Course Guide

IBM Cloud Garage - Cloud Native Developer Bootcamp for IBM Cloud Private

Course code CK105 ERC 1.0

IBM Training

IBM Training

CK105 IBM Cloud Private Cloud Native Developer Bootcamp

Course Introduction

IBM Training

Welcome!

- Instructor Introduction
- In a few minutes, you will be asked to quickly introduce yourself
  - What's your name?
  - What's your role?
  - Experience with IBM Cloud or Cloud Development?
  - Why are you here?

IBM Training

Course Overview

This course:

- Teaches the key features and architecture of cloud–native applications.
- Introduces concepts of Kubernetes and IBM Cloud Private
- Identifies the elements of twelve–factor applications and the characteristics of a resilient, scalable cloud application design.
- Explains the role of DevOps in the cloud–native application lifecycle.

IBM Training

Course Prerequisites

- Experience with enterprise application development in Java™ or other languages
- Familiarity with agile development practices

IBM Training

Course Objectives

- Understand the key features, practices, and architecture of cloud–native application development.
- Learn how to use IBM Cloud Private to develop and deploy cloud-native application.
- Explore and practice running DevOps pipeline for deploying Microservices application in IBM Cloud Private.
- Evaluate the use of Istio service mesh for managing a Microservices application.

IBM Training

Agenda

- Day 1 – Introduction to IBM Cloud
  - Introduction to Cloud Native Development
  - The twelve factor application
  - IBM Cloud Private architecture
  - IBM Cloud Full Stack Development
  - Lab 1 – Environment Preparation
- Day 2 – Container and Kubernetes
  - Introduction to Docker Container
  - Lab 2 – Docker introduction
  - Introduction to Kubernetes
  - Introduction to Helm
  - Building and Deploying application in Kubernetes
  - Lab 3 - Deploy Java application
- Day 3 – DevOps
  - DevOps Concepts
  - Jenkins in IBM Cloud Private
  - ICP Monitoring
  - Lab 4 – Setting up Jenkins
- Day 4 – Microservices architecture
  - Microservices architecture
  - Cloud Data Services
  - Microservices Development
  - Service Mesh
  - Lab 5 – Microservices Design
  - Lab 6 – Deploying BlueCompute
- Day 5 – Managing Microservices with Istio
  - Managing Traffic and Telemetry
  - Microservices Network Security
  - Lab 7 – Working with Istio

IBM Training

Introductions

- Name
- Company
- Where you live
- Your job role
- Your current experience with the products and technologies in this course
- Do you meet the course prerequisites?
- What you expect from this class

IBM Training

Class Logistics

- Course environment
  - You will use a browser to access a virtual desktop image running on IBM Remote Lab Platform (Skytap)
  - This image has a number of software packages pre-installed, typical of a developers desktop
  - Once you log into the virtual desktop image, you will log into IBM Cloud from the virtual desktop image
  - The virtual desktop image provides a consistent starting point and avoids problems in installing software directly on student machines
- Start and end times
- Lab exercise procedures
- Materials in your student packet
- Topics not on the agenda
- Evaluations
- Breaks and lunch
- Outside business
- Food
- Restrooms
- Fire exits
- Local amenities

IBM Training

Introduction to Cloud-Native Application Development

IBM Training

Unit objectives

- Differentiate IBM Cloud developer roles
- Explain cloud native application characteristics

IBM Training

Lesson 1
Developer roles in IBM Cloud

IBM Training

IBM Cloud developer personas

IBM Cloud developer personas can be thought of in these categories:

- Public cloud developer
- Enterprise cloud developer
- Probably others

IBM Training

Public cloud developer

- Simple applications
  - Greenfield development, with no existing apps to migrate
  - Runtime mainly mediates between GUIs and services
  - Little business logic
- Hosted entirely in public
  - No private hosting, neither dedicated or local
- Uses only public resources
  - Twitter feeds, Watson services, and others
  - No existing on-premises systems of record (SoRs) to integrate with
- Values variety of run times and services
  - Node.js, Go, Swift, and others
  - Data and Analytics, Cognitive (Watson services), Mobile services, Internet of Things services
  - Twillio, Payeezy, and others

IBM Training

Enterprise cloud developer

- Complex applications
  - Often existing traditional IT applications migrated to cloud
  - Lots of business logic
  - Most often Java and Java EE
- Hybrid cloud
  - Multiple public clouds for disaster recovery
  - Private clouds (dedicated and local) for guaranteed access, geographic location, and privacy
- Hybrid applications
  - Application components hosted across multiple public, dedicated, and local clouds
- Hybrid IT
  - Application components integrate with existing on-premises resources and systems of record (SoRs)
- Limited use of cloud services
  - The existing on-premises resources are the primary services these apps need
- Specialized skills
  - Examples: IBM Cloud integration developer (and architect), IBM Cloud migration specialist, and others

IBM Training

Public and enterprise developer together

- After the enterprise developers have done their jobs, future development is mostly of the public cloud variety
  - Even if the new development is being hosted privately (dedicated and local)
- New development doesn't have to worry about enterprise concerns as much
  - Useful on-premises applications have been migrated
  - Access to useful on-premises resources has been provided securely
- New development can focus on developing new apps
  - Innovative systems of engagement
  - Rapid development
  - Using IBM Cloud's variety of services

IBM Training

Lesson 2
Cloud-native application characteristics

IBM Training

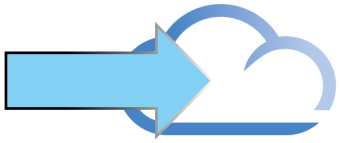
Range of application's cloud readiness

- Traditional IT
  - Stateful, transactional, vertical scaling, reliable hardware, SQL databases
  - Traditional middleware, stack products like Portal and BPM
- Cloud ready
  - Minimally sufficient to run on the cloud
  - IaaS is more flexible than PaaS, more like traditional IT servers
- Cloud native
  - Stateless, eventual consistency, horizontal scaling, commodity hardware, NoSQL databases
  - Middleware function provided by services, separate from runtimes

IBM Training

Common characteristics of an enterprise application

- Enterprise applications were usually not built with Cloud compatibility in mind, but they might have been built with future-proofing in mind
  - Trying to catch up to the latest versions of libraries, and others



- Might be built on WebSphere stack products
  - Such as portals with WebSphere Portal Server
  - Purpose-built (departmental) ESBs built with DataPower and IIB
- Deep ties into internal IT
  - Especially in cases where you not only read from, but update systems of record
  - May read from and update multiple systems not specified as APIs
  - Deep ties into corporate standards and security systems, such as LDAP, ISAM, SiteMinder
- Often have high and well-specified QoS attributes
  - Business customers have expectations on how often applications are available and how they perform

IBM Training

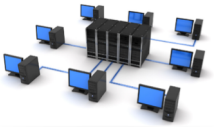
Characteristics of a cloud-native application

- Built with the cloud in mind
  - Use cloud services
    - Such as storage, queuing, and caching
  - Have rapid and repeatable deployments to maximize agility
  - Have automated setup to minimize time and cost for new developers
  - Have clean contract with underlying OS to ensure maximum portability
- Loose ties into corporate IT
  - Security often specified using open standards
    - Such as OpenID and OAuth
  - Data might be local to the application itself
- QoS attributes are those of the cloud
  - It's always expected to be there, but sometimes sites and mobile apps become unavailable, although they must be restored quickly
  - Applications must scale elastically without significant changes to tooling, architecture, or development practice
  - Application must be resilient to inevitable failures in the infrastructure and application

Introduction to Cloud-Native Application Development 11

IBM Training

Differing assumptions



- Enterprise or cloud-ready assumptions
  - The infrastructure is stable
  - The components of my application are co-located
  - My ops team controls the production servers
  - If a disaster happens, it's someone else's responsibility to fix it
- Cloud-native assumptions
  - The infrastructure is constantly changing; it is elastic
  - My application components might be globally distributed
  - As a DevOps team member, I control the production servers
    - "You build it, you run it"
  - If a disaster happens, it's my responsibility to make sure my app continues to run

Choosing one or the other has an effect on your team composition and roles

Introduction to Cloud-Native Application Development 12

IBM Training

Cloud native

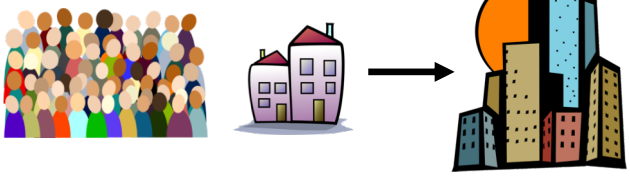
An ***application architecture*** designed to use the *strengths* and accommodate the *challenges* of a standardized cloud environment, including:

- Elastic scaling
- Immutable deployment
- Disposable instances
- Less predictable infrastructure

IBM Training

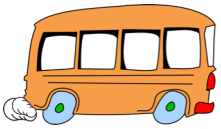
An important Cloud-native concept: Managing Capacity

32 People need to go from town to the Big City today



BUT

Your bus only holds 20 people

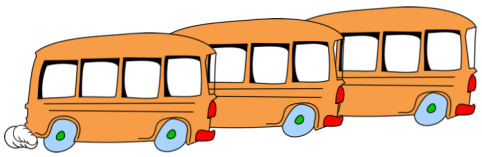


Vertical Scaling (1 bigger instance)

Double-decker bus holds 40 people

Horizontal Scaling (multiple instances)

More buses - 20 people each

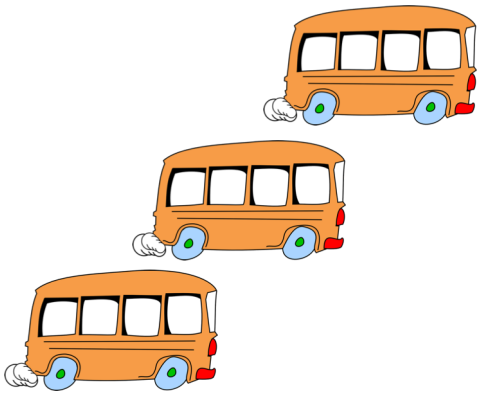


Which solution is better?

IBM Training

For Cloud-native - multiple instances are preferred – Why?

- No single point of failure
  - If a bus breaks down, you still have other buses
  - Can activate another bus to carry the stranded passengers
- Elastic scalability
  - You can add or remove buses from service to meet the demand
- Load distribution
  - Passengers don't care which bus they go on, as long as they reach their destination
- Flexibility
  - Some buses can use different roads if there is a problem on the default route
- Availability
  - You can take down buses for maintenance, and still provide your service with other buses



Introduction to Cloud-Native Application Development 15

IBM Training

Cloud-Native Application Goals

- Horizontal scaling
  - Application runs in multiple runtimes spread across multiple hosts (Twelve-Factor VIII)
- Immutable deployment
  - A runtime is not patched, it's replaced (Twelve-Factor IX)
  - A runtime is stateless (Twelve-Factor VI)
  - Shared functionality in backing services (Twelve-Factor IV)
- Elasticity
  - Automatic scale-out and scale-in to maintain performance
  - Achieved via containerization
- Pay-as-you-go charging model
  - Pay for what you use

Introduction to Cloud-Native Application Development 16

IBM Training

Recap: What it means to be cloud native

- Clean contract with underlying OS to ensure maximum portability
- Scale elastically without significant changes to tooling, architecture, or development practices
- Resilient to inevitable failures in the infrastructure and application
- Instrumented to provide both technical and business insight
- Use cloud services such as storage, queuing, and caching
- Rapid and repeatable deployments to maximize agility
- Automated setup to minimize time and cost for new developers

IBM Training

Unit summary

- Differentiate IBM Cloud developer roles
- Explain cloud-native application characteristics

IBM Training

The Twelve-Factor Application

IBM Training

Unit objectives

- Describe the twelve-factor app best practices for cloud applications
- Define and use IBM Cloud facilities that incorporate these best practices
- Describe how IBM Cloud incorporates these practices

IBM Training

Lesson 1
The twelve factors

IBM Training

The twelve-factor app

- A set of best practices for creating applications
  - Implementing, deploying, monitoring, and managing
- Typical modern applications
  - Deployed in the cloud
  - Accessible as web applications that deliver software-as-a-service (SaaS)
- Can be applied to any application
  - Implemented in any programming language
  - Using any backing services such as database, messaging, and caching
- Addresses common problems
  - The dynamics of the growth of an app over time
  - The dynamics of collaboration between developers
  - Avoiding the cost of software erosion
  - Systemic problems in modern application development
- Provides a shared vocabulary for addressing these problems

IBM Training

The twelve factors

I. **Codebase**: One codebase that is tracked in revision control, with many deployments

II. **Dependencies**: Explicitly declare and isolate dependencies

III. **Configuration**: Store Configuration in the environment

IV. **Backing services**: Treat backing services as attached resources

V. **Build, release, run**: Strictly separate build and run stages

VI. **Processes**: Execute the app as one or more stateless processes

VII. **Port binding**: Export services with port binding

VIII. **Concurrency**: Scale out using the process model

IX. **Disposability**: Maximize robustness with fast startup and efficient shutdown

X. **Development and production parity**: Keep development, staging, and production as similar as possible

XI. **Logs**: Treat logs as event streams

XII. **Admin processes**: Run administrative and management tasks as one-off processes

IBM Training

Lesson 2
Twelve-factor details

IBM Training

Factor 1: Codebase

I. **Codebase**
II. Dependencies
III. Configuration
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

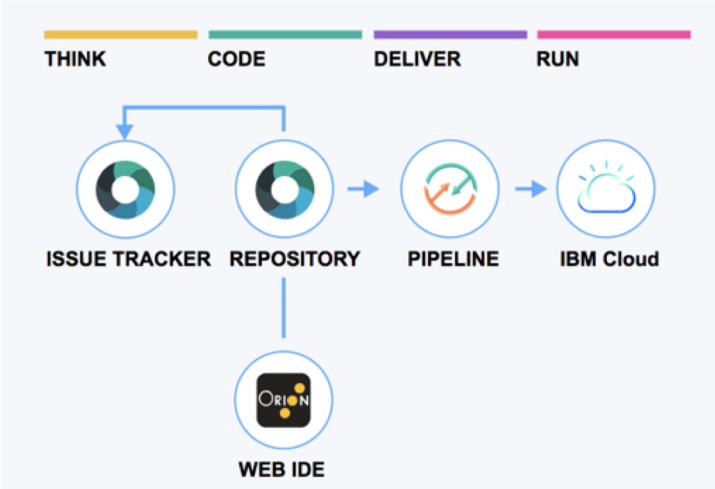
- One codebase tracked in source code management (SCM) with versioning
  - git & github most commonly used
- Multiple deployments from the same codebase

- IBM Cloud: use IBM Continuous Delivery toolchains or external automation with Cloud Foundry development tools such as these:
  - Urban Code Deploy
  - Gradle
  - Jenkins

The Twelve-Factor Application 7

IBM Training

Codebase: IBM Cloud continuous delivery toolchains



A *toolchain* is a set of tool integrations that support development, deployment, and operations tasks.

Tool integrations with the source code repository and delivery pipelines can drive multiple deployments from a single repository.

The Twelve-Factor Application 8

IBM Training

Factor 2: Dependencies

I. Codebase
II. Dependencies
III. Configuration
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

- Explicitly declare and isolate dependencies
- Typically language-specific
  - Node.js: Node Package Manager (NPM)
  - Liberty: Feature manager
  - Ruby: Bundler
  - Java EE: Application resources
- Never rely on system-wide dependencies

- IBM Cloud: Buildpack adds external dependencies

The Twelve-Factor Application 9

IBM Training

Dependencies: npm package file

- Application configuration is part of the application
- You manage it in source control

manifest.yml | app.js | package.json | index.jade | layout.jade

```
{
    "name": "MachineTranslationNodejs",
    "version": "0.0.1",
    "description": "A sample nodejs app for Bluemix that use the machine translation service",
    "dependencies": {
        "express": "3.4.7",
        "jade": "1.1.4",
        "cors": "2.4.2"
    },
    "engines": {
        "node": "0.10.26"
    },
    "repository": {}
}
```

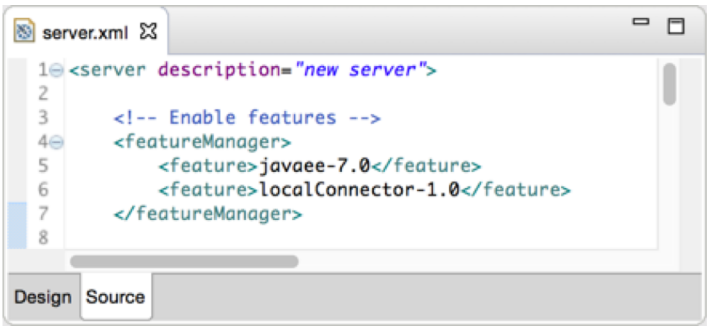
Added cors

The Twelve-Factor Application 10

IBM Training

Dependencies: Liberty feature manager

- Server configuration is part of deploying the application
- You manage it in source control



The Twelve-Factor Application 11

IBM Training

Dependencies: IBM Cloud application manifest

- Server configuration is part of deploying the application
- You manage it in source control

The Twelve-Factor Application 12

IBM Training

Factor 3: Configuration

I. Codebase
II. Dependencies
III. Configuration
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

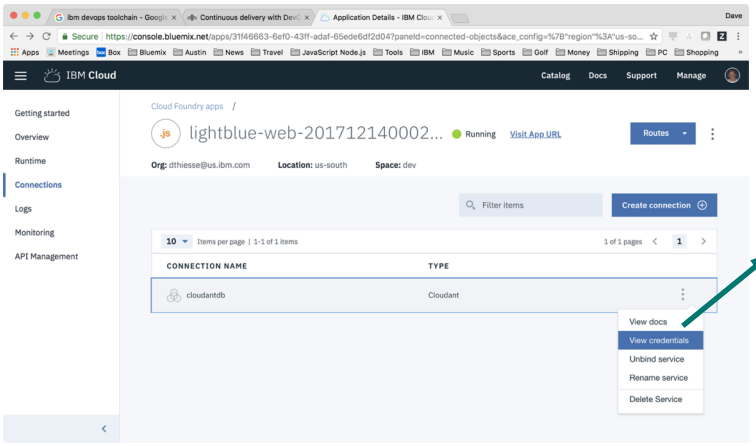
- Store configuration in the environment
- Separate configuration from source
- Enables the same code to be deployed to different environments

- IBM Cloud: You assign parameters to applications with environment variables, both system-provided and custom

The Twelve-Factor Application 13

IBM Training

Configuration: IBM Cloud VCAP_SERVICES




```
{
 "cloudantNoSQLDB": [
  {
   "credentials": {
    "username": "4e94679f-767a-4327-926f-cafef516bee0-blue
    "password": "e81181d22ae71b25106c4a8659778297b014d9
    "host": "4e94679f-767a-4327-926f-cafef516bee0-bluemix.c
    "port": 443,
    "url": "https://4e94679f-767a-4327-926f-cafef516bee0-blue
59778297b014d90cbf629aeb8dc80d2a672522d6@4e94679f-76
mix.cloudant.com"
   },
   "syslog_drain_url": null,
   "label": "cloudantNoSQLDB",
   "provider": null,
   "plan": "Lite",
   "name": "Your Application-cloudantNoSQLDB",
   "tags": [
```


The Twelve-Factor Application 14

IBM Training

Kubernetes configuration

- Kubernetes uses ConfigMap and Secret resources

```
kubectl create secret generic apikey --from-literal=API_KEY=123—456

kubectl create configmap language --from-literal=LANGUAGE=English
```


IBM Training

Factor 4: Backing services

I. Codebase
II. Dependencies
III. Configuration
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

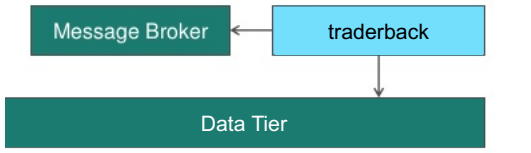
- Treat backing services as attached resources:
  - Databases
  - Messaging systems
  - LDAP servers
  - Others
- Local and remote resources should be treated identically
  Possible locations for run time and resources:
  - In the same process
  - On the same host
  - On different hosts in the same data center
  - In different data centers
- IBM Cloud: All services are bound to the run time as though they are remote, including custom user-provided services

IBM Training

Backing services: Cloud Foundry service bindings

```
cf bind-service appname service_instance
```

```
cf push traderback
cf bind-service traderback stsql
cf bind-service traderback stmessaging
```



- When deploying, do not manually bind services to an application
- Specify the services that an application needs in its manifest file

IBM Training

Backing services: Kubernetes bindings

- Bind a container in Kubernetes to an IBM Cloud service
  - Add a volume to the pod definition that stores the binding in a secret
  - By mounting the Kubernetes secret as a volume to your deployment, you make the IBM Cloud service credentials available to the container that is running in your pod.

