Pivotal

The Cloud Native Platform

MS .NET and Pivotal Cloud Foundry

Platform Architecture Team

Cloud Native Platform
Pivotal Cloud Foundry

MS .NET and Pivotal Cloud Foundry

Here is my .NET code

Run it in the cloud for me

I do not care how

Deploying .NET apps? It doesn't have to be hard

Traditional .NET deployment on VMs	Pivotal Cloud Foundry

The Big Picture

CLOUD NATIVE, CLOUD NATIVE PLATFORM, CLOUD NATIVE RUNTIME, CLOUD FOUNDRY, APPLICATION FRAMEWORKS

Cloud Native

- An approach to building and running applications that exploits the cloud computing model
  - Infrastructure independence
  - Automated provisioning & configuration
  - Automated scaling
  - Rapid recovery
  - Continuous delivery
  - Microservices architecture
  - Embracing DevOps
  - Automated & Proactive Security
  - Modern Application Frameworks

- Business Benefits
  - Business Agility
  - Competitive Advantage
  - Reduced IT Infrastructure Costs
  - IT Focused on Business Priorities

Cloud Native is not just about **where** you run your app, it's about **how** you build and deploy your app!

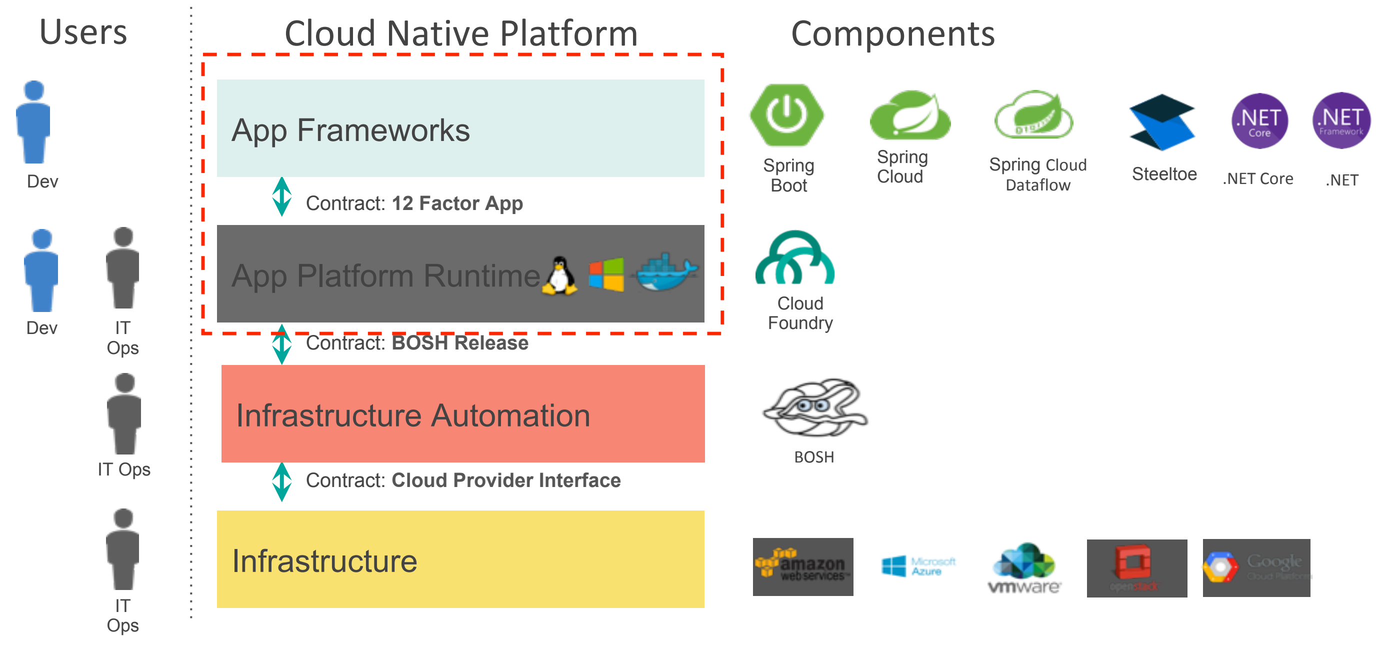
Cloud Native Applications

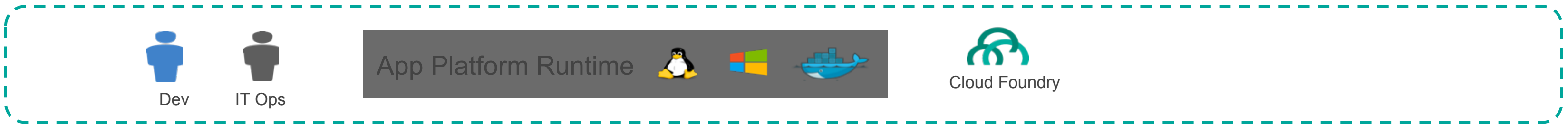
- Microservices Architecture
- Modern Application Frameworks
- Containers & Orchestration
- Twelve-Factor Principles

- Infrastructure Independence
- Continuous Delivery
- Shift from Silo IT to DevOps
- Automated & Proactive Security

