
Each application instance runs in its own isolated container. Thus a file written by one instance is not visible to other instances of the same application.
- **HTTP sessions are not persisted or replicated**
Session data that must be available after an application crashes or stops, or that needs to be shared by all instances of an application, should be stored in a Cloud Foundry service.
- **Run multiple instances to increase availability**
To avoid the risk of an application being unavailable during Cloud Foundry upgrade processes, you should run more than one instance of an application.
- **Design as if your application can be restarted, destroyed, started at any time!**

<http://bit.ly/cf-paas-bp>

Agenda

- What is Platform-as-a-Service?
- What is Cloud Foundry?
- Sample Cloud Foundry Flow
- The 12 Factor Application and Microservices
- **Questions**

Getting Started with Cloud Foundry

Basics

- Trial accounts with hosted providers
 - <http://bluemix.net>
 - <http://run.pivotal.io>
- Cloud Foundry documentation
 - <http://docs.cloudfoundry.org>
- Cloud Foundry community
 - <http://cloudfoundry.org>
- Cloud Foundry on GitHub
 - <https://github.com/cloudfoundry>

Advanced

- “Try Cloud Foundry” on AWS
 - <https://trycf.starkandwayne.com>
- BOSH bootstrap
 - <https://github.com/cloudfoundry-community/bosh-bootstrap>
- Deploy your own to AWS
 - <http://www.slideshare.net/SpringCentral/build-yourown-cf>
 - <http://docs.cloudfoundry.org/deploying/ec2/>
- Install on a laptop
 - <https://github.com/cloudfoundry/bosh-lite>
 - https://github.com/yudai/cf_nise_installer
- Stackato Micro Cloud
 - http://www.activestate.com/stackato/get_stackato

Cloud Foundry Versus Other Platforms-as-a-Service

<http://paasify.it>



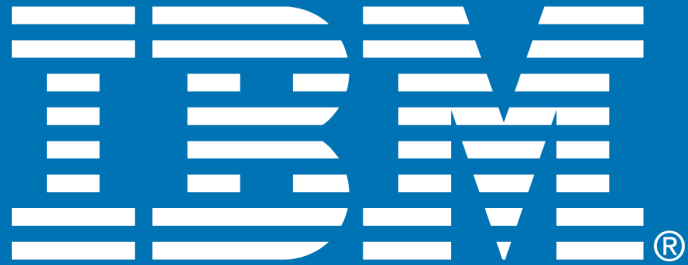
Cloud Foundry



OpenShift Origin

Links

- Cloud Foundry for PHP developers
<http://bit.ly/cf-for-php>
- Cloud Foundry 101 - Platform: The Cloud Foundry Conference
<https://www.youtube.com/watch?v=nOuxMHJIKFU>
- Matt Stine - Cloud Foundry and Microservices: A Mutualistic Symbiotic Relationship (CF Summit 2014)
<https://www.youtube.com/watch?v=RGZefc92tZs>
- Cloud Foundry Technical Overview
<http://www.slideshare.net/cdavisafc/cloud-foundry-technical-overview>
- A Brief History Of Cloud Foundry and Stackato
<http://www.activestate.com/blog/2014/03/brief-history-cloud-foundry-and-stackato>
- PaaS Comparison: Cloud Foundry, Microsoft Azure, Google App Engine, Amazon, Heroku, and OpenShift
<http://blog.pivotal.io/cloud-foundry-pivotal/features/paas-comparison-cloud-foundry-microsoft-azure-google-app-engine-amazon-heroku-and-openshift>
- CF Summit Sessions: “PaaS Comparison 2014”
<http://blog.altoros.com/cf-summit-2014-paas-comparison.html>



Open *by design*™