- Example: Bind to an instance of Watson Tone Analyzer named mytoneanalyzer

```
volumes:
      - name: service-bind-volume
        secret:
            defaultMode: 420
            secretName: binding-mytoneanalyzer
```

IBM Training

Factor 5: Build, release, run

I. Codebase
II. Dependencies
III. Configuration
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

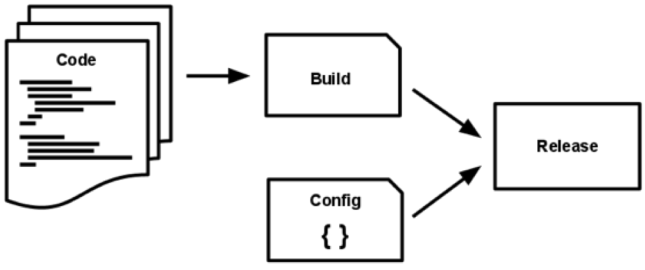
- Strictly separate build and run stages
- IBM Cloud: The output of build and release is an immutable container
  - In the IBM Cloud Kubernetes Service, the Docker build creates a container image that is stored to the private image registry.
  - You do not have to build the container again just to deploy another one

The Twelve-Factor Application 19

IBM Training

Build, release, run

- Build stage
  - converts a code repo into an executable bundle known as a build
  - fetches dependencies and compiles binaries and assets
- Release stage
  - takes the build produced by the build stage and combines it with the deploy's current config
  - ready for immediate execution in the execution environment
- Run stage
  - runs the app in the execution environment



Source: https://12factor.net

The Twelve-Factor Application 20

IBM Training

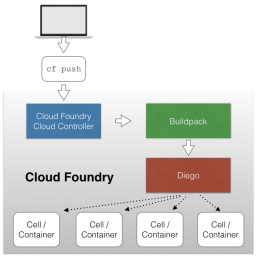
Build & release produce an Immutable image

- An immutable image does not get changed – only used for deploying instances
- If it needs to change, you delete it and create a new one

Examples

- Docker containers
  - A container image is created from a Dockerfile
  - After that, it is only deployed as a container, not changed
  - If you need to make changes, make a new container image by creating a new build and running the Dockerfile again

- Cloud Foundry
  - When an app is pushed to Cloud Foundry, the buildpack creates a droplet – from that CF creates Garden container instances that run the app
  - If you need to make changes, create a new build and push it



The Twelve-Factor Application 21

IBM Training

Factor 6: Processes

I. Codebase
II. Dependencies
III. Configuration
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

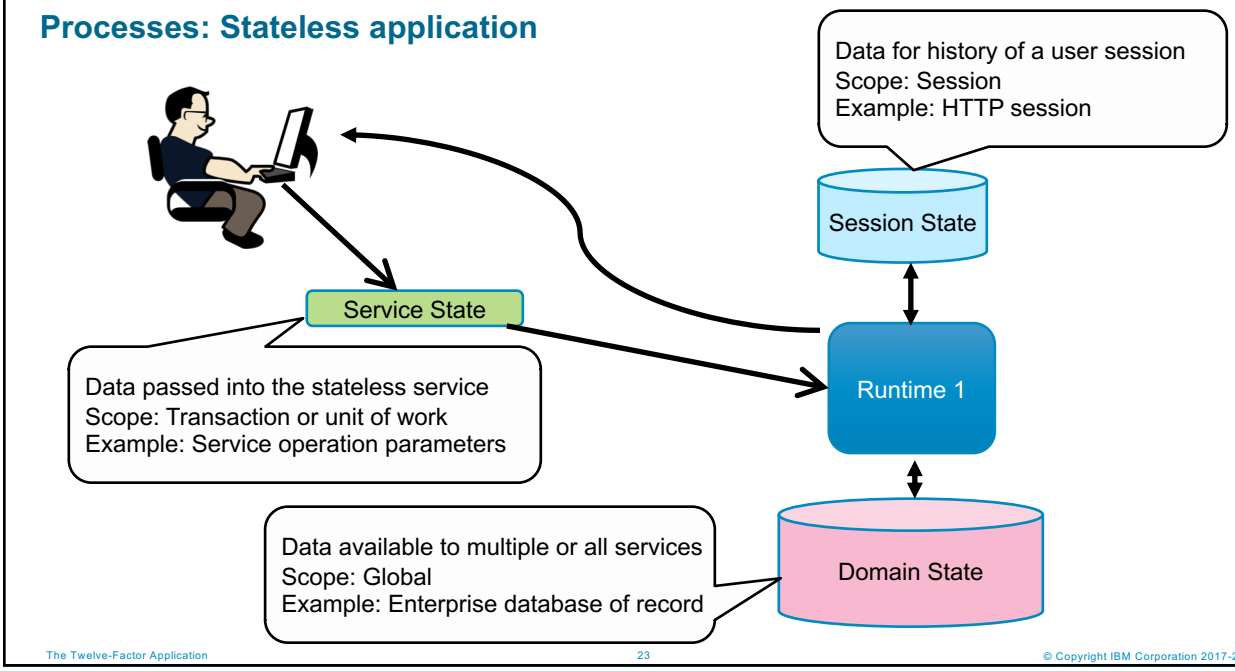
- Run the application as one or more stateless processes
- Do not rely on session affinity, also called sticky sessions

- IBM Cloud: Application instances are stateless, with the state held by service instances

The Twelve-Factor Application 22

IBM Training

Processes: Stateless application



The Twelve-Factor Application 23

IBM Training

Factor 7: Port binding

I. Codebase
II. Dependencies
III. Configuration
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

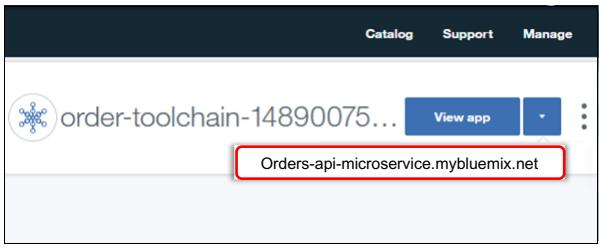
- Export services with port binding
- Web app binds to an HTTP port and listens for requests coming in on that port

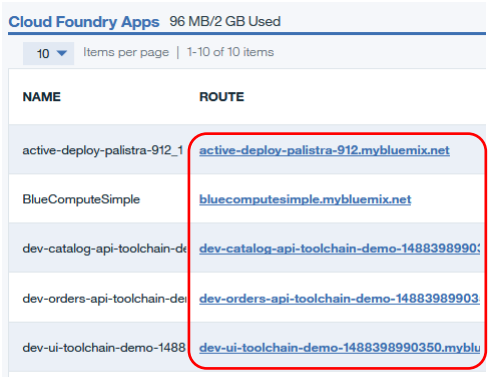
- IBM Cloud:
  - Container listens on an HTTP port
  - That port is externalized as a Cloud Foundry route or Kubernetes service

The Twelve-Factor Application 24

IBM Training

Port binding: IBM Cloud application routes





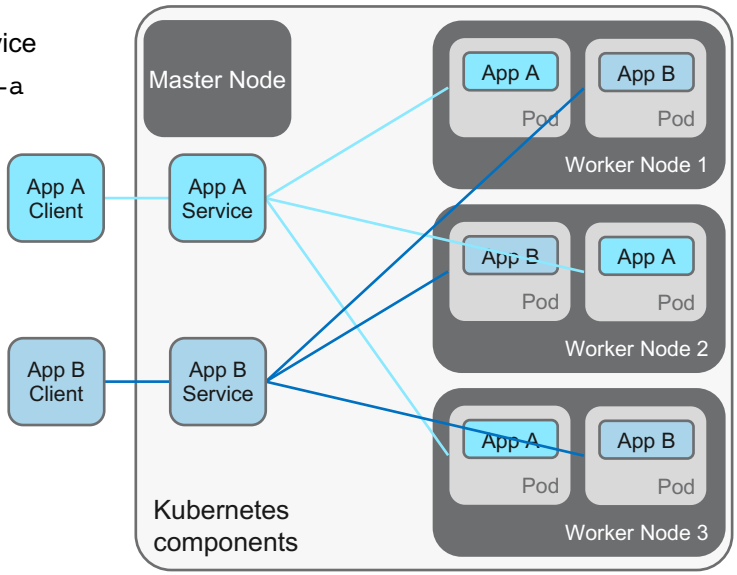
The Twelve-Factor Application 25

IBM Training

Port binding: Kubernetes service

- Expose a set of pod replicas as a service

```
kubectl expose deployment/app-a
      --type=LoadBalancer
      --port=8080
      --name=app-a-service
      --target-port=8080
```



The Twelve-Factor Application 26

IBM Training

Factor 8: Concurrency

I. Codebase
II. Dependencies
III. Configuration
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

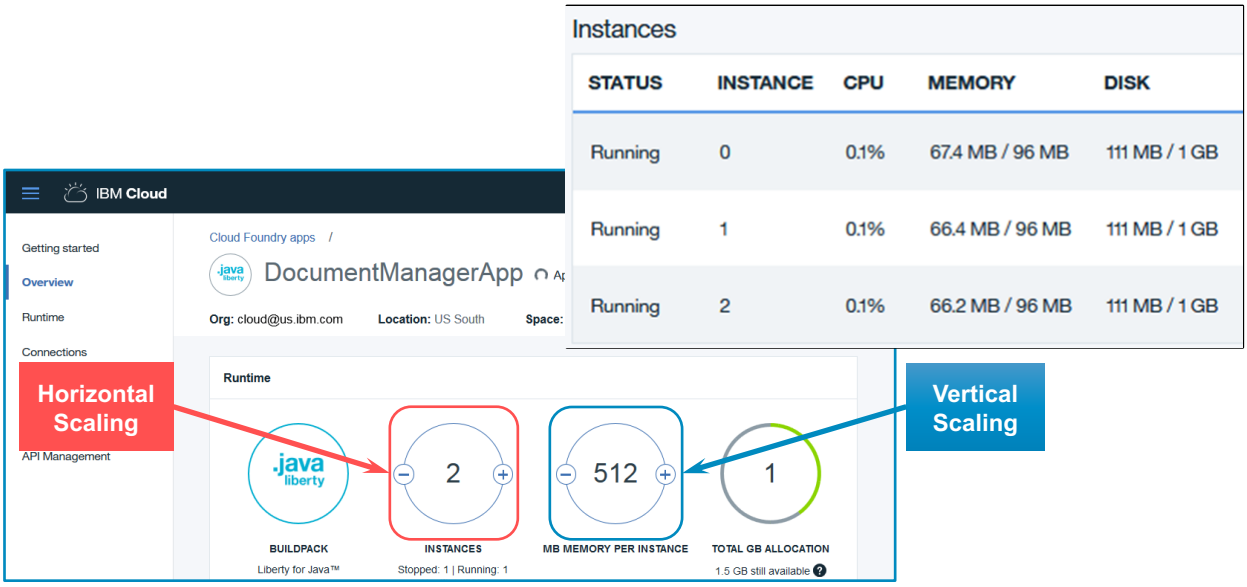
- Scale out using the process model
  To add capacity, run more instances
- There are limits to how far an individual process can scale
- Stateless applications make scaling simple

- IBM Cloud:
  - cf scale
  - Auto-Scaling service

The Twelve-Factor Application 27

IBM Training

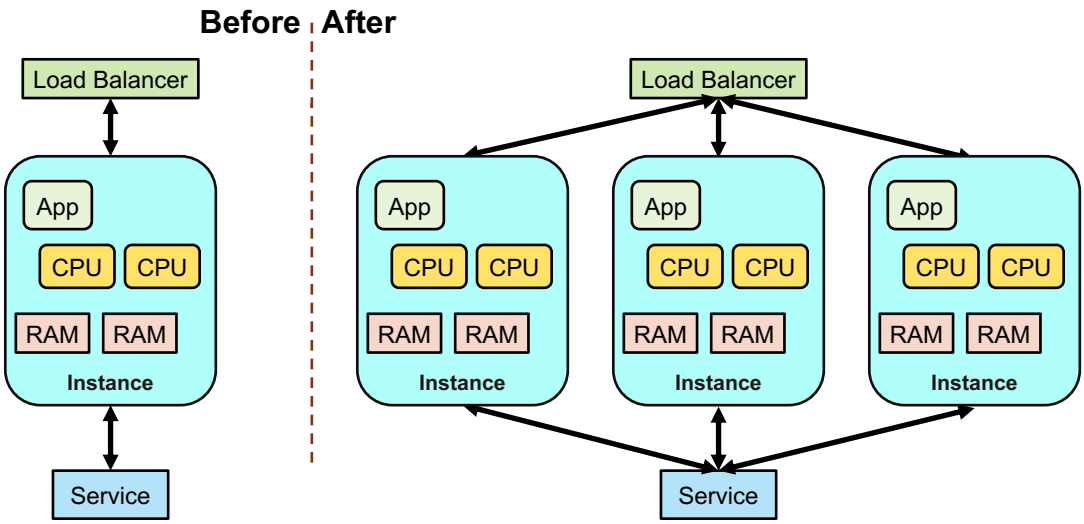
Concurrency: Runtime instances



The Twelve-Factor Application 28

IBM Training

Concurrency: Runtime instances



IBM Training

cf scale

```
cf scale appname -i number_of_instances
```

- Horizontal scaling in Cloud Foundry
  - Runs the specified number of instances of the application
  - Same as the Instances setting in the console
- Instances run independently
  - Stateless
  - Unrelated to each other
- For clustering
  - A router must distribute load
    - IBM Cloud automatically distributes across instances with the same route
  - If the app requires session state, it must be shared
    - For example, the instances can all share an instance of Redis or WebSphere eXtreme scale for session state
- The administrator manually adjusts the cluster size

IBM Training

Auto-Scaling service

- Dynamically increases and decreases application capacity
  - Makes the application *elastic*
  - Number of instances grows and shrinks based on load
- Requires no manual intervention by the administrator
  - Manual: `cf scale`
  - Automated: Auto-Scaling service
- Policy-driven: *Auto-Scaling Policy*
  - Policy declaratively specifies the proper range for these items, among others:
    - JVM heap
    - Memory
    - Throughput
    - Response time
  - Automatically keeps capacity within range

IBM Training

Factor 9: Disposability

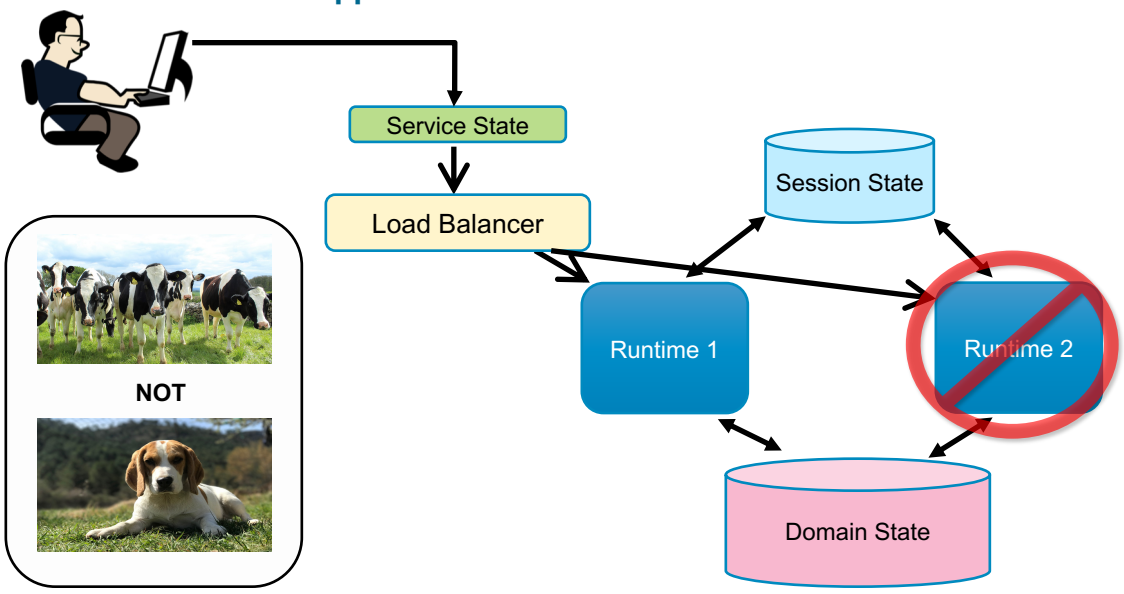
I. Codebase
II. Dependencies
III. Configuration
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

- Maximize robustness with fast startup and efficient shutdown
- Application instances are disposable
- Application should handle shutdown signal or hardware failure with crash-only design

- IBM Cloud:
  - Architecture can rapidly start and stop instances
  - Applications need to ensure they can respond

IBM Training

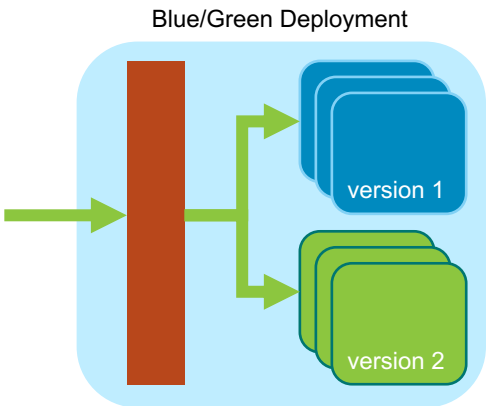
Processes: Stateless application

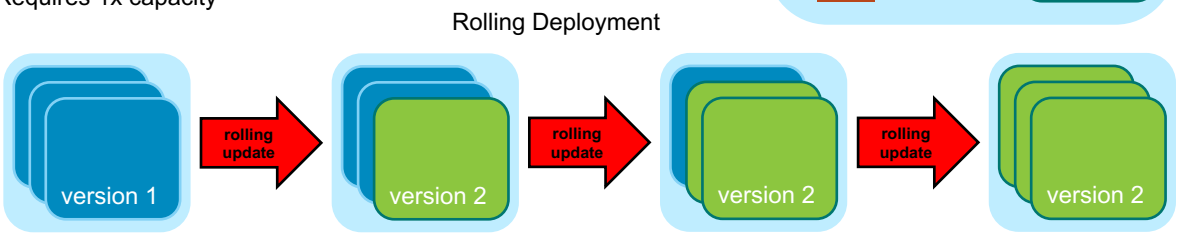


IBM Training

Zero-downtime Deployment

- Deploy a new version without causing an outage
  - Cloud makes this much easier to achieve
  - Two approaches
- Blue/green deployment
  - Deploy v2 alongside v1
  - Shift user load from v1 to v2
  - Requires 2x capacity
- Rolling deployment
  - Replace instances of v1 with v2
  - Requires 1x capacity





IBM Training

Cloud Foundry Blue-green deployment

- A zero-downtime deployment technique
  - Also known as red-black Deployment
- Two nearly identical production environments, called Blue and Green
  1. Deploy v1 (blue environment), which users access
  2. Deploy v2 (green environment)
  3. Shift client load from Blue to Green
  4. Delete Blue

`cf rename`

- Rename Blue (v1) app to Green so that you can deploy Blue (v2)
- Both share Blue's route until you delete Green, which is v1

`cf map-route`

- Push Blue, which is v1, and Green, which is v2
- Map Blue's route to both Blue and Green
- Both share Blue's route until you delete Blue, which is v1

The Twelve-Factor Application 35

IBM Training

Factor 10: Development and production parity

I. Codebase
II. Dependencies
III. Configuration
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

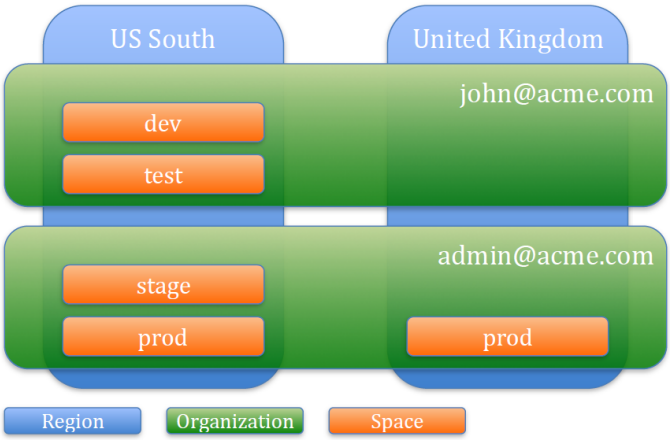
- Keep development, staging, and production environments as similar as possible
- Use the same backing services in each environment

- IBM Cloud:
  - Cloud Foundry - Use separate spaces for separate environments: (ie. Development, Test, Production)
    - Use different instances of the same services in each space
    - Deploy the same application to each space
  - Kubernetes – Use different namespaces or other methods to maintain separation between environments in the same cluster
  - Resource Groups – Can be used to segregate services

The Twelve-Factor Application 36

IBM Training

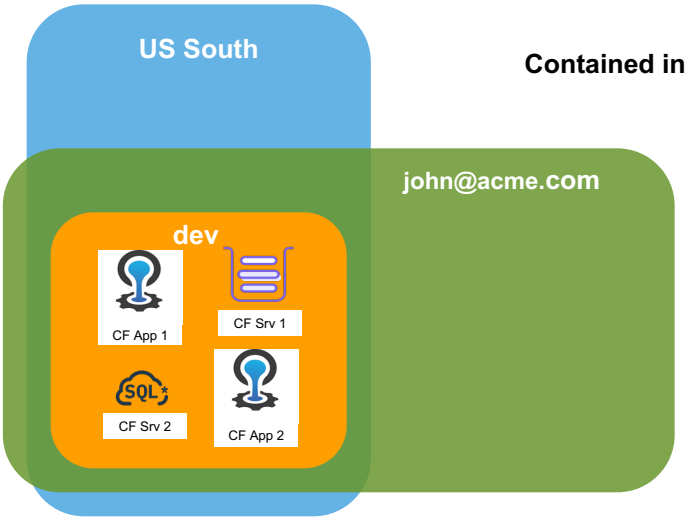
Regions, organizations, spaces, users in Cloud Foundry



- **Region**: A *region* is a specific installation of IBM Cloud. IBM Cloud public has regions like US-South, UK-South, and AP-South.
- **Organization**: An *organization* has a quota of resources available for deploying and running applications. It can span regions. It can have one or more users.
- **Space**: A *space* is a group of runtime artifacts, like applications or services, hosted in a region and belonging to an organization. Can be used in a variety of ways to group artifacts (dev, test, production).
- **User:** A *user* will have assigned roles and permissions within an organization or space.