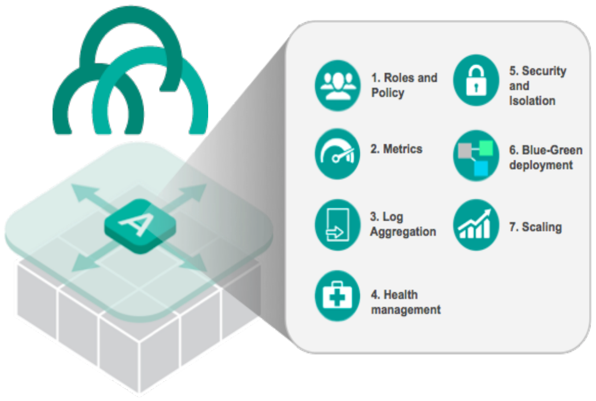
Cloud Native Platform

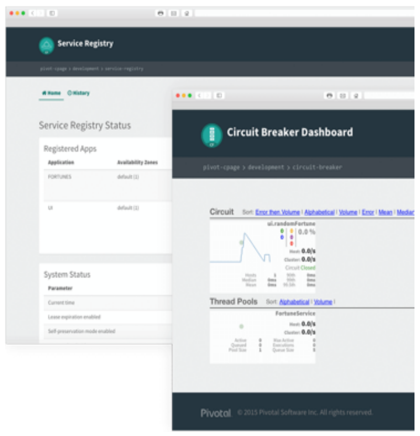
Cloud Native Platform



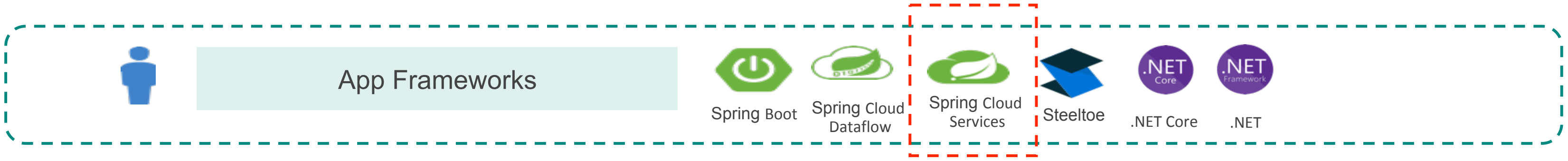


Cloud Foundry
Everything Needed to Deploy and Operate Cloud Native Applications





- Containers & Orchestration
- Continuous Delivery
- Facilitates DevOps
- Twelve-Factor Principles

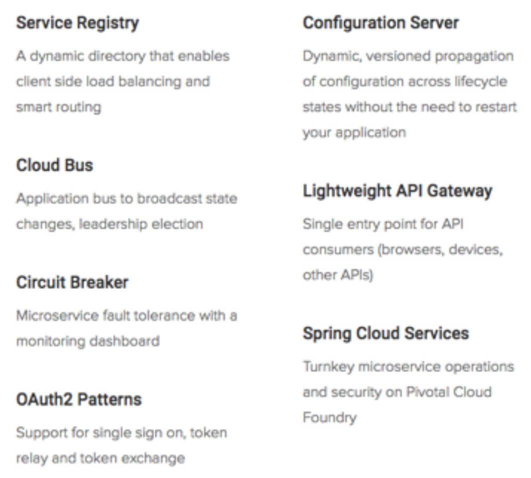


Facilitates Twelve-Factor Contract

Spring Cloud Services
Powered by Netflix OSS

- Spring Cloud Services
  - Which is built on Spring Boot simplifies distributed, microservice-style architecture by implementing proven patterns to bring resilience, reliability, and coordination to your microservices.
  - When used with PCF, customers have a turnkey, secure solution for production operations of this coordination infrastructure—service registry, config server, and circuit breaker dashboard.

Enabling Cloud Native Applications



Service Registry

A dynamic directory that enables client side load balancing and smart routing

Cloud Bus

Application bus to broadcast state changes, leadership election

Circuit Breaker

Microservice fault tolerance with a monitoring dashboard

OAuth2 Patterns

Support for single sign on, token relay and token exchange

Configuration Server

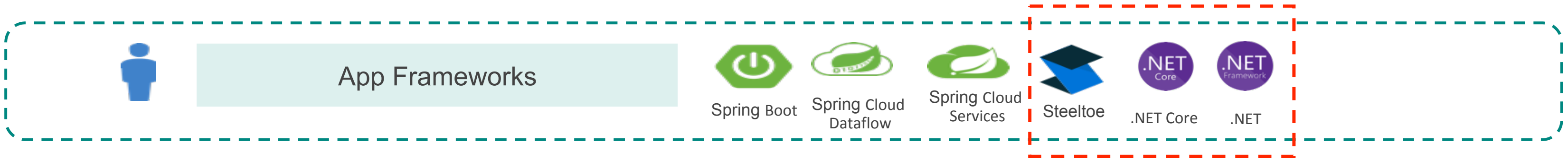
Dynamic, versioned propagation of configuration across lifecycle states without the need to restart your application

Lightweight API Gateway

Single entry point for API consumers (browsers, devices, other APIs)

Spring Cloud Services

Turnkey microservice operations and security on Pivotal Cloud Foundry



Facilitates Twelve-Factor Contract on .NET

Steeltoe

NETFLIX OSS

.NET Core

.NET Framework

Enabling Cloud Native Applications on .NET

- Simplifies using .NET & ASP.NET on Cloud Foundry
  - Connectors (e.g. MySql, Redis, Postgres, RabbitMQ, OAuth, etc.)
  - Security providers (e.g. OAuth SSO, JWT, Redis KeyRing Storage, etc.)
  - Configuration providers (e.g. Cloud Foundry)
  - Management & Monitoring
- Simplifies using Spring Cloud Services
  - Configuration (e.g. Config Server, etc.)
  - Service Discovery (e.g. Netflix Eureka, etc.)
  - Circuit Breaker (e.g. Netflix Hystrix)
  - Distributed Tracing (e.g. Slueth coming)

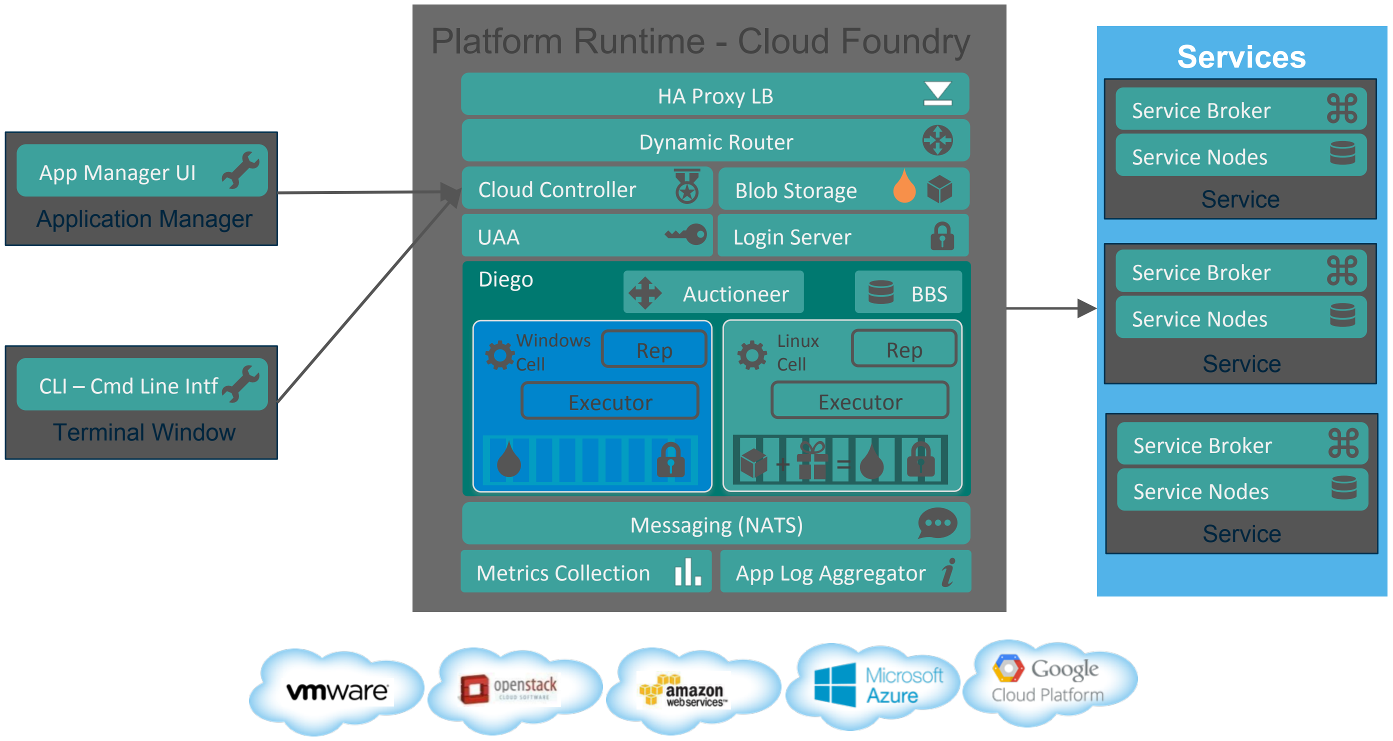
maturity model

Cloud Native	• Microservice architecture • API-first design
Cloud Resilient	• Designed for failure • Apps unaffected by dependencies • Proactive failure testing • Metrics and monitoring baked in • Cloud agnostic runtime
Cloud Friendly	• Twelve factor app • Horizontal scalable • Leverage platform for HA
Cloud Ready	• No file system dependency • Self contained application • Platform managed ports/address • Consume off-platform services

Pivotal Cloud Foundry Fundamentals

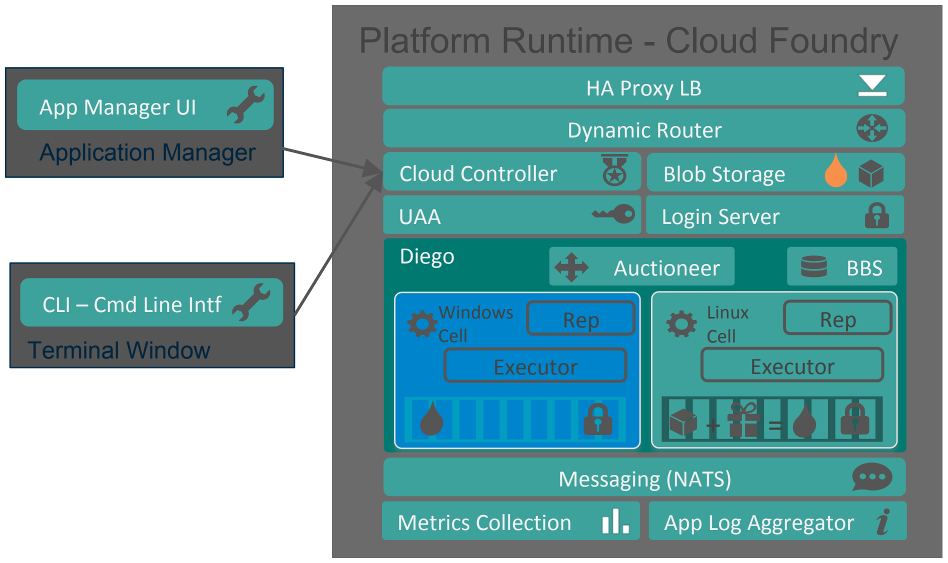
ORGS, SPACES, USERS, ROLES, CLI, API, APPS MANAGER

Pivotal Cloud Foundry (PCF) Architecture



Cloud Controller API

- Cloud Controller (CC) component of Cloud Foundry manages all APIs
- CF CLI and other clients like Apps Manager directly call this API
- Before accessing the CC API, you must get an access token from the User Account and Authentication (UAA) server
- http://apidocs.cloudfoundry.org



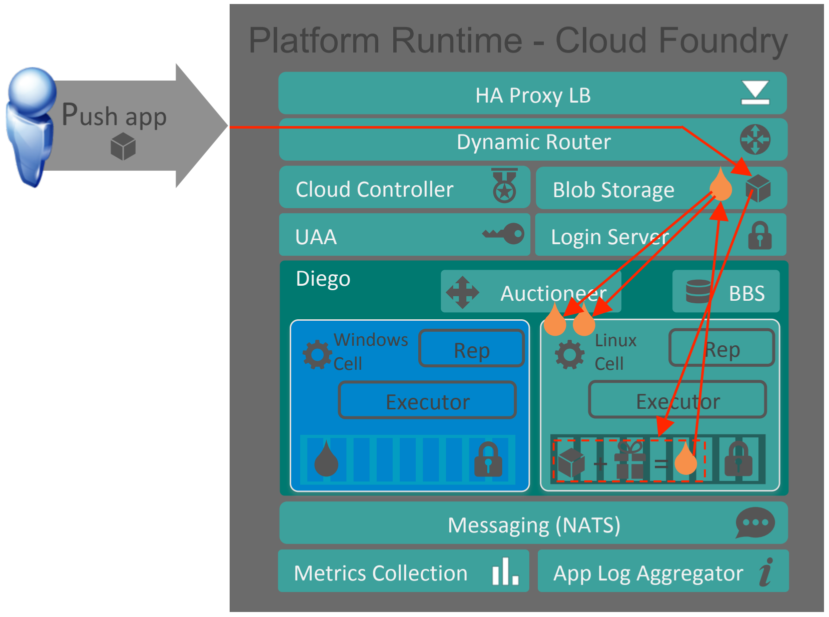
Running .NET Applications on Pivotal Cloud Foundry