IBM Training

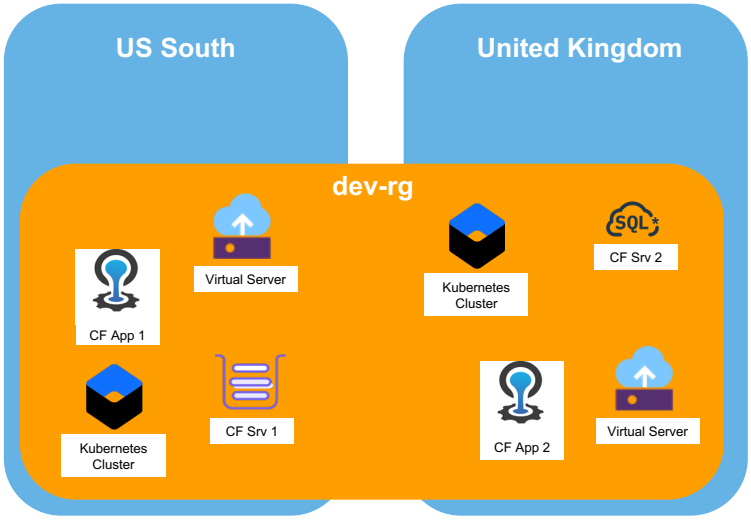
A Cloud Foundry Space can only contain Cloud Foundry Apps & Services



Contained in a single region

IBM Training

A Resource Group can contain any kind of resource



- **Can Span regions**
- **Users provided different levels of access**

IBM Training

Factor 11: Logs

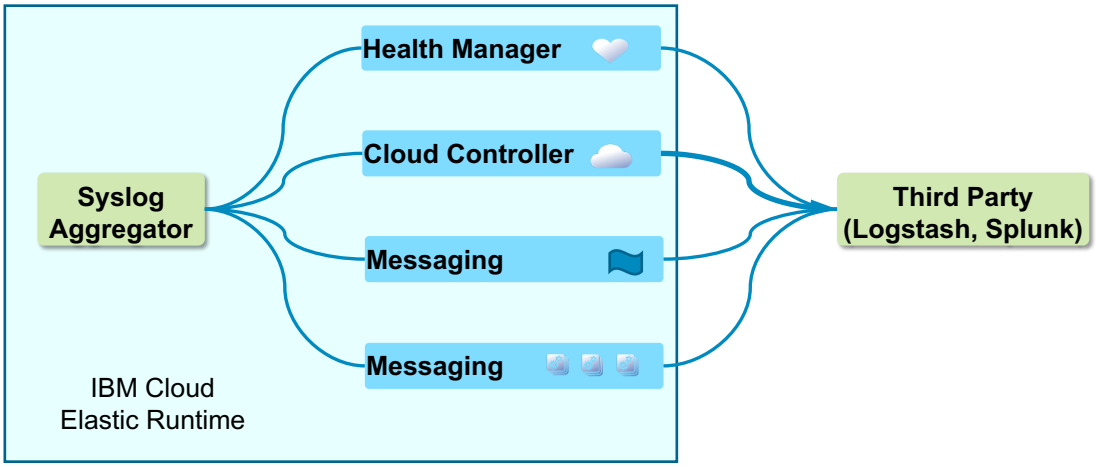
I. Codebase
II. Dependencies
III. Configuration
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

- Treat logs as event streams
- Each process writes to stdout
  - Application should not write to specialized log files
  - Environment decides how to gather, aggregate, and persist stdout output

- IBM Cloud
  - Syslog provides event streams for applications
  - Can be sent to third-party log management system

IBM Training

Logs: IBM Cloud syslog



IBM Training

Factor 12: Administrative processes

I. Codebase
II. Dependencies
III. Configuration
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

Run administrative and management tasks as single processes, such as these examples:

- Tasks for performing database migrations
- Debugging

- IBM Cloud: Push single-shot applications bound to same services

IBM Training

Administrative processes: Scripting

- Create an SQL database
  - DDL script that creates the schema
  - SQL script that populates initial data
  - Script that runs these scripts as part of creating the database
- Migrate data to a new schema
- Software-defined data center (SDDC)
- Deployment scripts as part of a CI/CD pipeline
  - Creates service instances, deploys runtimes, and binds them
- Scripts are repeatable
  - Test scripts in stage environment
  - Scripts run the same in production
- Store scripts in software configuration management (SCM) system

IBM Training

Unit summary

- Describe the twelve-factor app best practices for cloud applications
- Define and use IBM Cloud facilities that incorporate these best practices
- Describe how IBM Cloud incorporates these practices

IBM Training

Backup Charts

IBM Training

Factor 6 – Processes

Restaurant Example

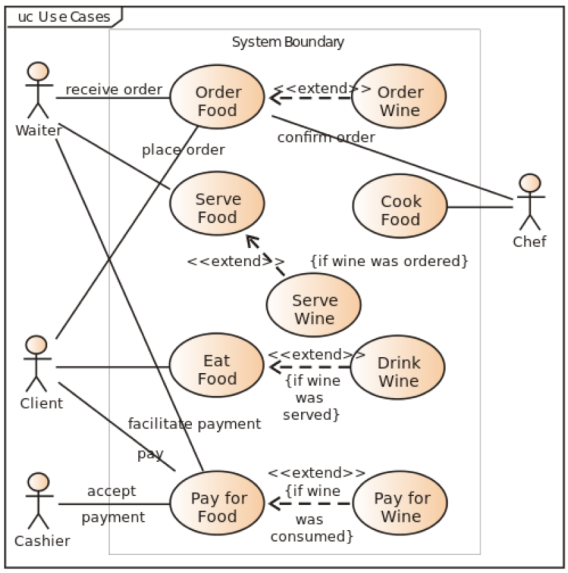
IBM Training

Here's an example from real life – Your Favorite Restaurant

Notice how the person who takes your order doesn't always bring your food? Well…

IBM Training

The Restaurant Model



Think of your waiter as a — Runtime

Each request you make is — Service State

The food & drink you've ordered but haven't yet received is — Session State

Your total order that will result in your bill is — Domain State

IBM Training

Stateless applications can be delicious

What if your waiter just tried to remember your whole order?
- What if he forgets?
- What if he gets too busy?
- What if he has to leave?

Maintaining state externally is a better idea
- Paper check, electronic
- Available to chefs, other waiters
- Builds domain state

Now others can know what needs to be done and can help deliver the service

IBM Training

IBM Cloud Private architecture

IBM Training

Objectives

- Describe differences between IBM Cloud Private editions
- List IBM Cloud Private major components
- Differentiate kubernetes and cloud foundry environment for IBM Cloud Private

IBM Training

IBM Cloud Private – editions and packaging

IBM Training

Private cloud is critical to the IBM cloud strategy

- **Choice with consistency**
- **Hybrid integration**
- **DevOps productivity**
- **Powerful, accessible data and analytics**
- **Cognitive solutions**

PUBLIC	DEDICATED	PRIVATE
Maximize on cloud agility and economics	Public cloud benefits, with dedicated infrastructure	Behind the firewall for the most demanding workloads

Seamless Experience
Regardless of which combination you choose, you get a single, seamless experience.

IBM Training

IBM Cloud Private – bringing cloud native to the enterprise

Rapid Innovation

- Open Kubernetes-based container platform
- Cloud Foundry for app dev and deployment
- DevOps toolchain integration

Hybrid Integration

- Integration capabilities to unlock and connect
- Secure access to public cloud services (AI, Blockchain)
- Consistent experience across private/public

Investment Leverage

- Containerized versions of IBM Middleware
- Prescriptive guidance to optimize workloads
- Work with existing apps, data, skills, infrastructure

Management & Compliance

- Core operational services including logging, monitoring, security
- Flexibility to integrate with existing tools and processes

IBM Cloud Private architecture 5 © Copyright IBM Corporation 2018

IBM Training

Enterprise grade content with open source technologies

IBM Middleware & Open Source – e.g. Data, Analytics and Developer Services

Cloud-enabled middleware, application runtimes, messaging, databases & analytics to optimize current investments and rapidly innovate

IBM WebSphere · IBM DB2 · IBM Data Science Experience · open source

Core Operational Services

To simplify Operations Management, Security, DevOps, and hybrid integration

Grafana · Prometheus

kubernetes

HELM

Kubernetes-based Container Platform

Industry leading container orchestration platform across private, dedicated & public clouds

CLOUDFOUNDRY

Cloud Foundry

For prescribed application development & deployment

Terraform

Terraform (CAM)

Infrastructure as Code for provisioning on public and on-prem cloud

IBM Cloud · Microsoft · amazon web services · openstack · vmware

Runs on existing IaaS (On Premise – e.g. Vmware, Openstack OR off Premise - e.g. SL, AWS) or hardware from IBM POWER and Z, Dell, Cisco, NetApp, Lenovo, ...

IBM Cloud Private architecture 6 © Copyright IBM Corporation 2018

IBM Training

Built with open standards, preventing vendor lock-in

docker	kubernetes	HELM	Terraform
Executable package of software that includes everything needed to run it	Automate deployment, scaling, and management of containerized applications	Define, install, and upgrade Kubernetes applications	Infrastructure as code to provision public cloud and on-premises environments
Containers	**Orchestration**	**Management**	**Provisioning**

IBM Training

Solution Overview

Enterprise Content Catalog
Open Source and IBM Middleware, Data, Analytics, and AI Software

Core Operational Services
Log Management, Monitoring, Security, Alerting

Kubernetes Container Orchestration Platform

kubernetes

docker

Choose your infrastructure: Power Systems, openstack, IBM Spectrum, vmware, IBM Z, intel

Strategic Values:

- Self-service catalog
- Agility, scalability, and elasticity
- Self-healing
- Enterprise security
- No vendor lock-in

IBM Training

Enterprise Content Catalog

Toolchain & Runtimes
Microclimate Beta
Microservice Builder
Jenkins (open source)
IBM WebSphere Liberty
Open Liberty (open source)
IBM SDK for Node.js
Swift runtime (open source)

Logging & Monitoring Services
ELK (open source) & Prometheus (open source)

App Modernization Tooling
IBM Transformation Advisor

Multi-cloud Management
IBM Cloud Automation Manager

Mobile
IBM Mobile Foundation

Digital Business Automation
IBM Operational Decision Manager for Developers

Data Services
IBM Db2 Dev-C
IBM Data Server Manager (for Db2 Dev-C)
IBM Db2 Direct Advanced Edition / AESE with Data Server Manager
IBM Db2 Warehouse Dev-C
IBM Db2 Warehouse Enterprise
IBM Cloudant Developer Edition
MongoDB (open source)
PostreSQL (open source)
MariaDB (open source)
Galera with MariaDB (open source)

Messaging
IBM MQ Advanced for Developers
IBM MQ Advanced
Rabbit MQ (open source)

Integration
IBM Integration Bus for Developers
IBM Integration Bus
IBM DataPower Gateway for Developers
IBM DataPower Gateway Virtual Edition

Data Science and Business Analytics
IBM Data Science Experience Developer Edition
IBM Data Science Experience Local

Data Governance and Integration
IBM InfoSphere Information Server for Evaluation

Cognitive Connectivity
IBM Voice Gateway Developer Trial

Tooling
Web Terminal (open source)
Skydive – network analyzer (open source)

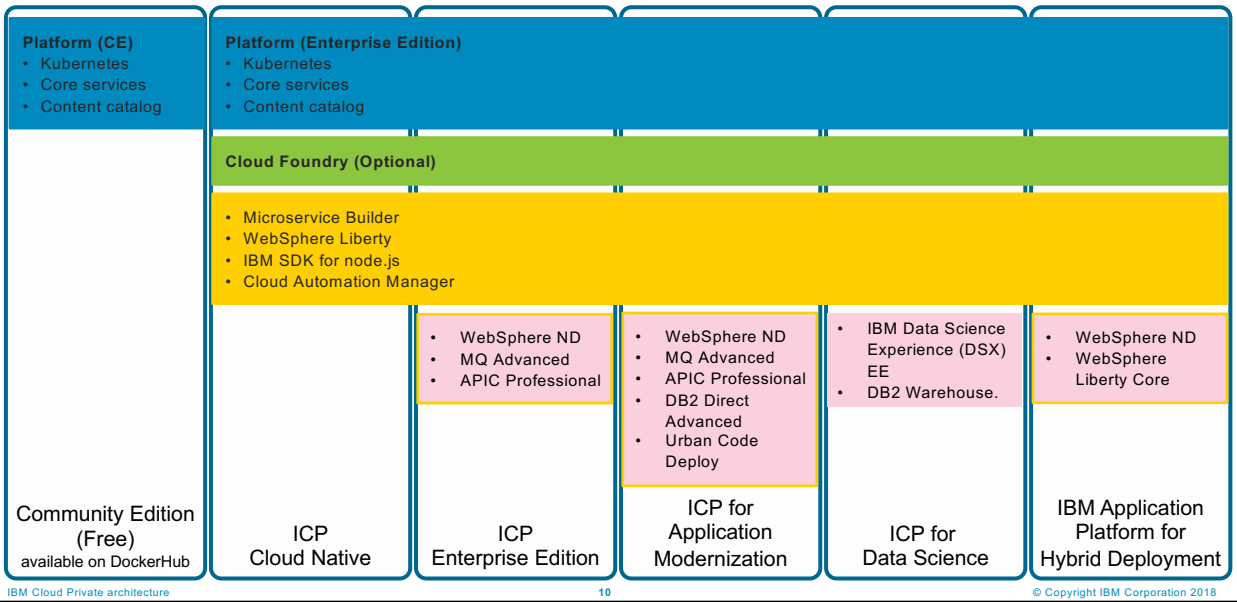
HPC / HPDA
IBM Spectrum LSF Community Edition
IBM Spectrum Symphony Community Edition

Bring your own
Add a Helm repo or your own charts

Catalog content is not distributed with IBM Cloud Private. Content is distributed separately, licensed under separate terms and conditions.

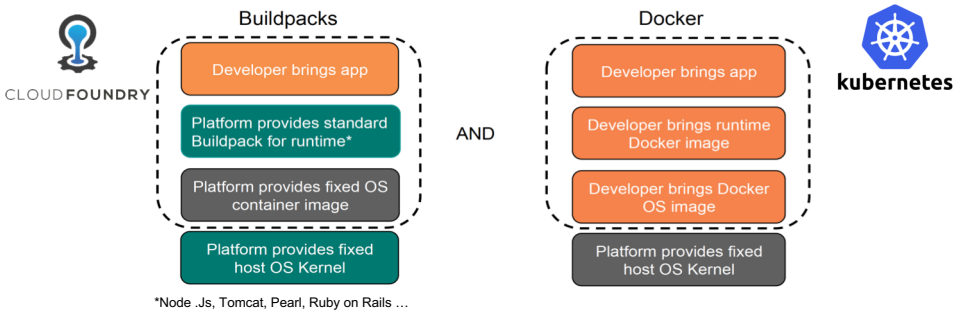
IBM Training

IBM Cloud Private editions



IBM Training

Cloud Foundry and Kubernetes



*Node .Js, Tomcat, Pearl, Ruby on Rails …

- **CF and Kubernetes (K8s) are operating on different levels of abstraction**
- **Choices to make**
  - K8s: flexibility to control all of the underlying technology, and freedom to deploy whatever - wherever at large scale – Optimizing for Performance & Scalability.
  - CF: the platform take care of all of the "plumbing" to get the code you wrote into a running application, monitor it's health, and scale at the expense of control and flexibility – Optimizing for Speed & Simplicity
- **IBM's recommendation – Use Both platforms at their best**
  - Do rapid prototyping on Cloud Foundry PaaS
  - Harden prototypes & deploy on an Enterprise Grade K8 platform

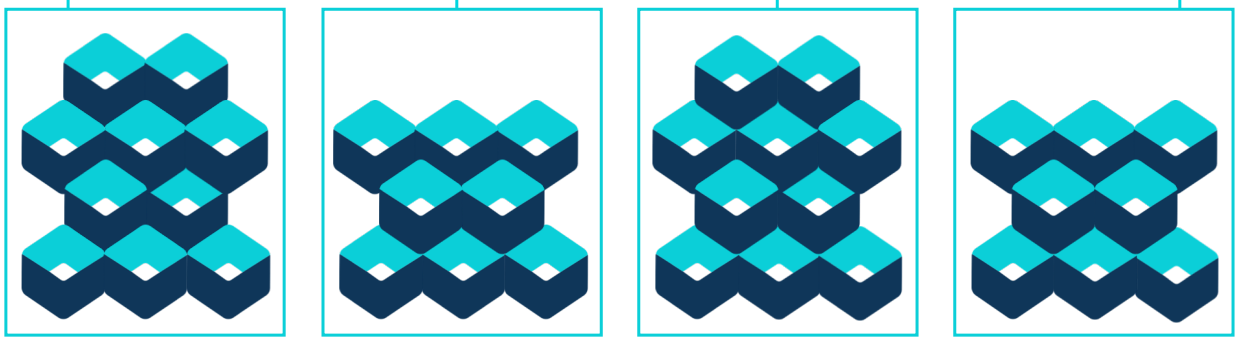
IBM Training

IBM Cloud Private – Kubernetes cluster

IBM Training

Kubernetes – (Κυβερνήτης - Captain in Greek)

Regain control with Containers and Kubernetes
- Organize and Govern the Container Chaos



IBM Cloud Private architecture

IBM Training

What does Kubernetes offer?

Intelligent Scheduling	Automatically places containers based on their resource requirements and other constraints, while not sacrificing availability. Mix critical and best-effort workloads in order to drive up utilization and save even more resources.
Self Healing	Restarts containers that fail, replaces and reschedules containers when nodes die, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
Horizontal Scaling	Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage.
Service Discovery and Load Balancing	No need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives containers their own IP addresses and a single DNS name for a set of containers, and can load-balance across them.
Automated rollout and rollback	Kubernetes progressively rolls out changes to your application, while monitoring application health to ensure it doesn't kill all your instances at the same time. If something goes wrong, Kubernetes will rollback the change for you. Take advantage of a growing ecosystem of deployment solutions.
Secret and configuration management	Deploy and update secrets and application configuration without rebuilding your image and without exposing secrets in your stack configuration.

IBM Cloud Private architecture

IBM Training

And HELM is …

The package manager for Kubernetes

Helm is the best way to find, share, and use software built for Kubernetes.

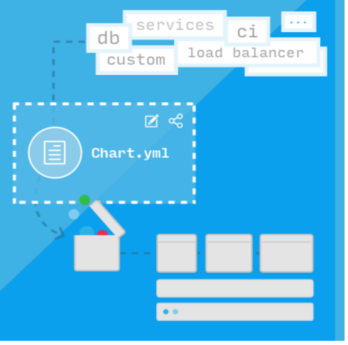
Tells Kubernetes all it needs to know about an application its parameters and dependencies

What is Helm?

Helm helps you manage Kubernetes applications — Helm Charts helps you define, install, and upgrade even the most complex Kubernetes application.

Charts are easy to create, version, share, and publish — so start using Helm and stop the copy-and-paste madness.

The latest version of Helm is maintained by the **CNCF** - in collaboration with **Microsoft**, **Google**, **Bitnami** and the **Helm contributor community**.

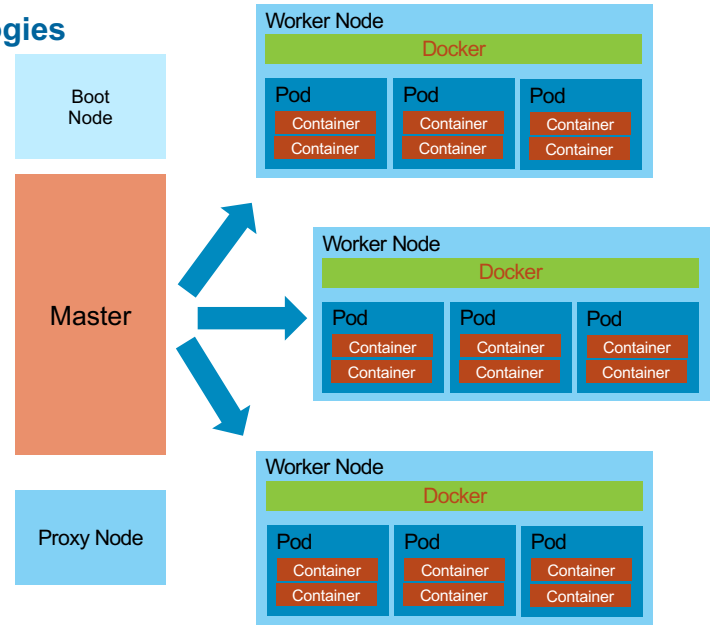


IBM Cloud Private architecture

IBM Training

Kubernetes Configuration topologies

- Simple
  - Single machine install (master is a worker)
  - Great for testing and learning about the platform
- Standard
  - Single master (single master, 3 workers, 1 proxy)
  - Great for non-production testing environment
- High Availability
  - Multiple masters (3 masters, 3+ workers, 3 proxy)
  - Production installation



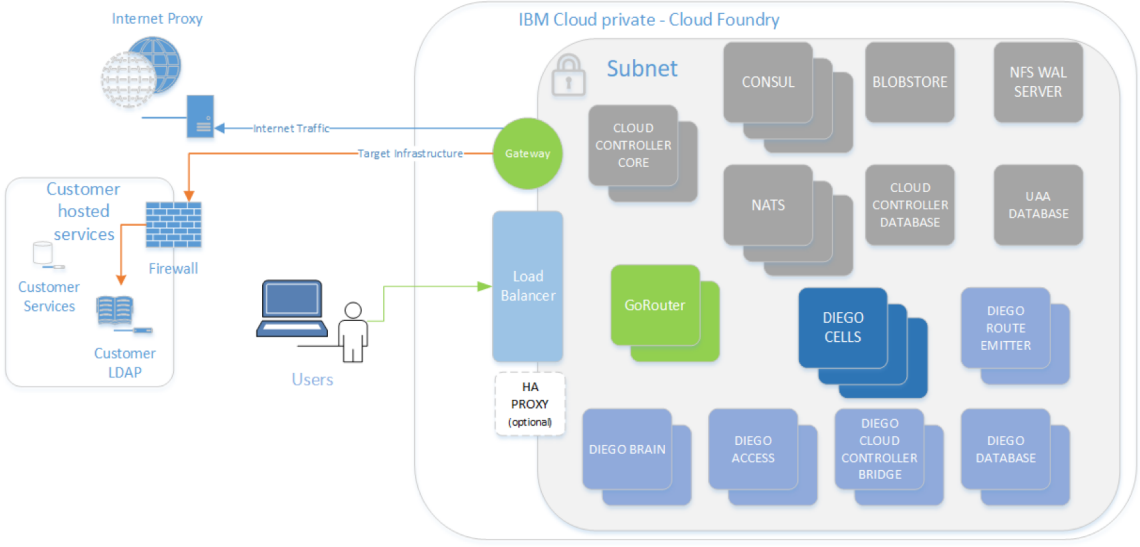
IBM Cloud Private architecture

IBM Training

IBM Cloud Private – Cloud Foundry

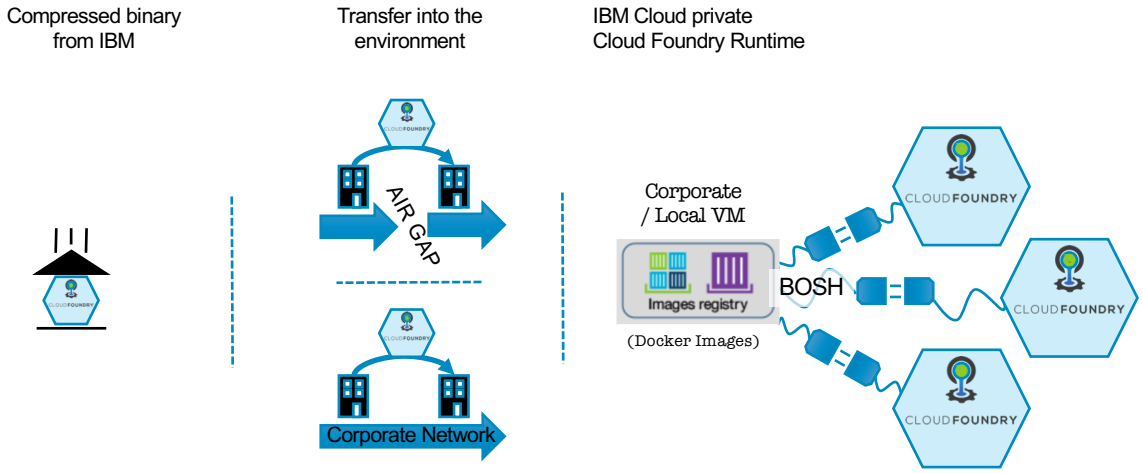
IBM Training

Cloud Foundry Architecture Diagram



IBM Training

IBM Cloud Private – Cloud Foundry

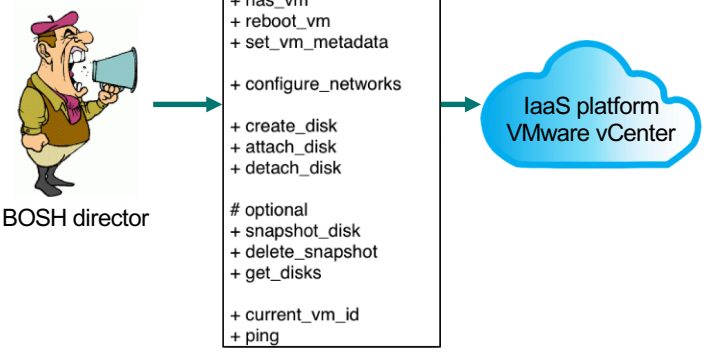


IBM Cloud Private architecture

IBM Training

BOSH components

- BOSH is a project that unifies release engineering, deployment, and lifecycle management of small and large-scale cloud software. (https://bosh.io)
- The inception machine contains the IBM supplied code to build the initial BOSH director and coordinate the deployment
- BOSH director works with the vCenter to setup the Cloud Foundry PaaS environment
- BOSH director talks to the IaaS platform using Cloud Platform Interface (CPI)