CF PUSH, MANIFEST, STAGING, BUILD PACKS, CONTAINERS, CELLS, ENVIRONMENT VARIABLES, VCAP_APPLICATION

Running an App on Pivotal Cloud Foundry (PCF)

1. Upload app bits using metadata from manifest to blob store
2. Bind services – *covered in next section*
3. Stage application & apply buildpack
4. Save staged application image (i.e. droplet 🔥)
5. Deploy & run image in container
6. Manage applications health

```
cf push <appname> -p <path to bits>
cf push <appname> -f <manifest> -p <path to bits>
```



Manifest Files – Application Metadata

- Application manifests tell cf push what to do with applications
- What OS stack
  - Linux default – cflinuxfs2
  - Windows – windows2012R2
- How many instances to create and how much memory to use
- Helps automate deployment, including multiple apps at once
- Can list services to be bound to the application
- Command line to use to start the application

```
---
applications:
- name: fortuneui
  random-route: true
  buildpack: binary_buildpack
  memory: 512M
  stack: windows2012R2
  command: cmd /c .\Fortune-Teller-UI --server.urls http://*:%PORT%
  env:
    ASPNETCORE_ENVIRONMENT: Production
  services:
   - myConfigServer
   - myDiscoveryService
   - myRedisService
   - myHystrixService
   - myOAuthService
```

```
---
applications:
- name: fortuneService
  random-route: true
  buildpack: dotnet_core_buildpack
  env:
    ASPNETCORE_ENVIRONMENT: Production
  services:
   - myConfigServer
   - myDiscoveryService
   - myMySqlService
   - myOAuthService
```

Staging an Application – Applying Buildpack

- Build container images (i.e. droplets)
  - Saved in Blobstore
- Takes care of
  - Detecting which type of application is being pushed
  - Installing the appropriate run-time if needed
  - Installing required dependencies or other artifacts if needed
  - Creating the command used to start the application
- Lots of Build packs
  - Staticfile, Java, Ruby, Nodejs, Go, Python, PHP, .NET Core, Binary, HWC
- Configurable on Cloud Foundry
  - cf buildpacks, cf create-buildpack, cf update-buildpack, cf delete-buildpack, etc.

Why Buildpacks

- Control what frameworks/runtimes are used on the platform
- Provides consistent deployments across environments
  - Stops deployments from piling up at operation's doorstep
  - Enables a self-service platform
- Eases ongoing operations burdens
  - Security vulnerability is identified
  - Subsequently fixed with a new buildpack release
  - Restage applications
- Three buildpacks used for staging .NET applications
  - .NET Core – `dotnet_core_buildpack`
  - Binary - `binary_buildpack`
  - .NET HWC - `hwc_buildpack`

.NET Core Buildpack

- Used to create container images ready to run .NET Core applications on Linux cells
  - Target Stack: cflinuxfs2
  - Supports pushing two types of directories
    - Source – project source code
    - Binaries – `dotnet publish` output

```
---
applications:
- name: env
  random-route: true
  memory: 1G
  stack: cflinuxfs2
  buildpack: dotnet_core_buildpack
```

- Pushing source
  - Must contain `.csproj`
  - Installs .NET Core runtime – version specify via global.json, else build pack chooses
  - Restores application dependencies
- Pushing binaries – two types of binary directories supported
  - Must NOT contain `.csproj`
  - FDD directory - Portable .NET Core application
    - Installs .NET Core runtime – version specified via global.json, otherwise build pack chooses
  - SCD directory - Self-contained .NET Core application
    - Installs Linux dependencies (i.e. libunwind.so)

Binary Buildpack

- Used to create container images ready to run .NET Core applications on Windows cells
  - Target Stack: Windows2012R2
  - Supports pushing
    - Binaries – `dotnet publish` output
  - Copies image, as is, no additional dependencies added
  - Provide shell command to be used to start the application

```
---
applications:
- name: fortuneService
  random-route: true
  buildpack: binary_buildpack
  memory: 512M
  stack: windows2012R2
  command: cmd /c .\Fortune-Teller-Service --server.urls http://*:%PORT%
```

- Pushing binary – two types of binary directories supported
  - FDD directory - Portable .NET Core application (Normally not used!)
    - Would require the .NET Core runtime to have been installed on the Windows machine
  - SCD directory - Self-contained .NET Core application