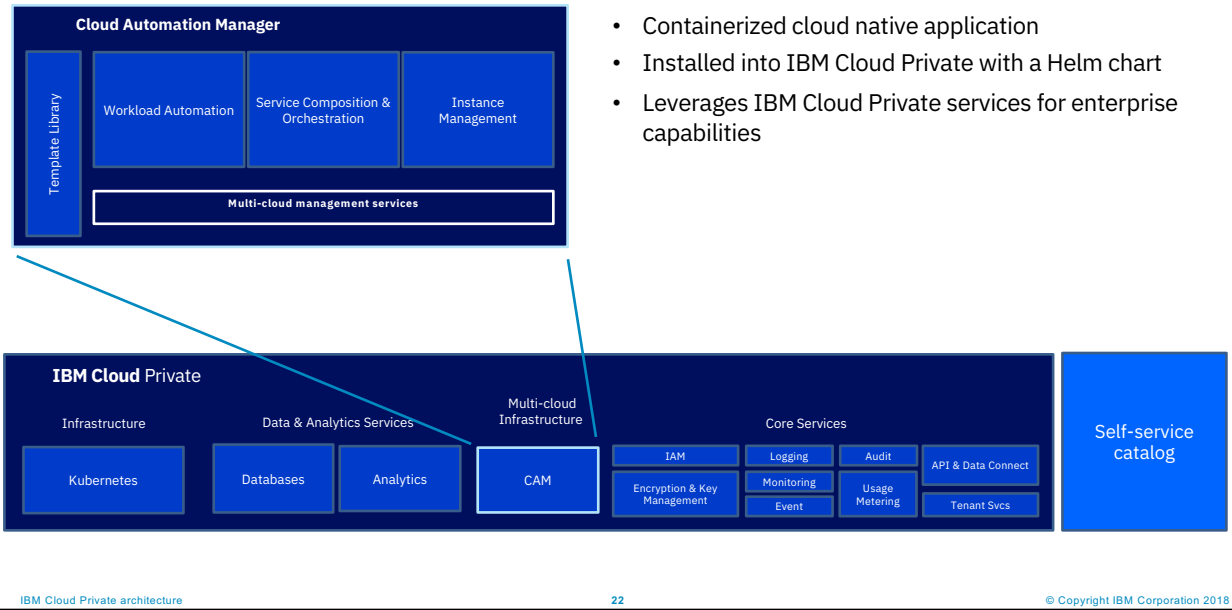
IBM Cloud Private architecture 20

IBM Training

Cloud Automation Manager

IBM Training

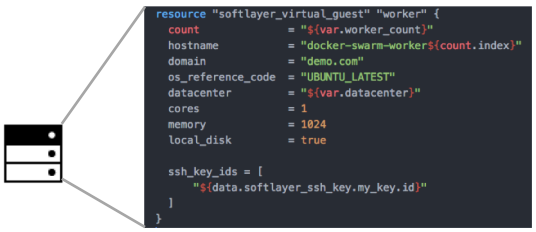
Cloud Automation Manager



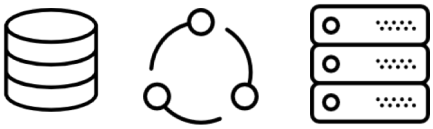
- Containerized cloud native application
- Installed into IBM Cloud Private with a Helm chart
- Leverages IBM Cloud Private services for enterprise capabilities

IBM Training

Terraform



Terraform is an infrastructure as code software. It allows users to define a datacenter infrastructure in a high-level configuration language, from which it can create an execution plan to build the infrastructure in a service provider such as IBM SL , AWS, Google, MSFT as well as Open Stack & VMWare



IBM Cloud Private architecture

IBM Training

Introduction to Cloud Full Stack Development

IBM Training

Agenda

- Introduction
- Reference architecture
- Architectural components
- Development process

IBM Training

Introduction

IBM Training

Challenge

- How should I develop an application in IBM Cloud?
- This question can be challenging to answer
  - Every solution is custom and unique
  - IBM Cloud has many available services, but not all services are used by all applications
    - 80 / 20 rule
- There is an easier way
  - The basics of most applications running in IBM Cloud are the same
  - You don't need to learn all of IBM Cloud to implement applications
  - Learn the general architectural view and add other components/services as needed

IBM Training

Solution

- **Cloud Full Stack Development**
  - How to develop a typical application in IBM Cloud
    - End-to-end (a.k.a. full stack): From client GUIs through server app to backends
  - Reference architecture shows a typical, complete but basic application in IBM Cloud
  - Captures the sweet spot: 20% of IBM Cloud Public product and skills that are used 80% of the time
- Forms the common foundation for more advanced applications in IBM Cloud

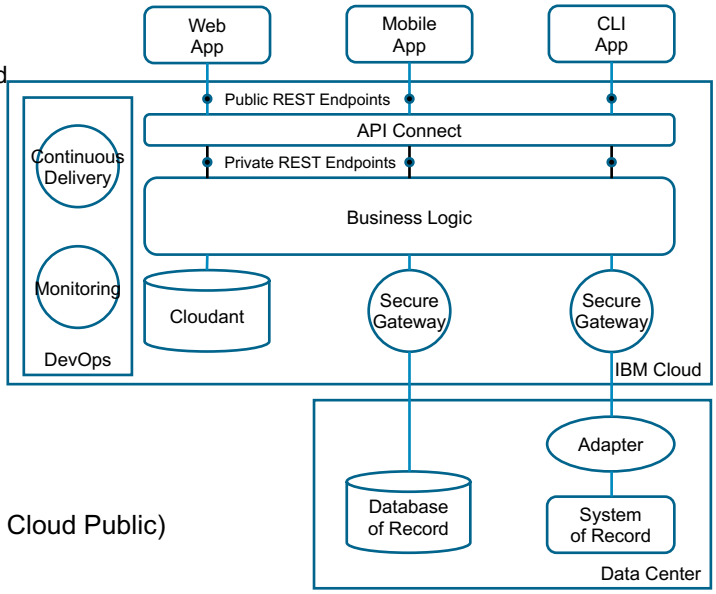
IBM Training

Reference Architecture

IBM Training

Cloud Full Stack Reference Architecture

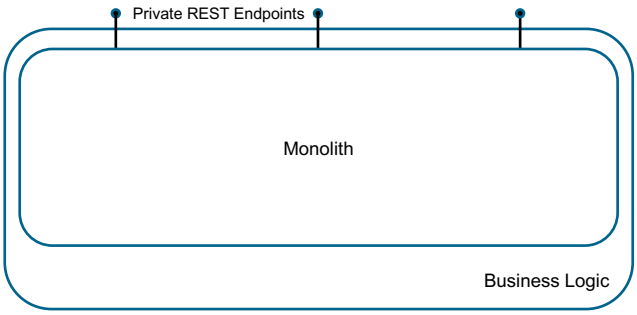
- Core architecture
  - For typical, basic applications in IBM Cloud
- Five layers
  1. Clients
     - Web, mobile, CLI, etc.
  2. API gateway (API Connect)
  3. Business logic
  4. Backends
     - Cloudant, on-prem resources, etc.
  5. DevOps
- Three business logic architectures
  1. Monolith
  2. Dispatchers + Monolith
  3. Microservices
- Two business logic technologies (in IBM Cloud Public)
  1. Cloud Foundry
  2. Docker containers in Container Service



IBM Training

Business Logic Architecture: Monolith

- Business logic architecture #1
  - Monolith
- REST API endpoints are all implemented by a single set of code running in a single process

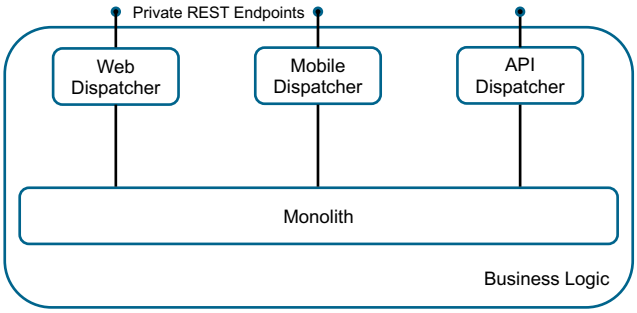


IBM Training

Business Logic Architecture: Dispatchers + Monolith

- Business logic architecture #2
  - Dispatchers + Monolith

- Each REST API endpoint is implemented by a separate dispatcher
  - I.E. Backend-for-frontend (BFF)
- All dispatchers route to a single set of code running in a single process



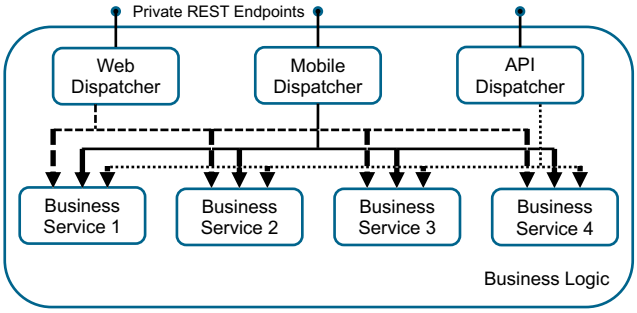
Introduction to IBM Cloud Full Stack Development 9 © Copyright IBM Corporation 2017

IBM Training

Business Logic Architecture: Microservices

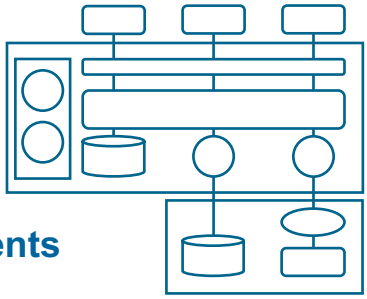
- Business logic architecture #3
  - Microservices

- Each REST API endpoint is implemented by a separate dispatcher (BFF)
- Dispatchers route to a combination of business services
  - I.E. Single-task processes
- Also teaches microservices discovery
  - Netflix OSS, Amalgam8, ISTIO



Introduction to IBM Cloud Full Stack Development 10 © Copyright IBM Corporation 2017

IBM Training



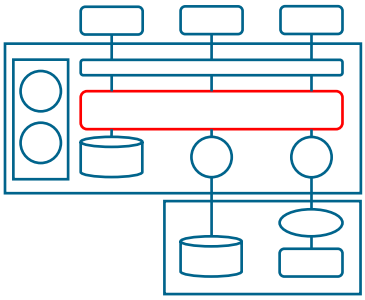
Architectural Components

11

IBM Training

Business Logic

- This is the custom code (Java, PHP, etc.) that makes your app unique
  - In IBM Cloud Public (or any PaaS), custom code is actually a much smaller part of the overall app
  - A lot of the behavior is implemented in (backing) services
    - What used to be the responsibility of middleware, such as the WebSphere stack products
- The app still starts with business logic
  - What do you want to do that makes this app unique?
- But let's first discuss the other parts of the architecture
  - How business logic fits into the rest of the architecture

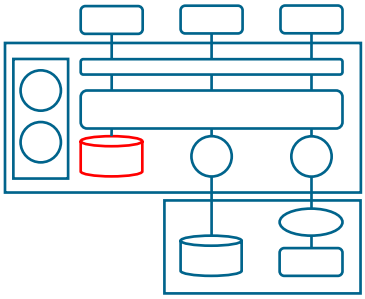


Introduction to IBM Cloud Full Stack Development 12

IBM Training

Backend: Cloudant NoSQL Database

- Pretty much every application needs to persist data
- NoSQL databases are more cloud-friendly than SQL databases
  - Horizontal scalability: better reliability and throughput
  - Documents: more natural than records, tables, and O/R mapping
- IBM Cloud catalog contains about 15 databases
  - An IBM Cloud developer doesn't need to learn them all
- Cloudant is a great general-purpose OLTP NoSQL database
  - Based on CouchDB, with lots of additional features
  - Available in IBM Cloud Public, Dedicated, and Private
- The first database cloud developers should learn is Cloudant

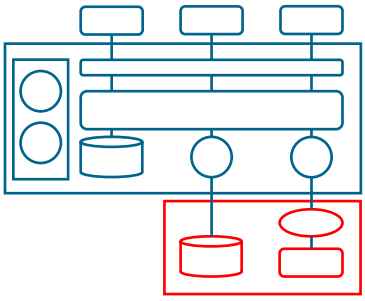


Introduction to IBM Cloud Full Stack Development 13

IBM Training

Data Center: Existing on-prem resources

- Many IBM and Enterprise customers have existing data centers
  - This on-premises technology is Traditional IT, not Cloud
- Existing systems that cannot easily be moved
  - Enterprise database of record
  - System of record
- Protected behind firewalls
  - Cannot be accessed publicly
  - Customer is unwilling to expose publicly
- IBM Cloud applications need to be able to access these existing systems
  - IBM Cloud Public and Dedicated are public, yet need private access

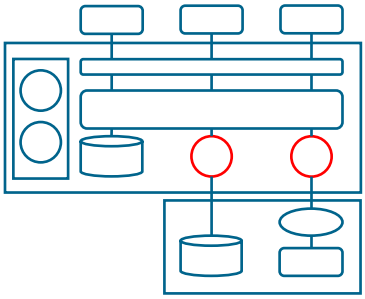


Introduction to IBM Cloud Full Stack Development 14

IBM Training

Backend: Secure Gateway

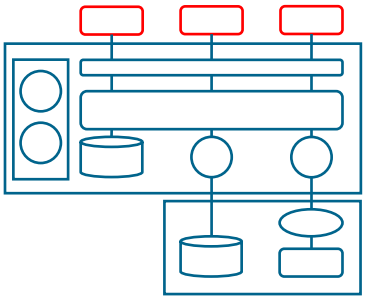
- IBM Cloud Public applications need to be able to access on-prem existing systems
  - IBM Cloud Public and Dedicated are public, yet need private access
- Secure Gateway service (IBM Cloud Public and Dedicated)
  - A quick, easy, and secure solution for connecting anything to anything
  - Gateway – Runs in IBM Cloud
  - Gateway client – Runs in the customer's data center (or other target)
  - Gateway establish a secure, persistent connection between client and IBM Cloud
  - Granular system-level access, not coarse network-level access like a VPN
- Use Secure Gateway to access on-prem systems and databases of record



IBM Training

Clients: Web, Mobile, CLI, etc.

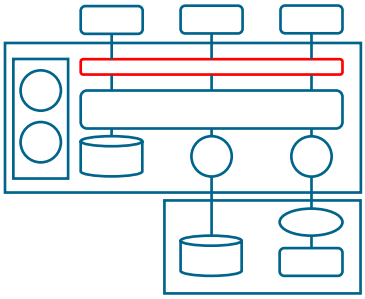
- Clients enable users to access the server application
  - Clients provided by the server team or business partners
- Clients are multimodal – Different ways to access the same experience
  - Web – Clients that run in Web browsers – HTML 5, CSS 3, JavaScript
  - Mobile – Clients that run on phones and tablets – iOS, Android, etc.
  - CLI – Clients written by business partners, often complete applications
- Use clients to provide GUIs for server applications



IBM Training

API Gateway: API Connect

- A gateway enables external clients to connect to internal server resources
  - Such connections need to be secured to block client attacks
- API Connect service
  - Gateway – Enforces policies that control access to services' APIs
  - Catalog – Lists available services, manages their lifecycle
  - Adapter – Connects gateway to services with inadequate APIs
- Use API Connect as an API gateway to control clients' access to services

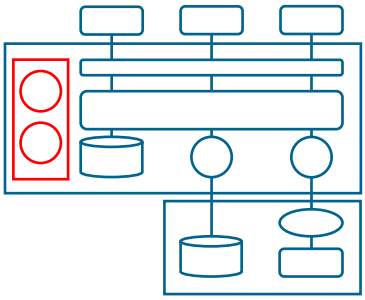


Introduction to IBM Cloud Full Stack Development 17 © Copyright IBM Corporation 2017

IBM Training

DevOps: Continuous Delivery and Monitoring

- DevOps ensures the application's quality
  - Delivered efficiently
  - Running properly
- Continuous Delivery service
  - A continuous integration/continuous delivery pipeline (CI/CD)
  - Each code change rapidly packaged, tested, and deployed into production
- Monitoring and Analytics service
  - Logs, runtimes, and services monitored for error conditions
  - Ensures an application successfully deployed into production is still running properly



Introduction to IBM Cloud Full Stack Development 18 © Copyright IBM Corporation 2017

IBM Training

Development Process

IBM Training

Overview and considerations

- Basic process
  - Phase 1: Monolith in Cloud Foundry
  - Phase 2: Dispatchers + Monolith in Cloud Foundry
  - Phase 3: Microservices in Cloud Foundry
  - Phase 4: Microservices in Docker containers
  - Phase 5: Microservices fabric and autoscaling
  - All phases: Add the rest of the architecture as needed
- Considerations
  - Monolith in Cloud Foundry: Simplest way to get started
    - Show how easy IBM Cloud is, show value quickly
  - Docker preferred?
    - If no opinion, Cloud Foundry is simpler
    - PaaS is simpler and provides greater value than CaaS
    - Postpones issues like microservices fabric
  - Microservices preferred?
    - If not, monolith is easier until the team gets too big
    - MVPs start as monoliths
    - Microservice architectures start as monoliths (or duoliths)

IBM Training

All phases: Add the rest of the architecture as needed

- As you're developing business logic
  - Regardless of architecture or technology
  - Add other architectural components as needed
  - Agile: Work on the other components in their own iterations
- When you need a GUI for testing and demos
  - Create a simple Web client
- When the business logic needs persistence
  - Add Cloudant
  - Make code modular, separate database/service instance per module
- When the business logic needs to connect to an on-prem resource
  - Create a gateway in Secure Gateway
  - Configure the gateway and client to connect to the resource(s)
- When connecting clients from an untrusted network
  - Expose the business logic's APIs via API Connect

IBM Training

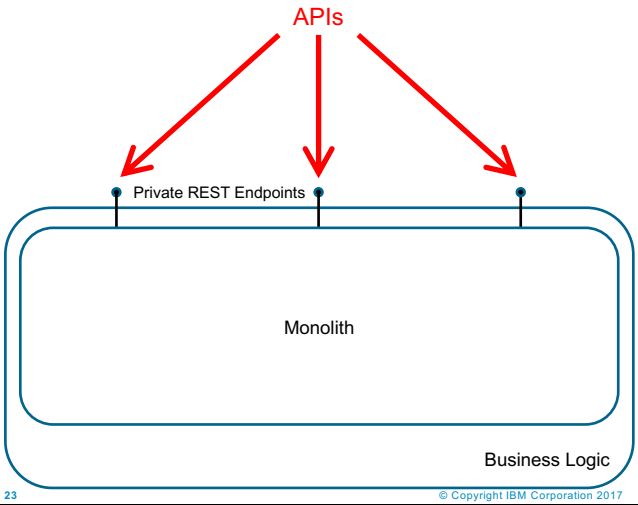
Phase 1: Monolith in Cloud Foundry

- The simplest way to get started
- Monolith
  - What developers are already used to implementing
  - Least disruptive to their development teams and processes
- Cloud Foundry
  - Full Platform-as-a-Service (PaaS), not just Container-as-a-Service (CaaS)
  - Builds the runtime for you
  - Built-in service discovery
- Migrating to IBM Cloud
  - Migrating an existing application to IBM Cloud usually starts at this phase
  - The monolithic application needs to be cloud-ready (stateless, non-transactional, etc.)

IBM Training

Expose business logic as APIs

- Expose the business logic's functionality as APIs
  - RESTful endpoints (JSON/HTTP)
- Make the code modular
  - Separate modules for separate functionality
  - Each module exposes a single REST endpoint
  - For independent usage scenarios, create separate REST endpoints



Introduction to IBM Cloud Full Stack Development 23

IBM Training

Phase 2: Dispatchers + Monolith in Cloud Foundry

- Refactor the dispatchers out of the monolith
  - Enables the dispatchers to be developed by the same teams developing the clients
  - Dispatchers can be developed and deployed independently of the monolith and each other
- Business functionality can continue to be developed as a monolith
- Cloud Foundry
  - Continues to be the simplest way to deploy runtimes
  - Simplest for both the monolith and the dispatchers

Introduction to IBM Cloud Full Stack Development 24

IBM Training

Phase 3: Microservices in Cloud Foundry

- Refactor the monolith into microservices
  - If the monolith code is modular, it will be easier to refactor into microservices
- Focus on developing good microservices
  - One business task per service
  - Business services implement requirements accurately
  - Advanced microservices fabric (autoscaling, circuit breaker, etc.) can come later
- Cloud Foundry
  - Simplest way to deploy each microservice
  - Built-in service discovery

IBM Training

Phase 4: Microservices in Docker containers

- Package the microservices as Docker containers
  - Each dispatcher and each business service in its own container
- Deploy the containers in IBM Cloud Container Service
  - May need service discovery beyond what Kubernetes provides
- Incremental step to container complexity
  - Microservices are already designed
  - Focus on building containers, deploying them, integrating them

IBM Training

Phase 5: Microservices fabric and autoscaling

- Deploy microservices configured within a fabric for better QoS
  - Service registry and discovery
  - Circuit breaker, commands, and retry/alternate path
  - Monitoring, dashboards, and business transaction tracing
- Deploy microservices configured to run elastically
  - Horizontally scale based on client load
  - Crash recovery and health monitoring
- Focus on microservices packaged as Docker containers deployed in IBM Cloud Container Service
  - Can also apply to microservices deployed in Cloud Foundry
  - Different techniques needed for Cloud Foundry

IBM Training

Conclusion

IBM Training

Summary

We've now discussed

- Introduction
- Reference architecture
- Architectural components
- Development process

IBM Training

Introduction to Containers and Docker

IBM Training

Agenda

- Why containers?
- Docker containers
- Conclusion

IBM Training

Why Containers?