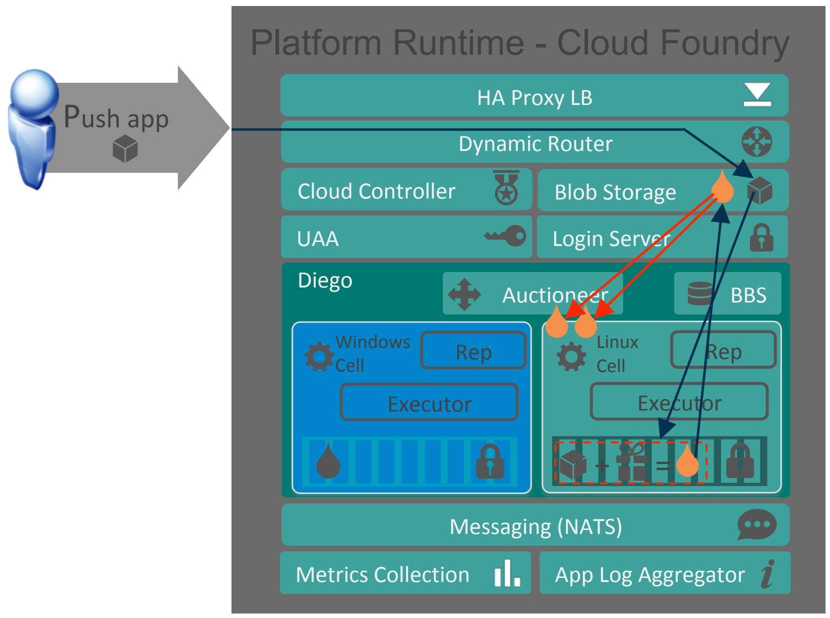
.NET Hosted Web Core (HWC) Buildpack

- Used to create container images ready to run ASP.NET/IIS applications on Windows cells
  - Target Stack: Windows
  - Buildpack ensures web.config is present
  - Installs `hwc.exe` in droplet and configures application to be launched under it
    - Runs the application as a "Windows Hosted Web Core" application
      - `hwc.exe` builds needed configuration files, and `Activates` application

```
---
applications:
- name: env
  random-route: true
  memory: 1G
  stack: windows2012R2
  buildpack: hwc_buildpack
```

Deploying Image to Containers in Cells

- Diego Container management handles deployment
  - Auction process determines what cells are selected
    - Stack requirements (i.e. Windows/Linux)
    - Cells load and resources
    - Availability requirements
  - Containers are created in each cell
    - Logs streamed to log system
  - Droplet image downloaded to cell and started in container
  - When instance health is good, Router notified of instance

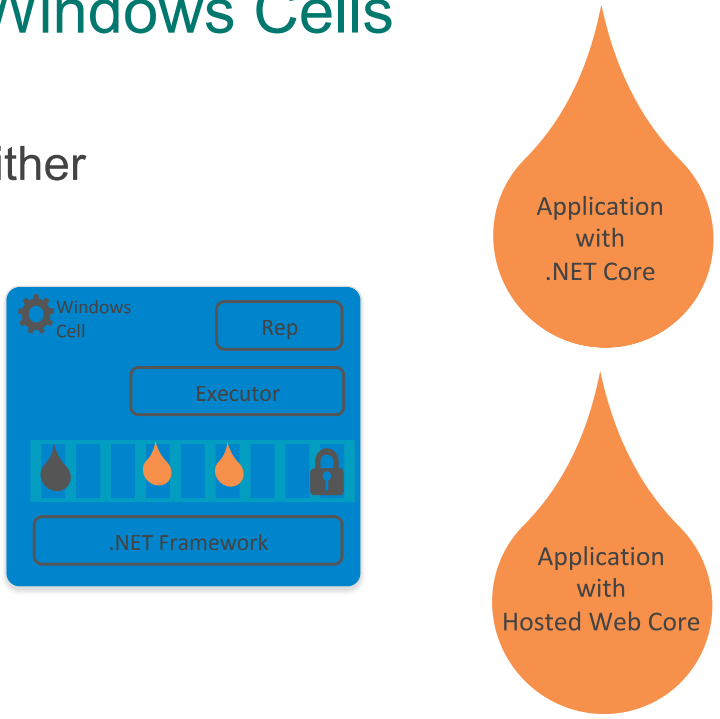


Why Containers

- Containers are OS level virtualization (i.e. process isolation)
- Small and allow for much higher packing density; typical container image is 10s of MB
- Easy to move around and to replicate
- Do not have any redundant or unnecessary operating system elements; they don't need the care and feeding of a large OS stack
- Have fast startup times; containers start in milliseconds
- Well suited for building hyper-scale, highly resilient infrastructure

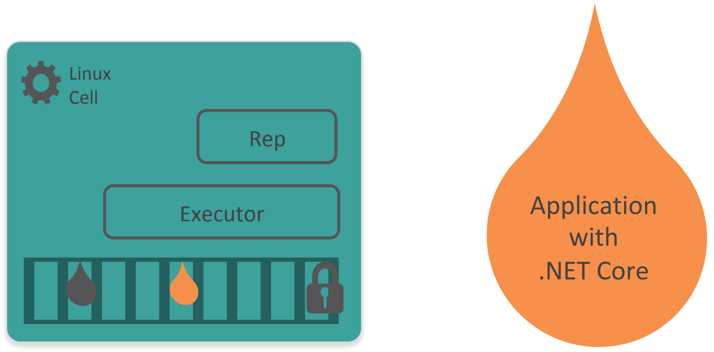
.NET Application Images on Windows Cells

- Run container images prepared by either
  - Binary buildpack
  - HWC buildpack
- Application types supported
  - .NET Full “Background processes”
    - Command line/Console apps
  - ASP.NET 4 applications
    - MVC, WebForm, WebAPI, WCF
  - .NET Core “Background processes”
    - Command line apps/Console apps
  - ASP.NET Core web apps
    - .NET Full Framework
    - .NET Core



.NET Application Images on Linux Cells

- Run container images prepared by
  - .NET Core buildpack
- Application types supported
  - .NET Core “Background processes”
    - Command line apps/Console apps
  - ASP.NET Core web apps
    - .NET Core only



Cloud Foundry Container Environment Variables

- Used to communicate application environment & configuration to deployed containers
  - VCAP_APPLICATION
    - Application attributes – version, instance index, limits, URLs, etc.
  - VCAP_SERVICES
    - Bound services – name, label, credentials, etc.
  - CF_INSTANCE_*
    - CF_INSTANCE_ADDR, CF_INSTANCE_INDEX, etc.

VCAP_APPLICATION Example

```
"VCAP_APPLICATION": {
   "application_id": "95bb5b8e-3d35-4753-86ee-2d9d505aec7c",
   "application_name": "fortuneService",
   "application_uris": [
      "fortuneservice-glottologic-neigh.apps.testcloud.com"
   ],
   "application_version": "40933f4c-75c5-4c61-b369-018febb0a347",
   "cf_api": "https://api.system.testcloud.com",
   "limits": {
     "disk": 1024,
     "fds": 16384,
     "mem": 512
    },
   "name": "fortuneService",
   "space_id": "86111584-e059-4eb0-b2e6-c89aa260453c",
   "space_name": "test",
   "uris": [
    "fortuneservice-glottologic-neigh.apps.testcloud.com"
   ],
   "users": null,
   "version": "40933f4c-75c5-4c61-b369-018febb0a347"
 }
```

Using Services on Pivotal Cloud Foundry

MANAGED, USER-PROVIDED, SERVICE BROKERS, INSTANCE CREATION, APPLICATION BINDING, ENVIRONMENT VARIABLES, VCAP_APPLICATION

What is a Service?