IBM Training

Containers

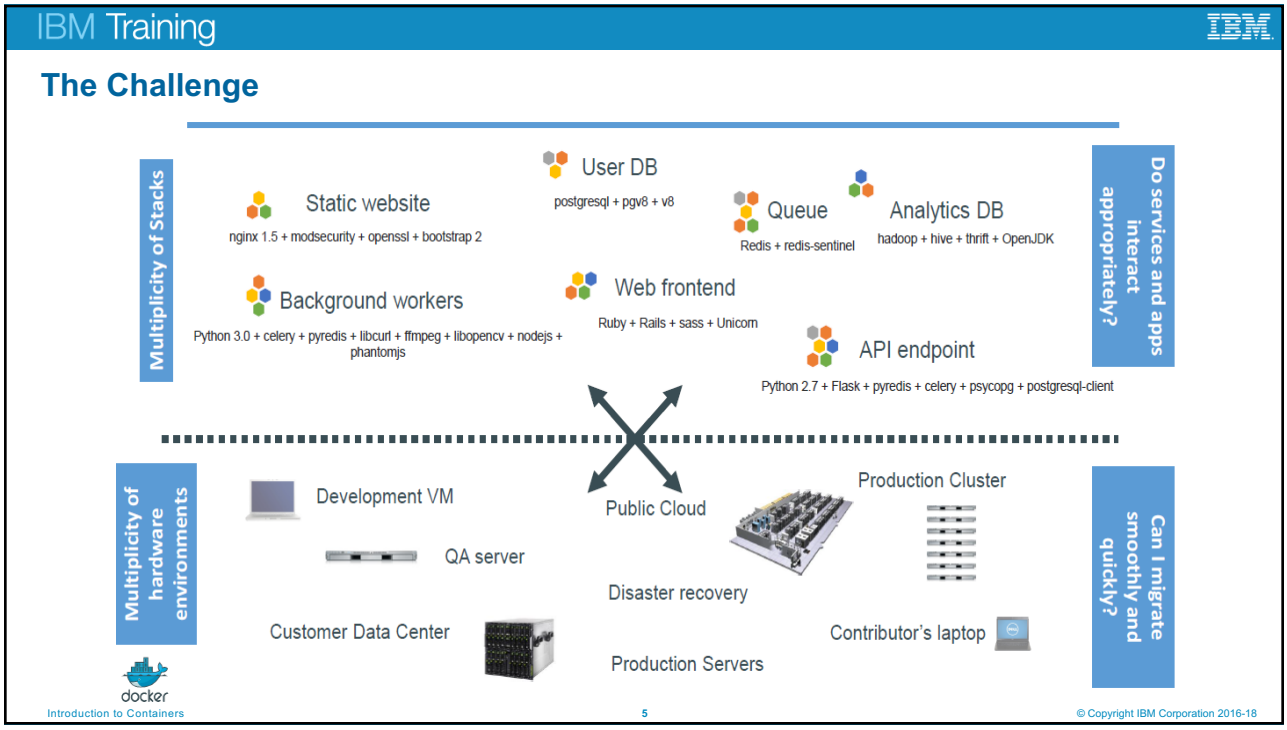
A standard way to **package** an application and all its **dependencies** so that it can be **moved** between environments and run **without change**

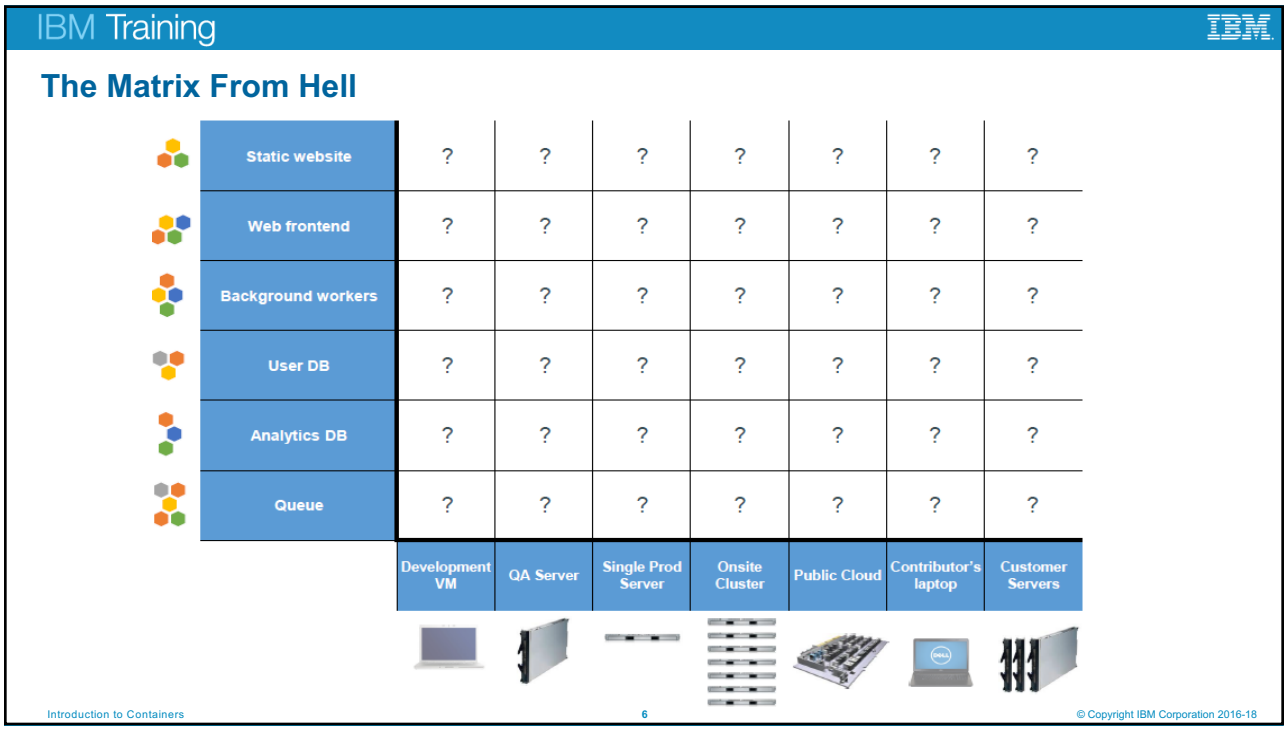
Containers work by **hiding the differences** between applications **inside** the container so that everything **outside** the container can be **standardized**

Docker: Created standard way to create images for Linux Containers

Linux Containers (LXC) details

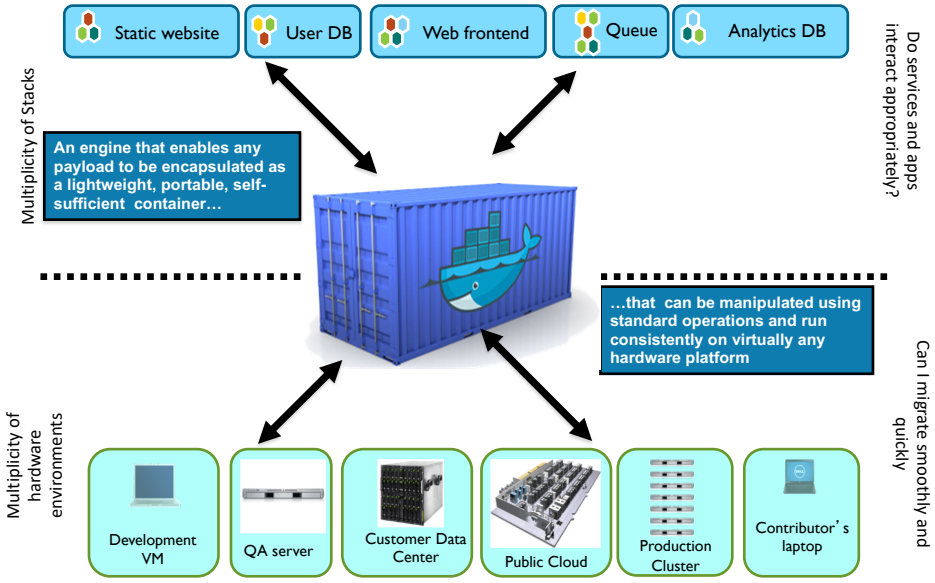
- An isolated user space within a running Linux OS
- Shared kernel across containers
- Direct device access
- All packages and data in an isolated runtime, saved as a filesystem
- Resource management implemented with control groups (cgroups)
- Resource isolation through namespaces


IBM Training
IBM
The Challenge
Multiplicity of Stacks
Static website
nginx 1.5 + modsecurity + openssl + bootstrap 2
User DB
postgresql + pgv8 + v8
Queue
Redis + redis-sentinel
Analytics DB
hadoop + hive + thrift + OpenJDK
Background workers
Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs
Web frontend
Ruby + Rails + sass + Unicorn
API endpoint
Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client
Do services and apps interact appropriately?
Multiplicity of hardware environments
Development VM
QA server
Customer Data Center
Public Cloud
Production Cluster
Disaster recovery
Production Servers
Contributor's laptop
Can I migrate smoothly and quickly?
docker
Introduction to Containers
5
© Copyright IBM Corporation 2016-18


IBM Training
IBM
The Matrix From Hell
Static website
Web frontend
Background workers
User DB
Analytics DB
Queue
Development VM
QA Server
Single Prod Server
Onsite Cluster
Public Cloud
Contributor's laptop
Customer Servers
Introduction to Containers
6
© Copyright IBM Corporation 2016-18

IBM Training

A Shipping Container for Code



Introduction to Containers 7 © Copyright IBM Corporation 2016-18

IBM Training

Dev vs Ops

Dev	Ops

- **Code**
- **Libraries**
- **Configuration**
- **Server runtime**
- **OS**

- **Logging**
- **Remote access**
- **Network configuration**
- **Monitoring**

Separation of concerns

A container separates and bridges the Dev and Ops in DevOps

- Dev focuses on the application environment
- Ops focuses on the deployment environment

Introduction to Containers 8 © Copyright IBM Corporation 2016-18

IBM Training

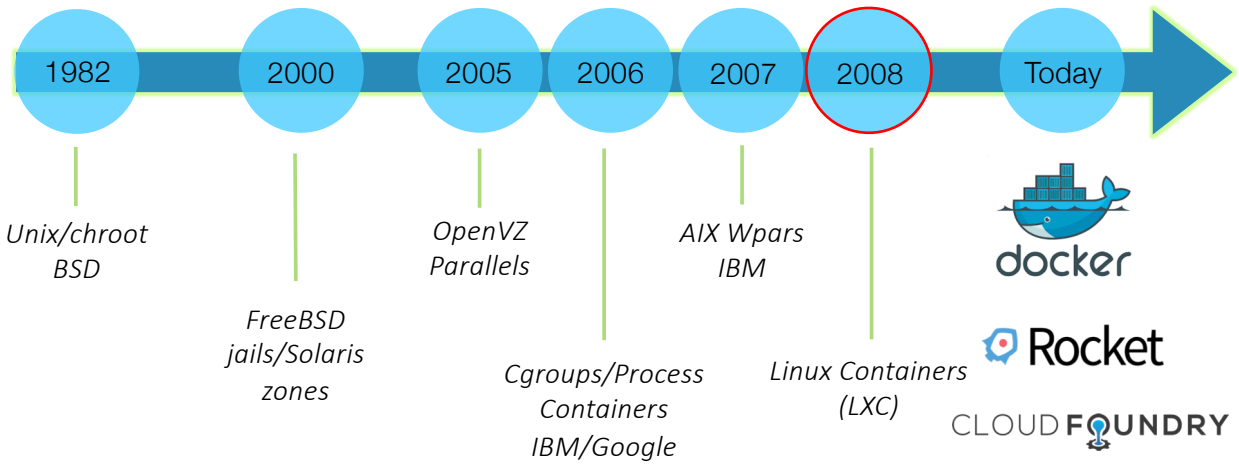
Everybody Loves Containers

- A standard way to package an application and all its dependencies so that it can be moved between environments and run without changes
- Containers work by isolating the differences between applications inside the container so that everything outside the container can be standardized

Introduction to Containers 9 © Copyright IBM Corporation 2016-18

IBM Training

Container History Lesson



Introduction to Containers 10 © Copyright IBM Corporation 2016-18

IBM Training

Multiple (De facto) Container Standards

- Docker
  - The most common standard, made Linux containers usable by the masses
- Rocket (rkt)
  - An emerging container standard from CoreOS, the company that developed etcd
- Garden
  - The format Cloud Foundry builds using buildpacks
- Open Container Initiative (OCI)
  - A Linux Foundation project developing a governed container standard

Introduction to Containers 11

IBM Training

Advantages of Containers

- Containers are portable
  - Any platform with a container engine can run containers
- Containers are easy to manage
  - Container images are easy to share, download, and delete
    - Especially with Docker registries
  - Container instances are easy to create and delete
  - Each container instance is easy and fast to start and stop
- Containers provide “just enough” isolation
  - More lightweight than virtual machines
  - Processes share the operating system kernel but are segregated
- Containers use hardware more efficiently
  - Greater density than virtual machines
    - Especially Docker containers, which can share layers
- Containers are immutable
  - Container images are versions
  - Containers cannot (should not) be patched

Introduction to Containers 12

IBM Training

Docker Containers

13

IBM Training

Docker Adoption

Docker enables application development **efficiency**, making deployment more **efficient**, eliminating vendor 'lock-in' with true **portability**

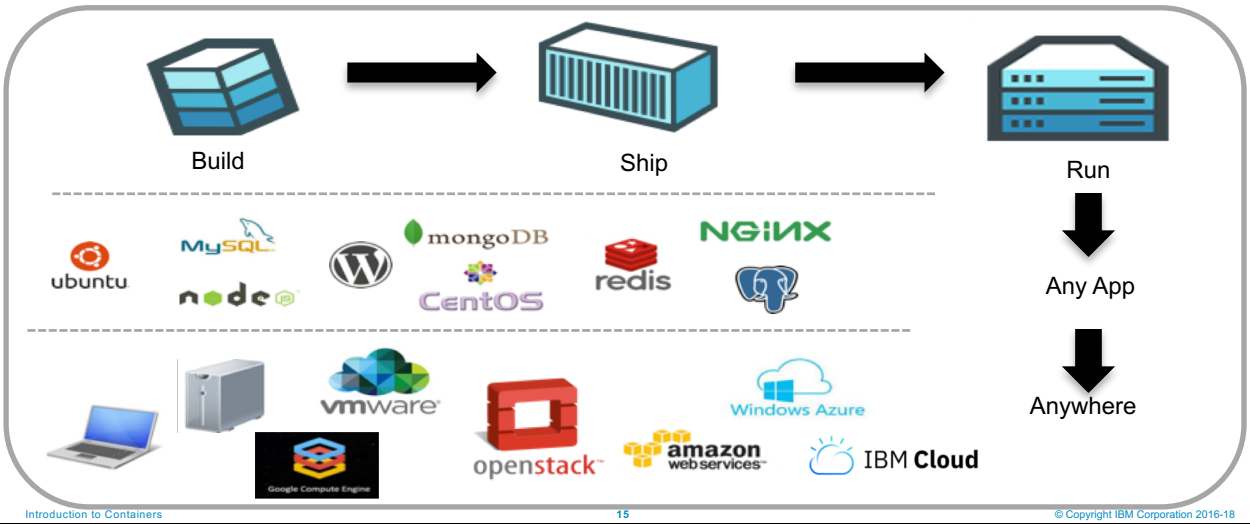
- Open software
  - Launched **March 2013**
  - 2.0+ billion downloads of Docker images
- Open contribution
  - 2000+ contributors
  - #2 most popular project
  - 185 community meet-up groups in 58 countries
- Open design
  - Contributors include IBM, Red Hat, Google, Microsoft, VMware, AWS, Rackspace, and others
- Open governance
  - Docker, the Open Container Initiative (OCI), and the Cloud Native Computing Foundation (CNCF) are jointly developing container standards

Introduction to Containers 14

IBM Training

Docker Mission

Docker is an **open platform** for building distributed applications for **developers** and **system administrators**

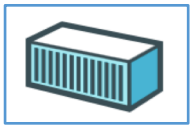


IBM Training

Docker Components

Image

- A read-only snapshot of a container stored in a registry to be used as a template for building containers

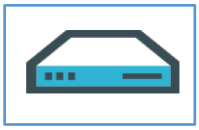


Container

- The standard unit in which the application service resides or transported

Docker Hub/Registry

- Available in SaaS or Enterprise to deploy anywhere you choose
- Stores, distributes, and shares container images

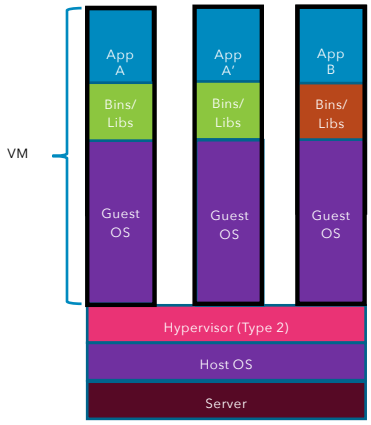


Docker Engine

- A program that creates, ships, and runs application containers
- Runs on any physical and virtual machine or server locally, in private or public cloud
- Client communicates with Engine to execute commands

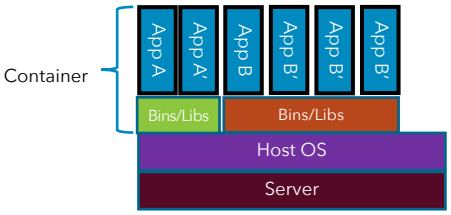
IBM Training

VMs vs Containers



Containers are isolated, but share OS and, where appropriate, bins/libraries

…faster, less overhead

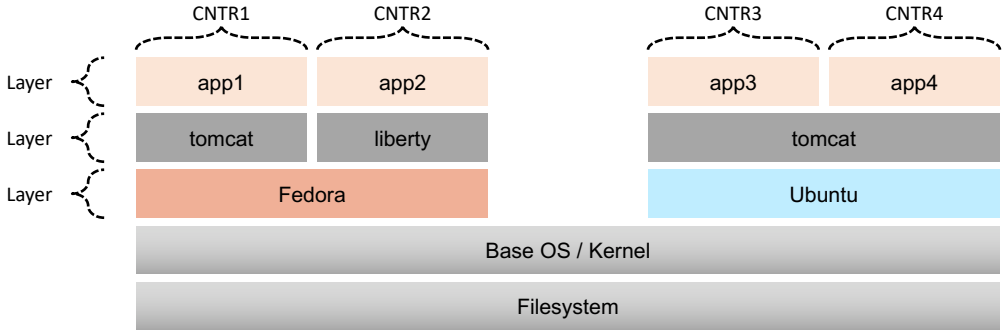


Introduction to Containers 17

IBM Training

Docker Containers

- Docker uses a copy-on-write (union) filesystem
- New files (& edits) are only visible to current/above layers



- Layers allow for reuse
  - More containers per host
  - Faster start-up/download time – base layers are "cached"
- Images
  - Tarball of layers (each layer is a tarball)

Introduction to Containers 18

IBM Training

Why Customers are Interested in Docker Containers

- Containers are a critical foundation for hybrid apps
- Ship more software
  - Accelerate development and CI/CD pipelines by eliminating headaches of setting up environments and dealing with differences between environments
  - On average, Docker users ship software 7X more frequently
- Resource efficiency
  - Lightweight containers run on a single machine and share the same OS kernel while images are layered file systems sharing common files to make efficient use of RAM and disk and start instantly
- App portability
  - Isolated containers package the application, dependencies and configurations together
  - These containers can then seamlessly move across environments and infrastructures

Introduction to Containers 19

IBM Training

Conclusion

20

IBM Training

Conclusion

- Why containers?
- Docker containers

IBM Training

Resources

- Docker tutorial
  - https://docs.docker.com/get-started/
- The Evolution of Linux Containers and Their Future
  - https://dzone.com/articles/evolution-of-linux-containers-future
- rkt
  - https://coreos.com/rkt
- Cloud Foundry Garden container
  - https://docs.cloudfoundry.org/concepts/architecture/garden.html
- Open Container Initiative (OCI)
  - https://www.opencontainers.org
- Cloud Native Computing Foundation (CNCF)
  - https://www.cncf.io
- Demystifying the Open Container Initiative (OCI) Specifications
  - https://blog.docker.com/2017/07/demystifying-open-container-initiative-oci-specifications

IBM Training

Introduction to Kubernetes

kubernetes

IBM Training

Agenda

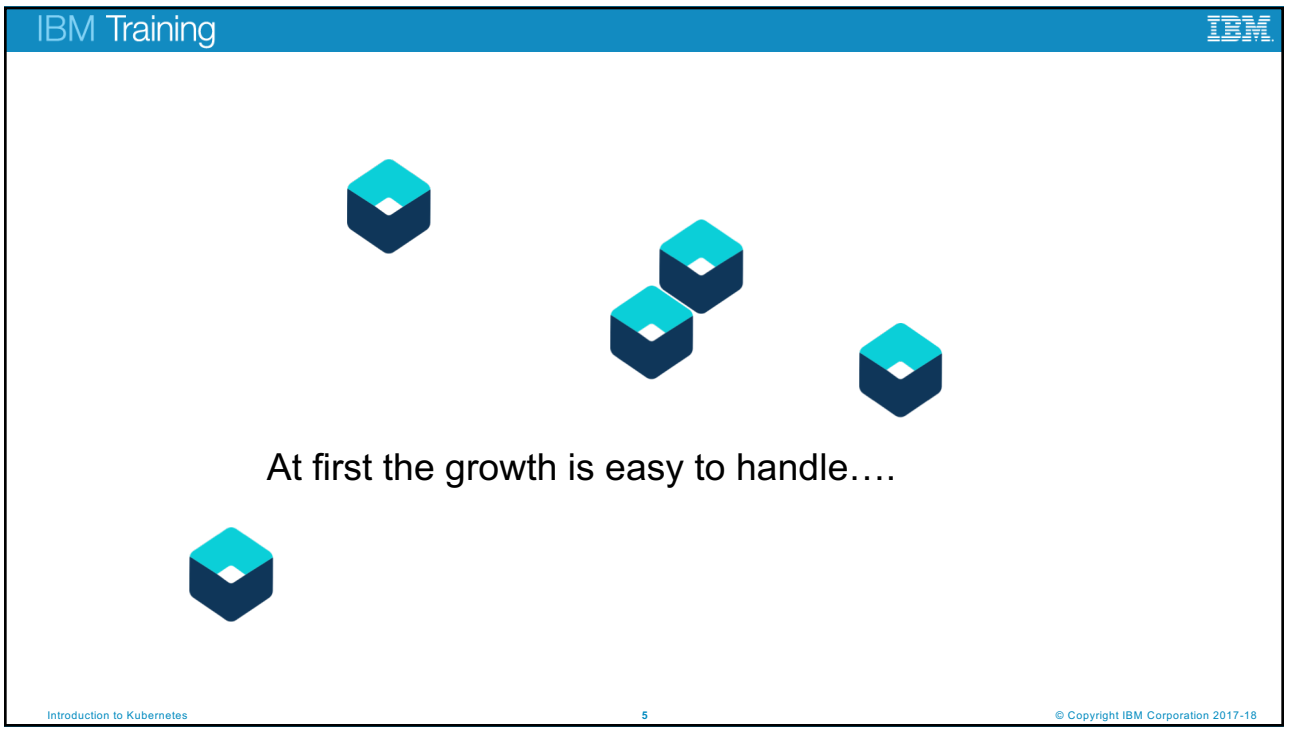
- Introduction
- Container orchestration
- Kubernetes
- Conclusion