- Allows resources to be easily provisioned on-demand
- Typically an external “component” necessary for applications
  - Database, cache, message queue, microservice, etc.
- Can be a persistent, stateful layer

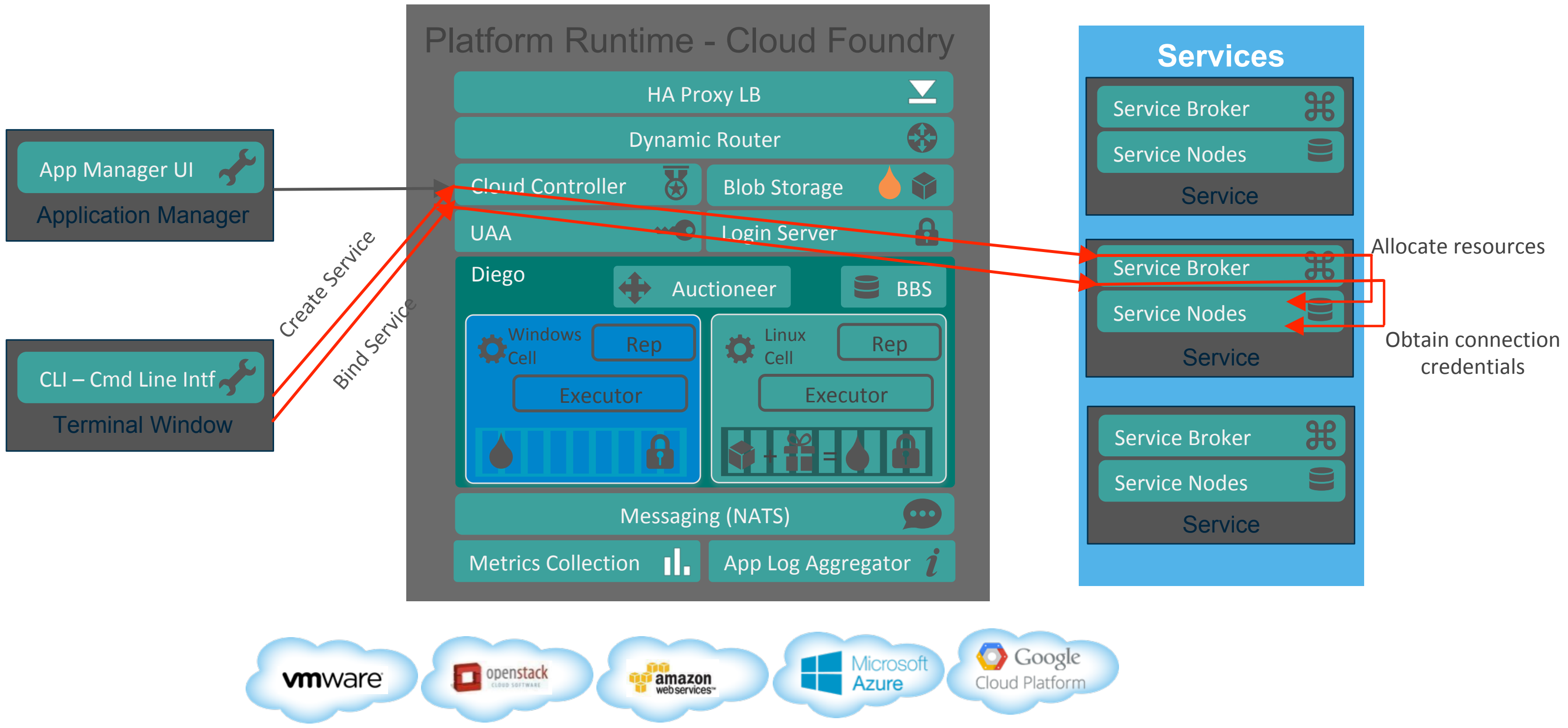
Types of Services

- Managed - Fully integrated, with fully lifecycle management
  - Part of the market place of services
- User-Provided – Created and managed external to the platform

Managed Services

- Integrated with Cloud Foundry
  - Services integrated with Cloud Foundry must implement the Service Broker API
  - Cloud Controller (CC) manages services using the API
- Service Brokers implement the API
  - Advertise a catalog of service offerings and service plans
  - Create service instances
  - Bind applications to service instances
  - Unbind applications from service instances
  - Delete service instances

Creating and Binding Services



User Provided Services

- Service instances managed outside of Cloud Foundry
- Behave like other service instances once created
- Familiar CLI commands provide service instance configuration
  - cf create-user-provided-service ….

EXAMPLE: AN ORACLE DATABASE MANAGED OUTSIDE OF CLOUD FOUNDRY

Cloud Foundry Container Environment Variables

- Used to communicate application environment & configuration to deployed containers
  - VCAP_APPLICATION
    - Application attributes – version, instance index, limits, URLs, etc.
  - VCAP_SERVICES
    - Bound services – name, label, credentials, etc.
  - CF_INSTANCE_*
    - CF_INSTANCE_ADDR, CF_INSTANCE_INDEX, etc.