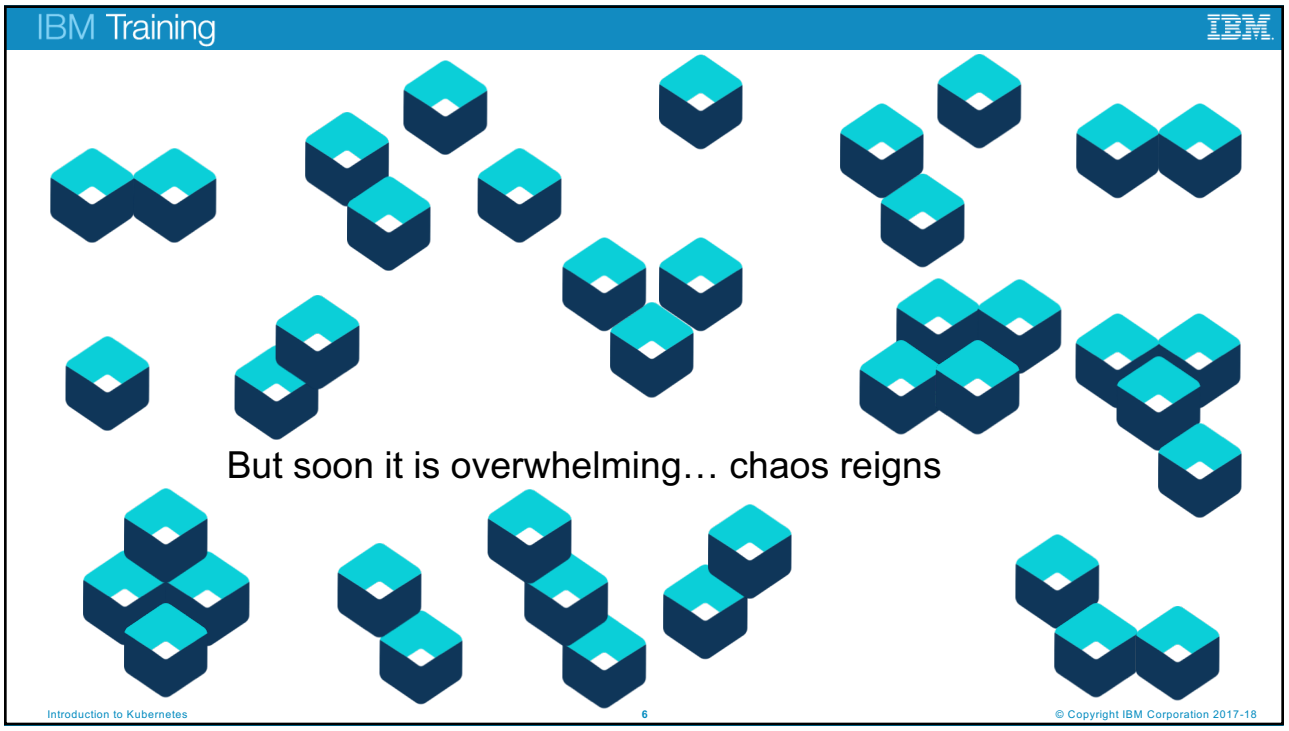
IBM Training

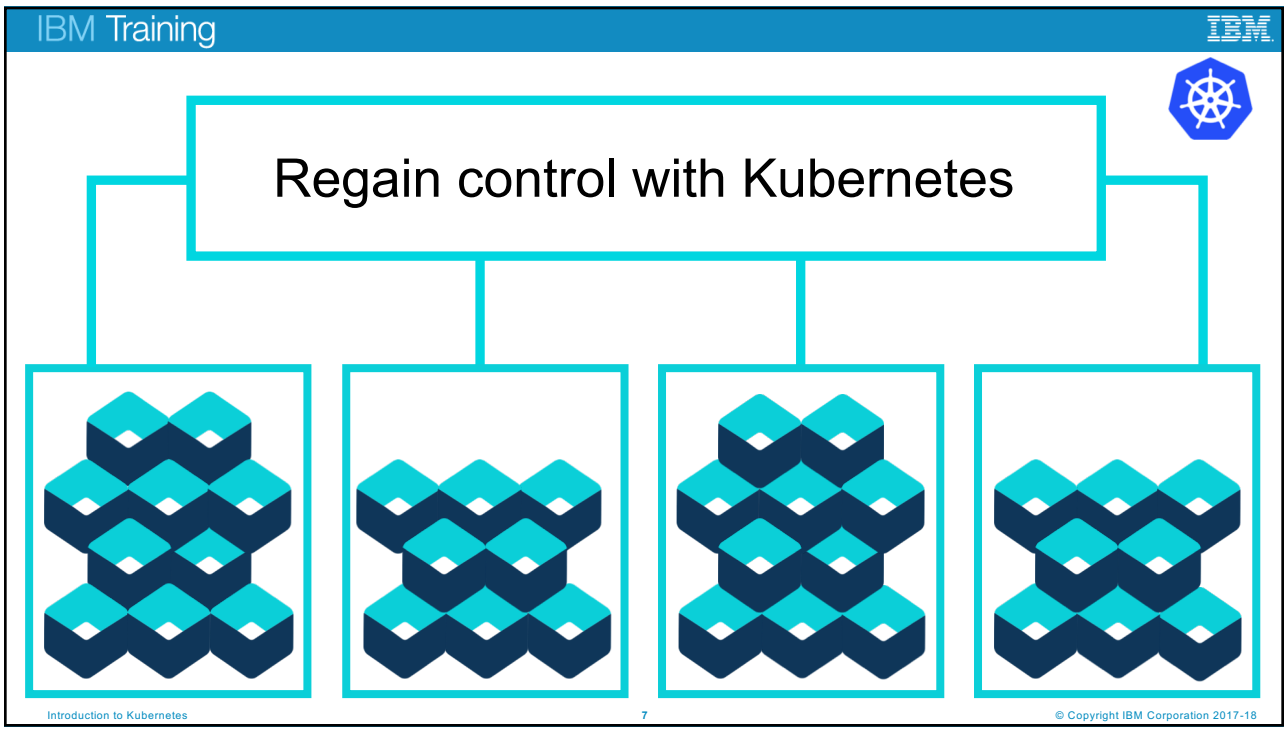
Introduction

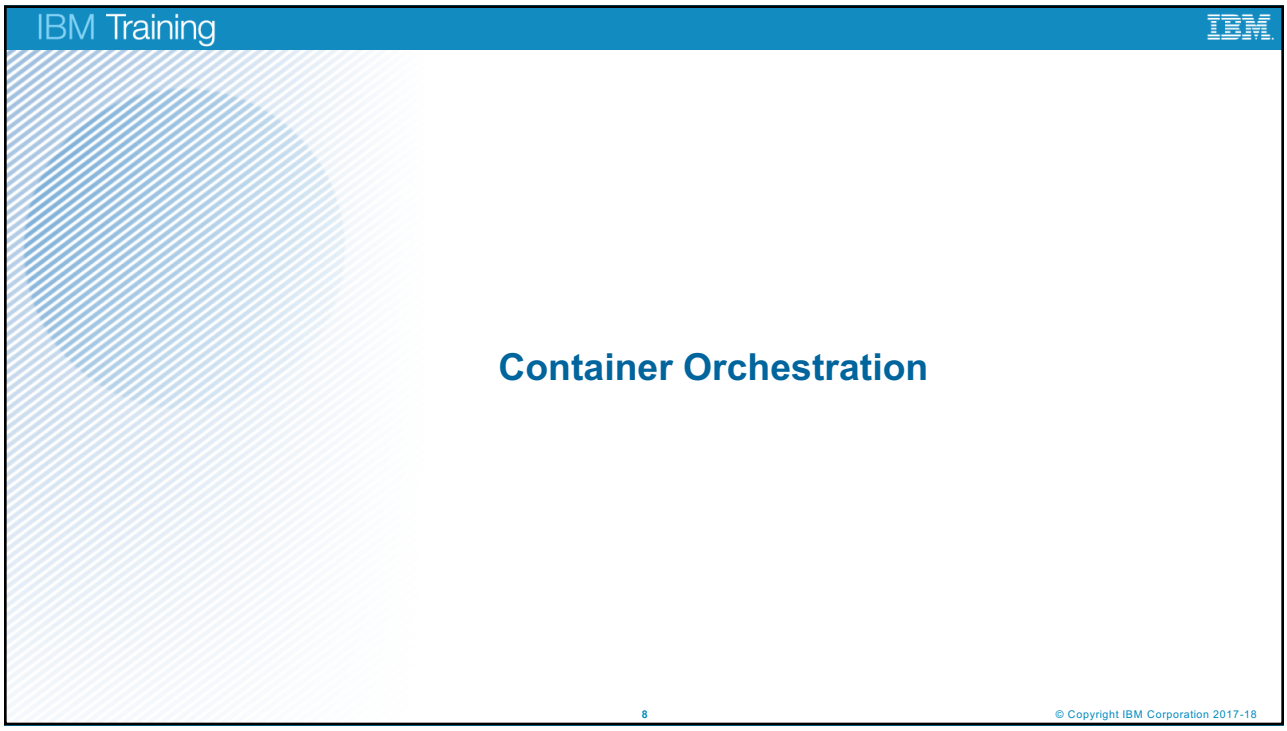
IBM Training

Everyone’s container journey starts with one container….


IBM Training
IBM
At first the growth is easy to handle….
Introduction to Kubernetes
5
© Copyright IBM Corporation 2017-18


IBM Training
IBM
But soon it is overwhelming… chaos reigns
Introduction to Kubernetes
6
© Copyright IBM Corporation 2017-18


IBM Training
IBM
Regain control with Kubernetes
Introduction to Kubernetes
7
© Copyright IBM Corporation 2017-18


IBM Training
IBM
Container Orchestration
8
© Copyright IBM Corporation 2017-18

IBM Training

More to Containers than just Docker

Serverless — APACHE OpenWhisk™

PaaS — CLOUD FOUNDRY

Container Orchestration — Apache MESOS, kubernetes, Swarm/ Swarm Mode

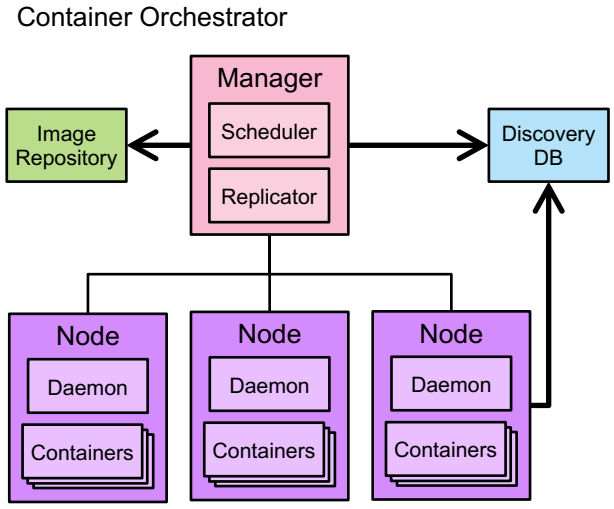
Container Engine — OPEN CONTAINER INITIATIVE, rkt, docker

Introduction to Kubernetes 9 © Copyright IBM Corporation 2017-18

IBM Training

What is container orchestration?

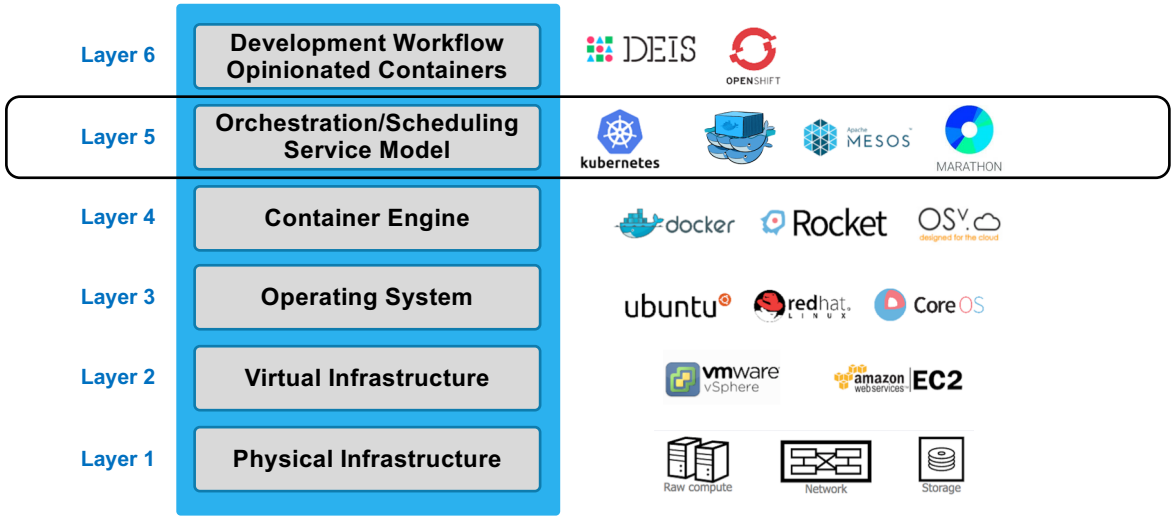
- Container orchestration
  - Manages the deployment, placement, and lifecycle of workload containers
- Cluster management
  - Federates multiple hosts into one target
- Scheduling
  - Distributes containers across nodes
- Service discovery
  - Knows where the containers are located
  - Distributes client requests across the containers
- Replication
  - Ensures the right number of nodes and containers
- Health management
  - Replaces unhealthy containers and nodes



Introduction to Kubernetes 10 © Copyright IBM Corporation 2017-18

IBM Training

Container Ecosystem Layers



IBM Training

Container Orchestration Responsibilities

- Container orchestration
  - Scheduling
  - Cluster management
  - Service discovery
- Related functionality
  - Provisioning
  - Monitoring
  - Configuration management

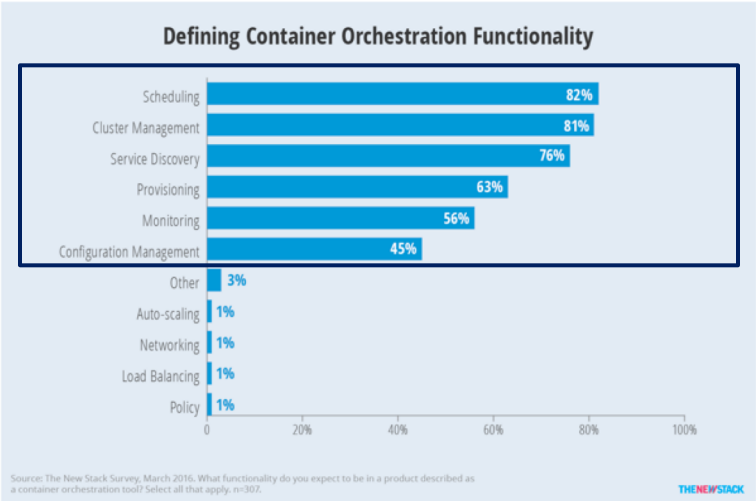


Figure 3: Only 45 percent of respondents consider configuration management to be part of a container orchestration product.

IBM Training

Container Orchestration in IBM Cloud

- In June 2015 when IBM Cloud Container Service launched, open source orchestration projects were not available for production
  - Kubernetes – July 21, 2015
  - Docker Swarm – November 3, 2015
  - Apache Mesos – July 27, 2016

- IBM developed custom orchestration to ensure anti co-location of containers within a group and container placement to the least utilized hosts
- IBM Research runs rigorous performance and scalability testing (https://github.com/Open-I-Beam/containers-os) on various open source projects; then work with those communities to make improvements

Introduction to Kubernetes 13

IBM Training

Kubernetes

14

IBM Training

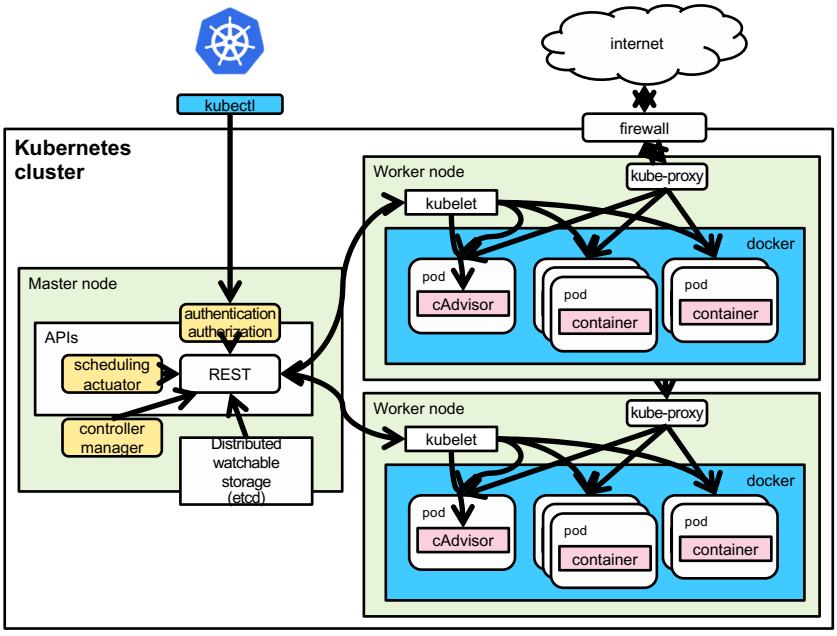
What is Kubernetes?

- Container orchestrator
  - Runs and manages containers
  - Unified API for deploying web applications, batch jobs, and databases
  - Maintains and tracks the global view of the cluster
  - Supports multiple cloud and bare-metal environments
- Manage applications, not machines
  - Rolling updates, canary deploys, and blue-green deployments
- Designed for extensibility
  - Rich ecosystem of plug-ins for scheduling, storage, networking
- Open source project managed by the Linux Foundation
  - Inspired and informed by Google's experiences and internal systems
  - 100% open source, written in Go

IBM Training

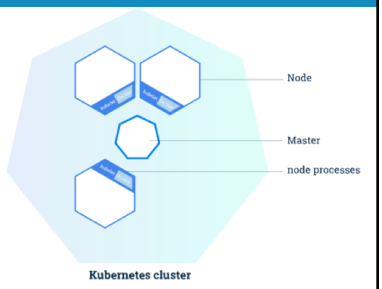
Kubernetes Architecture

- Master nodes
  - Computer with processes that manage the cluster
  - Multiple nodes for HA
- Worker nodes
  - Computers that host pods
  - Pods host application containers



IBM Training

Kubernetes Terminology: Topology



- Cluster
  - Collection of worker nodes managed by the same master node
  - Makes the worker nodes behave like one big computer
- Master node
  - Controls and manages the cluster
  - Scheduling and replication logic
  - Generally two or more master nodes for resiliency, but are not used for scaling out the cluster
- Worker node
  - Node where pods are run
  - Docker – Docker engine for running containers (including the kubelet)
  - kubelet
    - Kubernetes agent that accepts commands from the master
    - Manages pods in the node
  - cAdvisor – Container Advisor provides resources usage and performance statistics
  - kube-proxy – network proxy service responsible for routing activities for inbound or ingress traffic

IBM Training

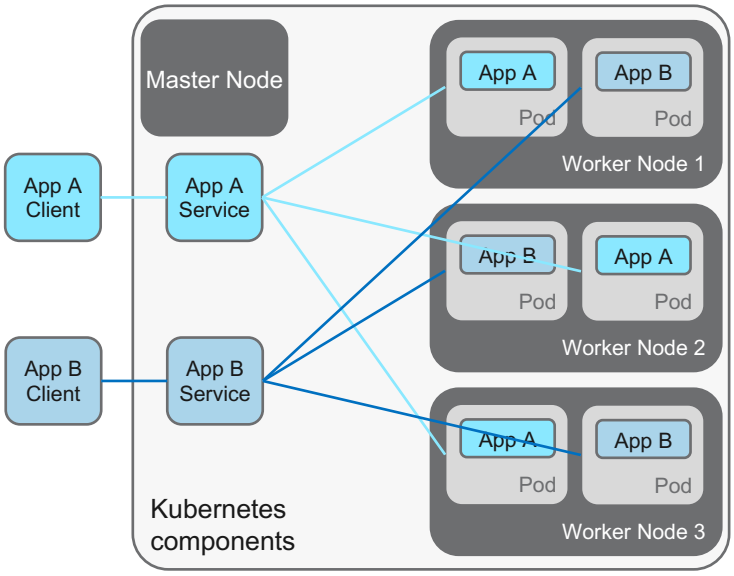
Kubernetes Terminology: Master Node Components

- Etcd
  - A highly-available key value store
  - All cluster data is stored here
- API Server
  - Exposes API for managing Kubernetes
  - Used by kubectrl CLI
- Controller manager
  - Daemon that runs controllers, which are the background threads that handle routine tasks in the cluster
  - Node Controller – Responsible for noticing and responding when nodes go down
  - Replication Controller – Replaced by ReplicaSet
  - Endpoints Controller – Populates the Endpoints object (that is, joins services and pods)
  - Service Account & Token Controllers – Create default accounts and API access tokens for new namespaces
- Scheduler
  - Selects the worker node each pods runs in

IBM Training

Kubernetes Architecture: Workloads

- Container
  - Packaging of an app
- Pod
  - Unit of deployment
- Service
  - Fixed endpoint for 1+ pods



IBM Training

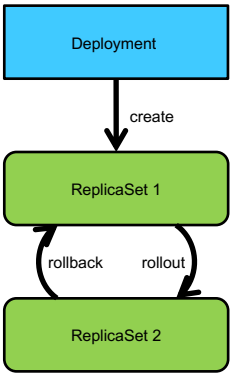
Kubernetes Terminology: Workloads

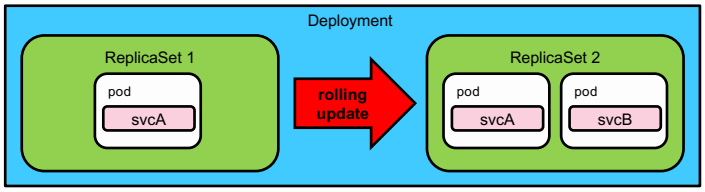
- Container
  - Unit of packaging
- Pod
  - Smallest deployment unit in Kubernetes
  - A collection of containers that run on a worker node
  - Each pod has its own IP
  - A pod shares a PID namespace, network, and hostname
- Service
  - Collection of pods exposed as an endpoint
  - Information stored in the Kubernetes cluster state and networking info propagated to all worker nodes
  - Types
    - ClusterIP – Exposes the service on a cluster-internal IP
    - NodePort – Exposes the service on each Node's IP at a static port
    - LoadBalancer – Exposes the service externally using a cloud provider's load balancer
    - ExternalName – Maps the service to an external name (such as foo.bar.example.com)

IBM Training

Kubernetes Terminology: Deployment

- Deployment
  - A set of pods to be deployed together, such as an application
  - Declarative: Revising a Deployment creates a ReplicaSet describing the desired state
  - Rollout: Deployment controller changes the actual state to the desired state at a controlled rate
  - Rollback: Each Deployment revision can be rolled back
  - Scale and autoscale: A Deployment can be scaled
- ReplicaSet
  - The next-generation ReplicationController
  - A set of pod templates that describe a set of pod replicas
  - Uses a template that describes specifically what each pod should contain
  - Ensures that a specified number of pod replicas are running at any given time