VCAP_SERVICES Example

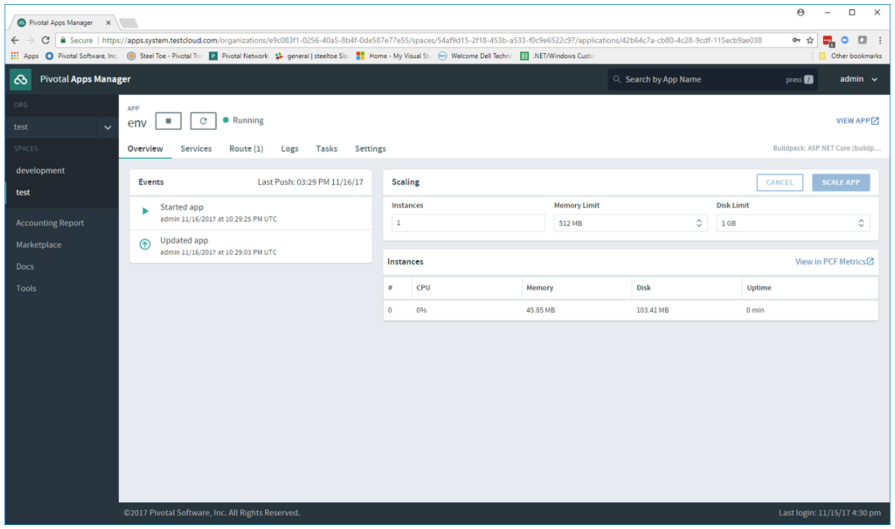
```
"VCAP_SERVICES": {
   "p-config-server": [
    {
     "credentials": {
      "uri": "https://config-bd112dd4-9870-4819-b9a6-62eb3311e27b.apps.testcloud.com",
      "client_secret": "X3e2gKs5Oqhp",
      "client_id": "p-config-server-5f0d1211-75f1-4105-9f94-7ec010de2d3a",
      "access_token_uri": "https://p-spring-cloud-services.uaa.system.testcloud.com/oauth/token"
     },
     "syslog_drain_url": null,
     "volume_mounts": [],
     "label": "p-config-server",
     "provider": null,
     "plan": "standard",
     "name": "myConfigServer",
     "tags": [
      "configuration",
      "spring-cloud"
     ]
    }
   ],
   "p-service-registry": [
………..
```

Scaling and Operating Applications on Cloud Foundry

DOMAINS, DNS, ROUTES, SCALING VIA CLI, SCALING VIA APP MANAGER

Scaling

- Can be done via CLI
  - At deployment time (manifest.yml or as a modifier to cf push)
  - During run time without interrupting operations (cf scale --i 10)
- Can also be done via Apps Manager
- Container image started on other available cells



Secure | https://apps.system.testcloud.com/organizations/e9c083f1-0256-40a5-8b4f-0de587e77e55/spaces/54af9d15-2f18-453b-a533-f0c9e6522c97/applications/42b64c7a-cb80-4c28-9cdf-115ecb9ae038

ORG: test

SPACES: test

APP: env — Running

Buildpack: ASP.NET Core (buildp...

Events — Last Push: 03:29 PM 11/16/17

Started app
admin 11/16/2017 at 10:29:25 PM UTC

Updated app
admin 11/16/2017 at 10:29:03 PM UTC

Scaling

Instances	Memory Limit	Disk Limit
1	512 MB	1 GB

Instances

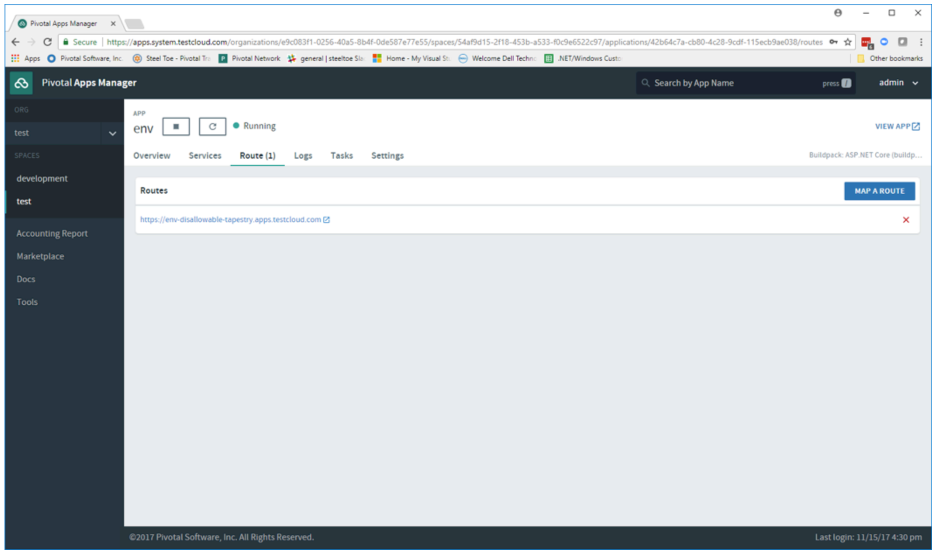
#	CPU	Memory	Disk	Uptime
0	0%	45.85 MB	103.41 MB	0 min

©2017 Pivotal Software, Inc. All Rights Reserved.

Last login: 11/15/17 4:30 pm

Cloud Foundry Routes

- HTTP requests are routed to apps by associating a Route with the application
  - Route = Hostname + Domain
- Many app instances can be mapped to a single Route resulting in load balanced requests
- Routes belong to a space
- Application can have multiple routes


https://apps.system.testcloud.com/organizations/e9c083f1-0256-40a5-8b4f-0de587e77e55/spaces/54af9d15-2f18-453b-a533-f0c9e6522c97/applications/42b64c7a-cb80-4c28-9cdf-115ecb9ae038/routes

APP env Running

Buildpack: ASP.NET Core (buildp...