Introduction to Kubernetes 21 © Copyright IBM Corporation 2017-18

IBM Training

Kubernetes Autoscaling

- Horizontal Pod Autoscaler (HPA)
  - Automatically scales the number of pods in a replication controller, deployment, or replica set
  - Matches the observed average CPU utilization to the specified target
  - Fetches metrics in two different ways: direct Heapster access and REST client access
  - Kubernetes Heapster enables container cluster monitoring and performance analysis
  - Default config: query every 30 sec, maintain 10% tolerance, wait 3 min after scale-up, wait 5 min after scale-down

```
$ kubectl autoscale deployment <deployment-name>
  --min=3 --max=10 --cpu-percent=50
```

- Creates a horizontal pod autoscaler
  - An HPA instance that autoscales the specified deployment
  - Maintains between 3 and 10 replicas of the pods controlled by the deployment
  - Maintains an average CPU utilization across all pods of 50%

Introduction to Kubernetes 22 © Copyright IBM Corporation 2017-18

IBM Training

Kubernetes Terminology: Naming

- Name
  - Each resource object by type has a unique name
- Namespace
  - Resource isolation: Each namespace is a virtual cluster within the physical cluster
    - Resource objects are scoped within namespaces
    - Low-level resources are not in namespaces: nodes, persistent volumes, and namespaces themselves
    - Names of resources need to be unique within a namespace, but not across namespaces
  - Resource quotas: Namespaces can divide cluster resources
  - Initial namespaces
    - `default` – The default namespace for objects with no other namespace
    - `kube-system` – The namespace for objects created by the Kubernetes system
- Resource Quota
  - Limits resource consumption per namespace
  - Limit can be number of resource objects by type (pods, services, etc.)
  - Limit can be total amount of compute resources (CPU, memory, etc.)
  - Overcommit is allowed; contention is handled on a first-come, first-served basis

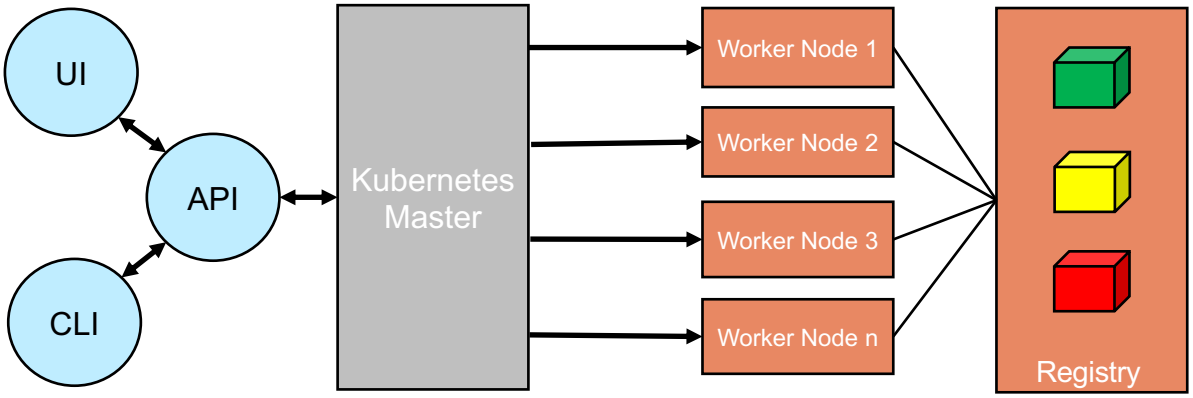
IBM Training

Kubernetes Configuring Resources and Containers

- Label
  - Metadata assigned to Kubernetes resources (pods, services, etc.)
  - Key-value pairs for identification
  - Critical to Kubernetes as it relies on querying the cluster for resources that have certain labels
- Selector
  - An expression that matches labels to identify related resources
- ConfigMap
  - Configuration values to be used by containers in a pod
  - Stores configuration outside of the container image, making containers more reusable
- Secrets
  - Sensitive info that containers need to read or consume
  - Encrypted in special volumes mounted automatically

IBM Training

Kubernetes Architecture



- Etcd
- API Server
- Controller Manager Server
- Scheduler

IBM Training

Kubectl Commands

- Get the state of your cluster
  ```
  $ kubectl cluster-info
  ```
- Get all the nodes of your cluster
  ```
  $ kubectl get nodes -o wide
  ```
- Get info about the pods of your cluster
  ```
  $ kubectl get pods -o wide
  ```
- Get info about the replication controllers of your cluster
  ```
  $ kubectl get rc -o wide
  ```
- Get info about the services of your cluster
  ```
  $ kubectl get services
  ```
- Get full config info about a Service
  ```
  $ kubectl get service
  NAME_OF_SERVICE -o json
  ```
- Get the IP of a Pod
  ```
  $ kubectl get pod NAME_OF_POD -
  template={{.status.podIP}}
  ```
- Delete a Pod
  ```
  $ kubectl delete pod NAME
  ```
- Delete a Service
  ```
  $ kubectl delete service
  NAME_OF_SERVICE
  ```

IBM Training

Conclusion

IBM Training

Conclusion

- Container orchestration
- Kubernetes

IBM Training

Resources

- Kubernetes tutorial
  - https://kubernetes.io/docs/tutorials/kubernetes-basics/
- Introduction to container orchestration
  - https://www.exoscale.ch/syslog/2016/07/26/container-orch/
- TNS Research: The Present State of Container Orchestration
  - https://thenewstack.io/tns-research-present-state-container-orchestration/
- Large-scale cluster management at Google with Borg
  - https://research.google.com/pubs/pub43438.html

IBM Training

Kubernetes in IBM Cloud Private

IBM Training

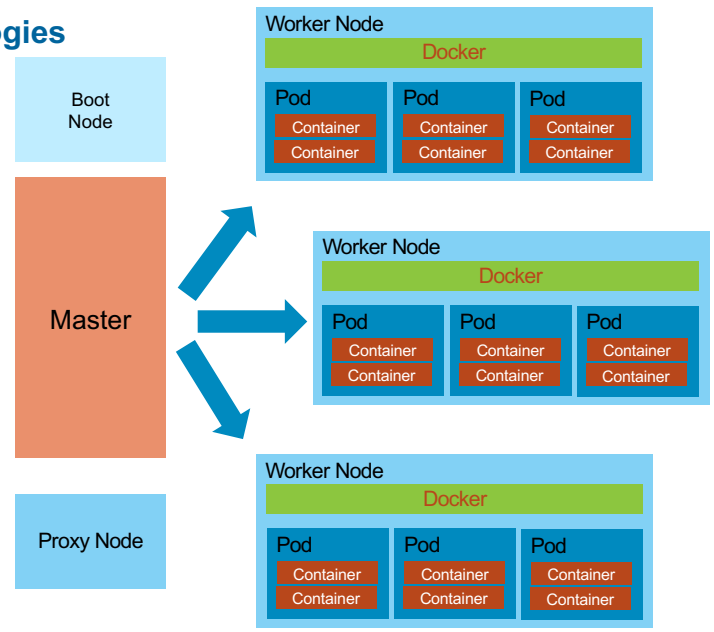
Objectives

- Explain IBM Cloud Private implementation of Kubernetes
- List major components of IBM Cloud Private implementation of Kubernetes (calico, authentication, user interface, etc)

IBM Training

Kubernetes Configuration topologies

- Simple
  - Single machine install (master is a worker)
  - Great for testing and learning about the platform
- Standard
  - Single master (single master, 3 workers, 1 proxy)
  - Great for non-production testing environment
- High Availability
  - Multiple masters (3 masters, 3+ workers, 3 proxy)
  - Production installation



IBM Training

IBM Cloud Private node types

- **Boot node**: A boot or bootstrap node is used for running installation, configuration, node scaling, and cluster updates. Only one boot node is required for any cluster. You can use a single node for both master and boot.
- **Master node**: A master node provides management services and controls the worker nodes in a cluster. Master nodes host processes that are responsible for resource allocation, state maintenance, scheduling, and monitoring. Multiple master nodes are in a high availability (HA) environment to allow for failover if the leading master host fails. Host that can act as the master are called master candidates.
- **Worker node**: A worker node is a node that provides a containerized environment for running tasks. As demands increase, more worker nodes can easily be added to your cluster to improve performance and efficiency. A cluster can contain any number of worker nodes, but a minimum of one worker node is required.
- **Proxy Node**: A proxy node is a node that transmits external request to the services created inside your cluster. Multiple proxy nodes are deployed in a high availability (HA) environment to allow for failover if the leading proxy host fails. While you can use a single node as both master and proxy, it is best to use dedicated proxy nodes to reduce the load on the master node. A cluster must contain at least one proxy node if load balancing is required inside the cluster.

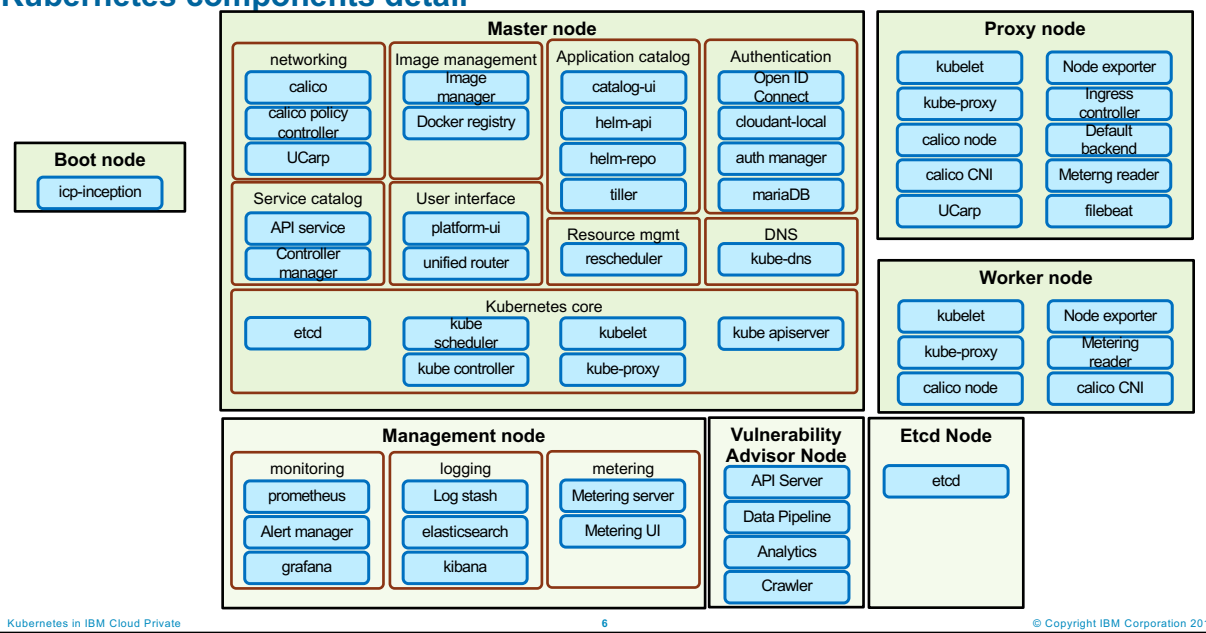
IBM Training

IBM Cloud Private node types – optional nodes

- **Management node**: A management node is an optional node that only hosts management services such as monitoring, metering, and logging. By configuring dedicated management nodes, you can prevent the master node from becoming overloaded. You can enable the management node only during IBM Cloud Private installation.
- **VA node**: A VA (Vulnerability Advisor) node is an optional node that is used for running the Vulnerability Advisor services. Vulnerability Advisor services are resource intensive. If you use the Vulnerability Advisor service, specify a dedicated VA node.
- **Etcd node**: An etcd node is an optional node that is used for running the etcd distributed key value store. Configuring an etcd node in an IBM Cloud Private cluster that has many nodes, such as 100 or more, helps to improve the etcd performance.

IBM Training

Kubernetes components detail



IBM Training

Kubernetes master components

Master components provide the cluster's control plane and global decisions about the cluster and detecting and responding to cluster events

- **Etcd**: A strong, consistent, and highly-available key value store which Kubernetes uses for persistent storage of all of its API objects.
- **Kubelet**: The primary "node agent" that runs on each node.
- **Kube Proxy**: The Kubernetes network proxy runs on each node.
- **Kube Scheduler**: A policy-rich, topology-aware, workload-specific function that significantly impacts availability, performance, and capacity.
- **Kube Control Manager**: A daemon that embeds the core control loops shipped with K8s. In K8s, a controller is a control loop that watches the shared state of the cluster through the apiserver and makes changes attempting to move the current state towards the desired state.
- **Kube apiserver**: Validates and configures data for the api objects which include pods, services, replication controllers, and others. The API Server services REST operations and provides the frontend to the cluster's shared state through which all other components interact.

IBM Training

Network services components

- **DNS**: (kube-dns, Cluster DNS) K8s DNS schedules a DNS pod and service on the cluster, and configures the kubelets to tell individual containers to use the DNS service's IP to resolve DNS names. Every service defined in the cluster (including the DNS server itself) is assigned a DNS name. By default, a client pod's DNS search list will include the pod's own namespace and the cluster's default domain.
- **UCarp**: UCarp allows a couple of hosts to share common virtual IP (or floating IP) addresses in order to provide automatic failover.
- **Calico**: A new approach to virtual networking and network security for containers, VMs, and bare metal services, that provides a rich set of security enforcement capabilities running on top of a highly scalable and efficient virtual network
  - The calico/node Docker container runs on the Kubernetes master and each Kubernetes node in the cluster
  - The calico-cni plug-in integrates directly with the Kubernetes kubelet process on each node to discover which pods have been created, and adds them to Calico networking
  - The calico/kube-policy-controller container runs as a pod on top of Kubernetes and implements the NetworkPolicy API

IBM Training

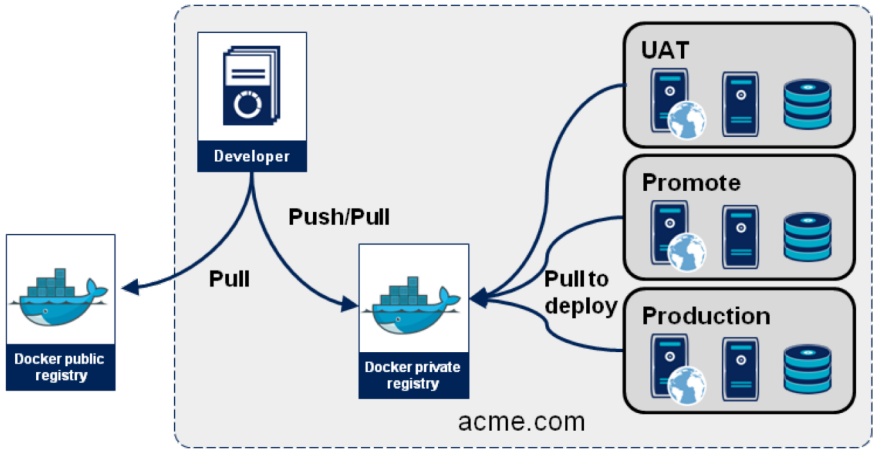
Authentication

- **Authentication Manager:** Provides an HTTP API for managing users. Protocols are implemented in a RESTful manner. Open ID Connect is used for authentication.
- **Open ID Connect**: WebSphere Liberty based authentication server that implements Open ID Connect protocol.
- **MariaDB:** An open source relational database made by the original developers of MySQL.

IBM Training

Images and Registries

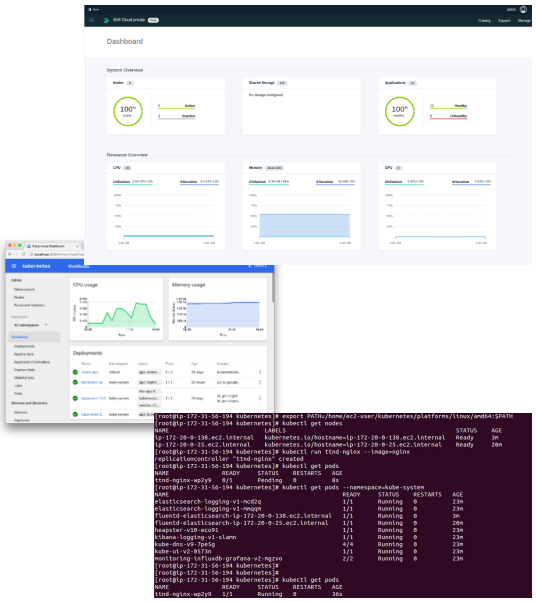
- You create a Docker image and push it to a registry before referring to it in a Kubernetes pod
- There will likely be many registries used in your deployment



IBM Training

User Interfaces

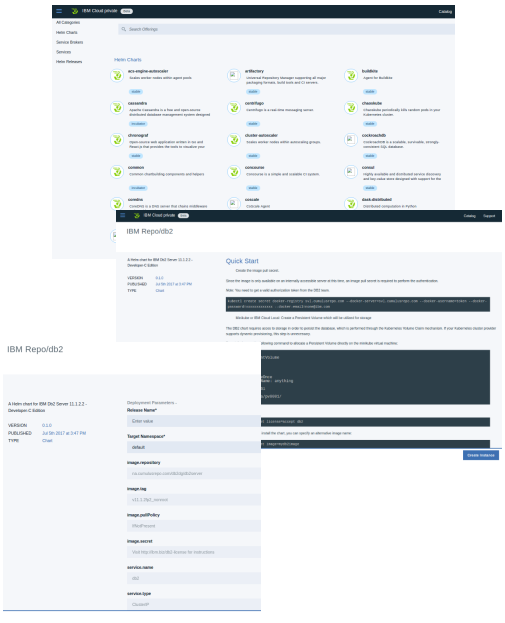
- Cluster Management Console: (ICP component) Use to manage, monitor, and troubleshoot your applications and cluster from a single, centralized, and secure management console.
- **Platform-UI**: Can use to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized application, and manage the cluster itself along with its attendant resources.
- **kubectl**: A command-line interface for running commands against Kubernetes clusters.



IBM Training

Application Center components

- **Catalog-UI**: provides a centralized location from which you can browse, and install packages in your cluster.
- **Helm**: A tool for managing Kubernetes charts. Charts are packages of pre-configured Kubernetes resources.
- **Helm Repository**: A Helm chart repository is a location where packaged charts can be stored and shared.
- **Tiller**: Runs inside of the cluster, and manages releases (installations) of your charts.



IBM Training

Ingress resources

- **Ingress**: Typically, services and pods have IPs only routable by the cluster network. All traffic that ends up at an edge router is either dropped or forwarded elsewhere.
  - An Ingress is a collection of rules that allow inbound connections to reach the cluster services.
  - It can be configured to give services externally-reachable URLs, load balance traffic, terminate SSL, offer name based virtual hosting etc.
  - Users request ingress by POSTing the Ingress resource to the API server
- **Ingress Controller**: Responsible for fulfilling the Ingress, usually with a load balancer, though it may also configure your edge router or additional frontends to help handle the traffic in an HA manner.

IBM Training

Logging components

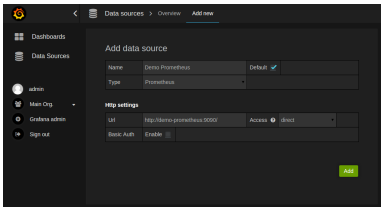
The easiest and most embraced logging method for containerized applications is to write to standard out and standard error

- **Filebeat**: A log data shipper for local files. Filebeat monitors the log directories or specific log files, tails the files, and forwards them either to Elasticsearch and/or Logstash for indexing.
- **Elasticsearch**: An open source full-text search engine based on Lucene. It provides HTTP web interface and schema-free JSON documents.
- **Log stash**: A open source tool for collecting, parsing, and storing logs for future use.
- **Heapster**: The Kubernetes network proxy runs on each node.
- **Kibana**: An open source data visualization plugin for Elasticsearch. Users can create bar, line and scatter plots, or pie charts and maps on top of large volumes of data.

IBM Training

Monitoring: Prometheus and Grafana

- **Prometheus**: An open-source systems monitoring and alerting toolkit originally built at SoundCloud. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. It is now a standalone open source project and maintained independently of any company.
- **Grafana**: An open-source, general purpose dashboard and graph composer, which runs as a web application.



Kubernetes in IBM Cloud Private 15

IBM Training

Persistent storage components

- Traditionally Containers: stateless, ephemeral in nature
  - Storage exists within the container
  - The container goes away and so goes the storage
- Some applications desire state and thus persistent storage:
  - Specific aspects of configuration
  - Database (structured and unstructured)
  - Application data (website definitions, etc.)
- Storage must be universally accessible across the K8s environment
- ICP Persistent Storage Support: HostPath, NFS, GlusterFS, vSphereVolume
  - hNote: Reclaim policy and access modes and behaviors can vary
- Access Modes:
  - ReadWriteOnce – the volume can be mounted as read-write by a single node
  - ReadOnlyMany – the volume can be mounted read-only by many nodes
  - ReadWriteMany – the volume can be mounted as read-write by many nodes

Kubernetes in IBM Cloud Private 16

IBM Training IBM

Introduction to Helm

kubernetes

IBM Training IBM

Agenda

- What is Helm?
- Using Helm
- Developing charts
- Developing templates
- Conclusion
- Resources

IBM Training

What is Helm?