Routes

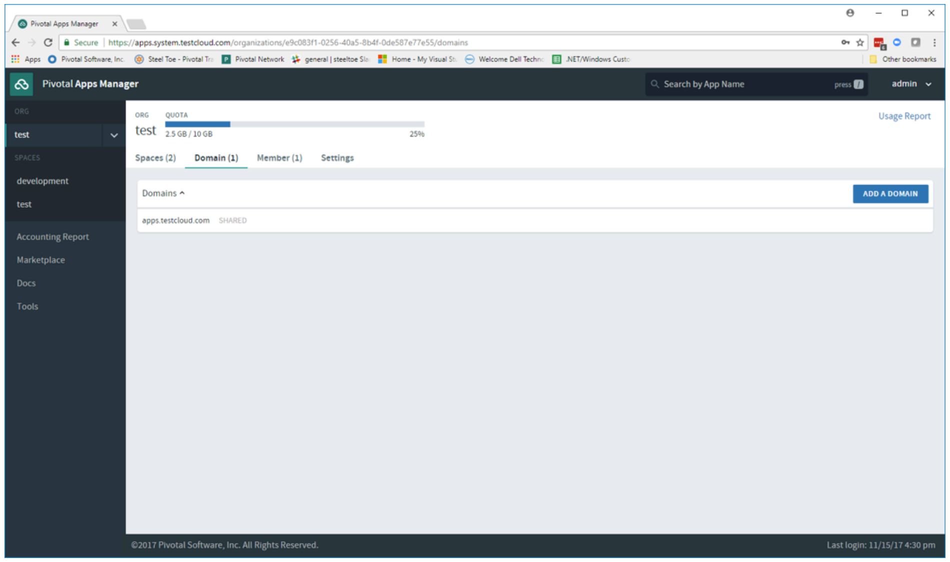
https://env-disallowable-tapestry.apps.testcloud.com

©2017 Pivotal Software, Inc. All Rights Reserved.

Last login: 11/15/17 4:30 pm

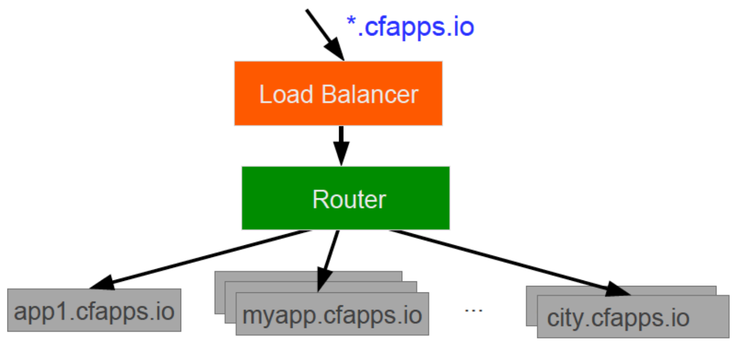
Cloud Foundry Domains

- Each Cloud Foundry installation has a default app domain
- Domains provide a namespace from which to create routes
- Requests for any routes created using the domain will be routed to Cloud Foundry applications
- Domains can be shared or private with regards to PCF organizations

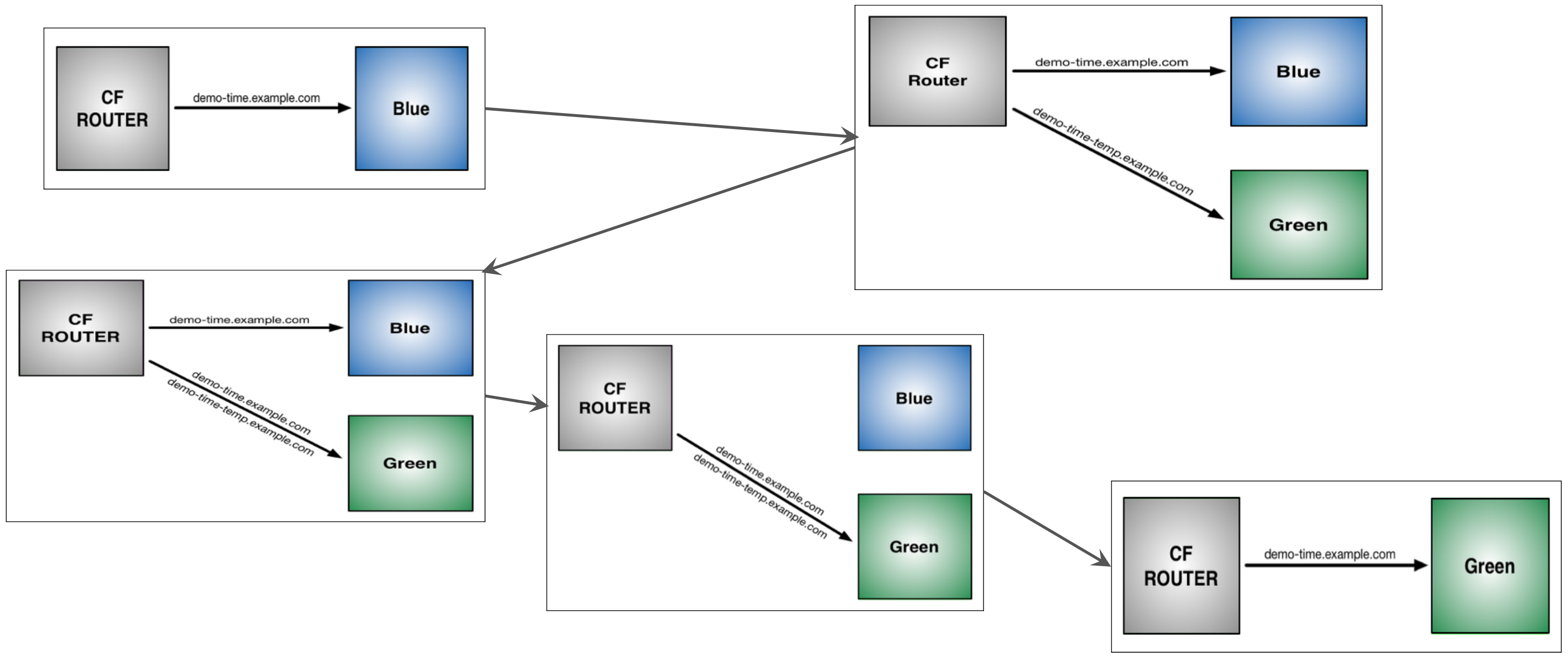


Domains – Behind the Scenes

- A wildcard entry (*) is added to the DNS for the app domain
- That DNS entry points to a load balancer (or Cloud Foundry’s HA Proxy), which points to the Cloud Foundry Router
- The Router uses the subdomain to map to application instance(s)



Blue-Green Deployments



https://docs.pivotal.io/pivotalcf/1-8/devguide/deploy-apps/blue-green.html