IBM Training

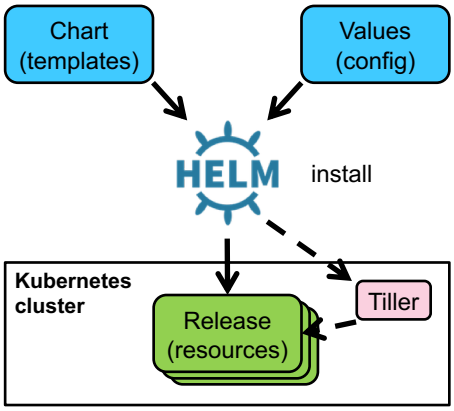
Helm – A package manager for Kubernetes

- What is a package manager?
  - Automates the process of installing, configuring, upgrading, and removing computer programs
  - Examples: Red Hat Package Manager (RPM), Homebrew (macOS), Chocolatey (Windows)
- Helm enables multiple Kubernetes resources to be created with a single command
  - Deploying an application often involves creating and configuring multiple resources
  - A Helm chart defines multiple resources as a set
- An application in Kubernetes typically consists of (at least) two resource types
  - Deployment – Describes a set of pods to be deployed together
  - Services – Endpoints for accessing the APIs in those pods
  - Could also include ConfigMaps, Secrets, Ingress, etc.
- A default chart for an application consists of a deployment template and a service template
  - The chart creates all of these resources in a Kubernetes cluster as a set
  - Rather than manually having to create each one separately via `kubectl`

IBM Training

Helm Terminology

- Helm – The CLI
  - Helm installs charts into Kubernetes, creating a new release for each installation
  - To find new charts, search Helm chart repositories
- Chart – The application package
  - Templates for a set of resources necessary to run an application
  - The chart includes a values file that configures the resources
- Repository – The library
  - Storage for Helm charts
  - `stable` – The namespace of the hub for official charts
- Release – The application runtime
  - An instance of a chart running in a Kubernetes cluster
  - The same chart installed multiple times creates many releases
- Tiller – The server-side engine
  - Helm templating engine, runs in a pod in a Kubernetes cluster
  - Tiller processes the chart to generate the resource manifests, then installs the release into the cluster
  - Tiller stores each release as a Kubernetes config map



Introduction to Helm 5 © Copyright IBM Corporation 2017

IBM Training

Advantages of Using Helm

- Deploy all of the resources for an application with a single command
  - Makes deployment easy and repeatable

```
$ helm install <chart>
```

- Separates configuration settings from manifest formats
  - Edit the values without changing the rest of the manifest
  - `values.yaml` – Update to deploy the application differently
- Upgrade a running release to a new chart version

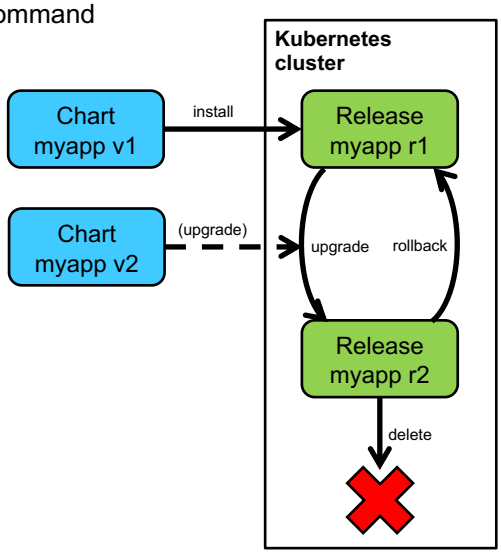
```
$ helm upgrade <release> <chart>
```

- Rollback a running release to a previous revision

```
$ helm rollback <release> <revision>
```

- Delete a running release

```
$ helm delete <release>
```



Introduction to Helm 6 © Copyright IBM Corporation 2017

IBM Training

Installing Helm

- Helm runs as a CLI client
  - Typically installed on your laptop
- See Installing Helm
  - https://docs.helm.sh/using_helm/#installing-helm
- Options for installing Helm
  1. Download the release, including the binary
     - https://github.com/kubernetes/helm/releases
  2. Homebrew on MacOS
     - `brew install kubernetes-helm`
  3. Installer script
     - `curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get > get_helm.sh`

Introduction to Helm 7

IBM Training

Using Helm

8

IBM Training

Helm Commands

- Install Tiller
  `$ helm init`
- Create a chart
  `$ helm create <chart>`
- List the repositories
  `$ helm repo list`
- Search for a chart
  `$ helm search <keyword>`
- Info about a chart
  `$ helm inspect <chart>`
- Deploy a chart (creates a release)
  `$ helm install <chart>`
- List all releases
  `$ helm list --all`
- Get the status of a release
  `$ helm status <release>`
- Get the details about a release
  `$ helm get <release>`
- Upgrade a release
  `$ helm upgrade <release> <chart>`
- Rollback a release
  `$ helm rollback <release> <revision>`
- Delete a release
  `$ helm delete <release>`

Introduction to Helm 9

IBM Training

Working with Repositories

```
$ helm repo list
NAME        URL
stable      https://kubernetes-charts.storage.googleapis.com/

$ helm search jenkins
NAME              VERSION     DESCRIPTION
stable/jenkins    0.1.14      A Jenkins Helm chart for Kubernetes.

$ helm repo add my-charts https://my-charts.storage.googleapis.com
$ helm repo list
NAME        URL
stable      https://kubernetes-charts.storage.googleapis.com/
my-charts   https://my-charts.storage.googleapis.com
```

Introduction to Helm 10

IBM Training

Installing an Application

- To deploy an application into Kubernetes, install that application's Helm chart

```
$ helm search mysql
NAME                VERSION   DESCRIPTION
stable/mysql        0.1.1     Chart for MySQL
$ helm install stable/mysql
Fetched stable/mysql to mysql-0.1.1.tgz
NAME: loping-toad
LAST DEPLOYED: Thu Oct 20 14:54:24 2016
NAMESPACE: default
STATUS: DEPLOYED
RESOURCES:
==> v1/Secret
NAME                TYPE      DATA      AGE
loping-toad-mysql   Opaque    2         3s
==> v1/Service
NAME                CLUSTER-IP          EXTERNAL-IP         PORT(S)   AGE
loping-toad-mysql   192.168.1.5         <none>              3306/TCP  3s
==> extensions/Deployment
NAME                DESIRED   CURRENT   UP-TO-DATE          AVAILABLE AGE
loping-toad-mysql   1         0         0                   0         3s
==> v1/PersistentVolumeClaim
NAME                STATUS    VOLUME    CAPACITY  ACCESSMODES         AGE
loping-toad-mysql   Pending
```

- Install output
  - Details about the release
  - Details about its resources
- Chart
  - `stable/mysql`
- Release name
  - `loping-toad` (auto generated)
- Resources
  - Four total, one of each type
  - All named `loping-toad-mysql`
  - `Secret`
  - `Service`
  - `Deployment`
  - `PersistentVolumeClaim`

Introduction to Helm 11

IBM Training

Overriding Values

- Default values are stored in the chart
  `<chart-path>/values.yaml`
- Helm CLI uses Kubernetes CLI's configuration to connect to your current cluster
  `~/.kube/config`
  `$ kubectl config view`
- To specify a release's name, use the *name* flag
  `$ helm install --name CustomerDB stable/mysql`
- To deploy the release into a particular Kubernetes namespace, use the *namespace* flag
  `$ helm install --namespace ordering-system stable/mysql`
- To override an individual value, use the *set* flag
  `$ helm install --set user.name='student',user.password='passw0rd' stable/mysql`
- To override values with a values file, use the *values* or *f* flag
  `$ helm install --values myvalues.yaml stable/mysql`

Introduction to Helm 12

IBM Training IBM

Developing Charts

13

IBM Training IBM

Creating a Chart

- Creating a new chart generates a directory with sample files

```
$ helm create my-chart
$ tree my-chart
```

```
my-chart/                   # The content of this directory is the chart
    |- Chart.yaml           # Information about the chart
    |- values.yaml          # The default configuration values for this chart
    |- charts/              # Charts that this chart depends on
    |- templates/           # This chart's template files
        |- NOTES.txt        # OPTIONAL: A plain text file containing short usage notes
        |- _helpers.tpl     # OPTIONAL: The default location for template partials
        |- deployment.yaml  # Sample template for a deployment resource
        |- service.yaml     # Sample template for a service resource
```

- By default, a chart starts with sample templates for a Kubernetes deployment and service
  - In the simplest case, just edit the `values.yaml` file

IBM Training

How Install Uses Charts

- The main step of installing a chart is rendering its templates
- How Helm installs a chart
  1. User runs an install in the Helm CLI
     ```
     $ helm install myapp
     ```
  2. Helm CLI loads the chart into Tiller
  3. **Tiller renders the chart templates**
  4. Tiller loads the resulting resources into Kubernetes
  5. Tiller returns the release data to the client
  6. The client exits
- Rendering the templates
  - Each template generates a Kubernetes resource manifest file (yaml)
  - Tiller runs each of the template files, generating the resource files
- Tiller then loads the resources—as described by the manifests—into the Kubernetes cluster

IBM Training

Chart Lifecycle Hooks

Hooks

- **pre-install**
  - Executes after templates are rendered
  - Before any resources are created in Kubernetes
- **post-install**
  - Executes after all resources are loaded into Kubernetes
- **pre-delete**
  - Executes before any resources are deleted from Kubernetes
- **post-delete**
  - Executes after all of the release's resources have been deleted
- **pre-upgrade**
  - Executes after templates are rendered
  - Before any resources are loaded into Kubernetes
- **post-upgrade**
  - Executes after all resources have been upgraded
- **pre-rollback**
  - Executes after templates are rendered
  - Before any resources have been rolled back
- **post-rollback**
  - Executes after all resources have been modified

Hooks in the Helm Install Lifecycle

1. User runs an install in the Helm CLI
2. Helm CLI loads the chart into Tiller
3. Tiller renders the chart templates
4. **Tiller executes the pre-install hooks**
5. Tiller loads the resulting resources into Kubernetes
6. **Tiller executes the post-install hook**
7. Tiller returns the release data to the client
8. The client exits

- A hook can be any Kubernetes resource
  - A hook is often a Kubernetes job
  - Goes in the `templates` directory

IBM Training

Sharing Charts

- A chart is a directory
  - Easy for a Helm client to use the chart directories on the same computer
  - Difficult to share with other users on other computers
- Packaging a chart
  - Bundle `Chart.yaml` and related files into a tar file

```
$ helm package <chart-path>          # Bundles chart directory into a tar file
$ helm install <chart-name>.tgz      # Installs the chart in the chart file
```

- Chart repository
  - HTTP server that houses an `index.yaml` file and optionally some packaged charts
  - Server can be any HTTP server that can serve YAML and tar files and can answer GET requests
    - Ex: Google Cloud Storage (GCS) bucket, Amazon S3 bucket, Github Pages, or even create your own web server
  - To add a chart to the repository, copy it to the directory and regenerate the index

```
$ helm repo index <charts-path>      # Generates an index of the charts in the repo
```

Introduction to Helm 17

IBM Training

Developing Templates

18

IBM Training

Creating Templates

- The main aspect of implementing a chart is implementing its templates
- A related task: Create and populate the files that contain the settings used by the templates
  - These settings files, particularly `values.yaml`, define the chart's API
  - The settings files list the variables the templates can use, therefore the only values worth changing
- Examples of chart templates can be found in https://github.com/kubernetes/charts/
  - Each file is a Golang template
  - Includes functions from the Sprig template library
  - A template can create the manifest for any type of Kubernetes resource
- Each file in a chart's `templates` directory is expected to be a template
  - Expected to generate a Kubernetes resource manifest
  - Filename can be anything, should describe the resource it defines
  - Exception: The notes file (i.e. `NOTES.txt`) provides instructions to the chart's users
  - Exception: Files whose names begin with an underscore (e.g. `_helpers.tpl`) are expected to contain partials

Introduction to Helm 19

IBM Training

Chart Template for Deployment Manifest

Kubernetes Deployment Manifest

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
           - containerPort: 80
```

Helm Deployment Template

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
{{- if .Values.podAnnotations }}
      annotations:
{{ toYaml .Values.podAnnotations | indent 8 }}
{{- end }}
      labels:
        app: {{ template "name" . }}
        release: {{ .Release.Name }}
    spec:
      containers:
        - name: {{ template "name" . }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
          - name: http
            containerPort: 80
            protocol: TCP
. . .
```

Introduction to Helm 20

IBM Training

Chart Template for Service Manifest

Kubernetes Service Manifest

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Helm Service Template

```
apiVersion: v1
kind: Service
metadata:
{{- if .Values.service.annotations }}
  annotations:
{{ toYaml .Values.service.annotations | indent 4 }}
{{- end }}
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
spec:
  selector:
    app: {{ template "name" . }}
    release: {{ .Release.Name }}
  ports:
    - name: http
      protocol: TCP
      port: {{ .Values.service.port }}
      targetPort: http
      {{- if (and (eq .Values.service.type "NodePort") ...) }}
      nodePort: {{ .Values.service.nodePort }}
      {{- end }}
. . .
```

Introduction to Helm 21

IBM Training

Values YAML – A Chart's API

Values (`values.yaml`)

```
replicaCount: 1
restartPolicy: Never
# Evaluated by the post-install hook
sleepyTime: "10"
index: >-
  <h1>Hello</h1>
  <p>This is a test</p>
image:
  repository: nginx
  tag: 1.11.0
  pullPolicy: IfNotPresent
service:
  annotations: {}
  clusterIP: ""
  externalIPs: []
  loadBalancerIP: ""
  loadBalancerSourceRanges: []
  type: ClusterIP
  port: 8888
  nodePort: ""
podAnnotations: {}
resources: {}
nodeSelector: {}
```

Helm Deployment Template

```
. . .
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
{{- if .Values.podAnnotations }}
      annotations:
{{ toYaml .Values.podAnnotations | indent 8 }}
{{- end }}
. . .
```

Helm Service Template

```
. . .
spec:
  ports:
    - name: http
      protocol: TCP
      port: {{ .Values.service.port }}
      targetPort: http
      {{- if (and (eq .Values.service.type "NodePort") ...) }}
      nodePort: {{ .Values.service.nodePort }}
      {{- end }}
. . .
```

Introduction to Helm 22

IBM Training

Chart YAML – A Chart's Meta Information

Chart (`Chart.yaml`)

```
name: nginx
description: A basic NGINX HTTP server
version: 0.1.0
keywords:
  - http
  - nginx
  - www
  - web
home: https://github.com/kubernetes/helm
sources:
  - https://hub.docker.com/_/nginx/
maintainers:
  - name: technosophos
    email: mbutcher@deis.com
```

Helm Template

```
. . .
metadata:
{{- if .Values.service.annotations }}
  annotations:
{{ toYaml .Values.service.annotations | indent 4 }}
{{- end }}
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
. . .
```


IBM Training

Chart Template Helpers – More Default Settings

Helpers (`templates/_helpers.tpl`)

```
{{/* vim: set filetype=mustache: */}}
{{/* Expand the name of the chart. */}}
{{- define "name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" -}}
{{- end -}}
{{/* Create a default fully qualified app name. We truncate at 63 chars because . . . */}}
{{- define "fullname" -}}
{{- $name := default .Chart.Name .Values.nameOverride -}}
{{- printf "%s-%s" .Release.Name $name | trunc 63 | trimSuffix "-" -}}
{{- end -}}
```

Helm Template

```
. . .
metadata:
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
. . .
```

IBM Training

Chart Predefined Values – More Default Settings

Predefined Values

- `Release` – Information about the release being created
  - `Release.Name` – The name of the release (not the chart)
  - `Release.Service` – The service that conducted the release
    - Usually this is Tiller
  - `Release.Revision` – The revision number
    - It begins at 1, and increments with each helm upgrade
  - Lots of other Release values
- `Chart` – The contents of the `Chart.yaml`
  - `Chart.Name` – The chart name
  - `Chart.Version` – The chart version
  - `Chart.Maintainers` – The maintainers
  - Etc.
- `Files` – Map of all non-special files in the chart
- `Capabilities` – Map of info about Kubernetes and Helm
  - `Capabilities.KubeVersion` – Version of Kubernetes
  - `Capabilities.TillerVersion` – Version of Tiller
  - `Capabilities.APIVersions` – Kubernetes API versions
- `Template` – Information about the current template

Helm Template

```
. . .
metadata:
{{- if .Values.service.annotations }}
  annotations:
{{ toYaml .Values.service.annotations | indent 4 }}
{{- end }}
  name: {{ template "fullname" . }}
  labels:
    app: {{ template "name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version }}
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
. . .
```

Introduction to Helm 25

IBM Training

Conclusion

26

IBM Training

Conclusion

- What is Helm?
- Using Helm
- Developing charts
- Developing templates

IBM Training

Resources – Introduction

- Helm - The Kubernetes Package Manager
  - https://helm.sh
  - https://docs.helm.sh
  - https://github.com/kubernetes/helm
  - https://github.com/kubernetes/helm/blob/master/docs/index.md
- Taking the Helm: Delivering Kubernetes-Native Applications by Michelle Noorali (KubeCon 2016)
  - https://www.youtube.com/watch?v=zBc1goRfk3k
- Installing Helm
  - https://docs.helm.sh/using_helm/#installing-helm

IBM Training

Resources – Developing Charts

- Helm examples
  - https://github.com/kubernetes/helm/tree/master/docs/examples
- Stable Helm charts
  - https://github.com/kubernetes/charts/tree/master/stable
- Golang templates
  - https://golang.org/pkg/text/template
- Sprig template library
  - https://godoc.org/github.com/Masterminds/sprig
- Getting Started Authoring Helm Charts
  - https://deis.com/blog/2016/getting-started-authoring-helm-charts
- How to Create Your First Helm Chart
  - https://docs.bitnami.com/kubernetes/how-to/create-your-first-helm-chart
- Packaged Kubernetes Deployments – Writing a Helm Chart
  - https://www.influxdata.com/packaged-kubernetes-deployments-writing-helm-chart

IBM Training

Helm in IBM Cloud Private

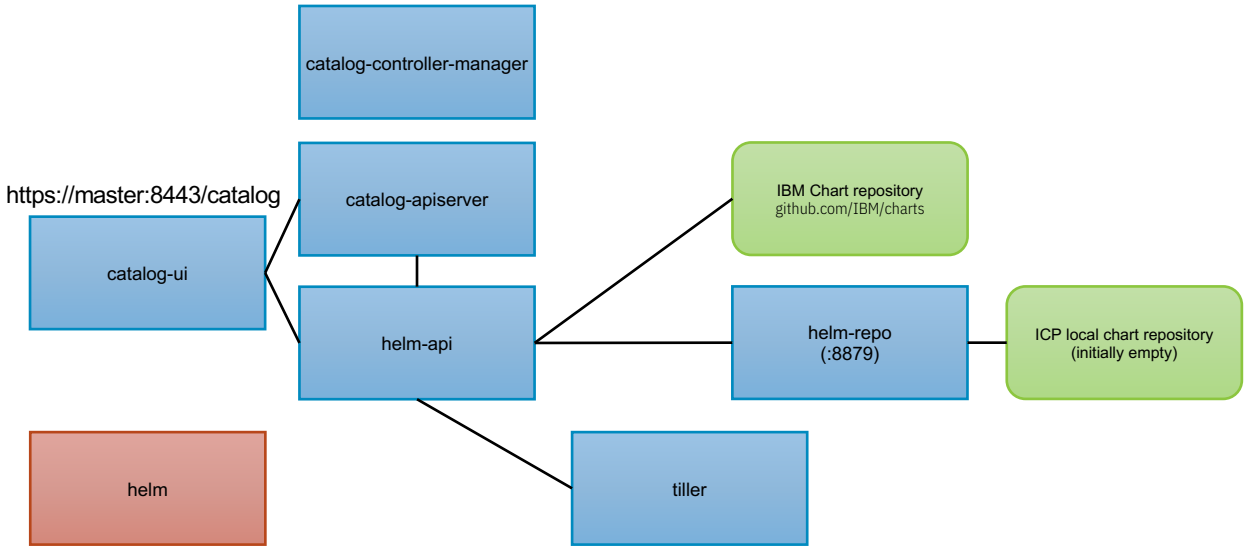
IBM Training

Objectives

- List IBM Cloud Private catalog feature and usage

IBM Training

Catalog in IBM Cloud Private



IBM Training

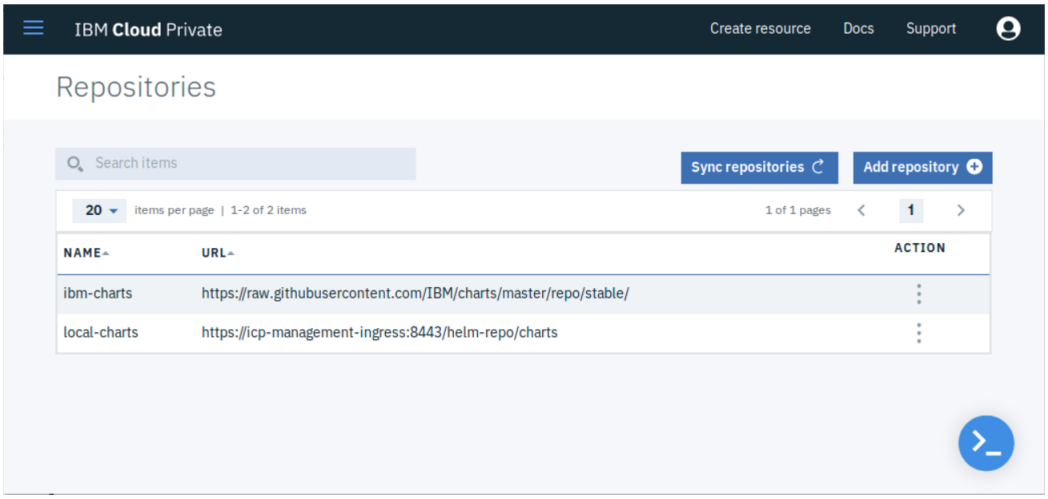
Initialize Helm CLI access

- Use the bx pr plugin
  `bx plugin install icp-linux-amd64`
- Login to Cloud Private:
  `bx pr login -a https://<master>:8443`
- Setup helm client directories:
  `helm init --client-only`
- Setup helm certificates:
  `bx pr cluster-config <cluster-name>`
- Check connectivity to tiller:
  `helm version --tls`

IBM Training

Helm repositories

- **Manage > Helm Repositories**



IBM Training

Catalog entries

Toolchain & Runtimes
- UrbanCode Deploy
- Microclimate
- Microservice Builder
- Jenkins (open source)
- IBM WebSphere Liberty (MicroProfile, Web Profile, JEE Profile)
- Open Liberty (open source)
- IBM SDK for Node.js
- Swift runtime (open source)
- Nginx (open source)

Logging & Monitoring Services
- ELK (open source)
- Prometheus & Grafana (open source)

App Modernization Tooling
- IBM Transformation Advisor

Multi-cloud Management
- IBM Cloud Automation Manager

Digital Business Automation
- IBM Operational Decision Manager
- IBM Operational Decision Manager for Developers

Mobile
- IBM Mobile Foundation

Data Services
- IBM Db2 Direct Advanced Edition / AESE with Data Server Manager
- IBM Db2 Dev-C
- IBM Data Server Manager (for Db2 Dev-C)
- IBM Db2 Warehouse Enterprise
- IBM Db2 Warehouse Dev-C
- IBM Cloudant Developer Edition
- MongoDB (open source)
- PostreSQL (open source)
- MariaDB (open source)
- Galera clustering with MariaDB (open source)
- Redis HA Topology (open source)

Messaging
- IBM MQ Advanced
- IBM MQ Advanced for Developers
- Rabbit MQ (open source)

Integration
- IBM Integration Bus
- IBM Integration Bus for Developers
- IBM DataPower Gateway Virtual Edition
- IBM DataPower Gateway for Developers
- IBM API Connect Professional
- IBM API Connect Enterprise

Watson
- IBM Watson Compare & Comply: Element Classification

Data Science and Business Analytics
- IBM Data Science Experience Local
- IBM Data Science Experience Developer Edition
- IBM Watson Explorer Deep Analytics Edition

Data Governance and Integration
- IBM InfoSphere Information Server for Evaluation

Management
- IBM Netcool - integration (Probe for ICP Services – Logging events & Monitoring alerts)

Connectivity
- IBM Voice Gateway Developer Trial

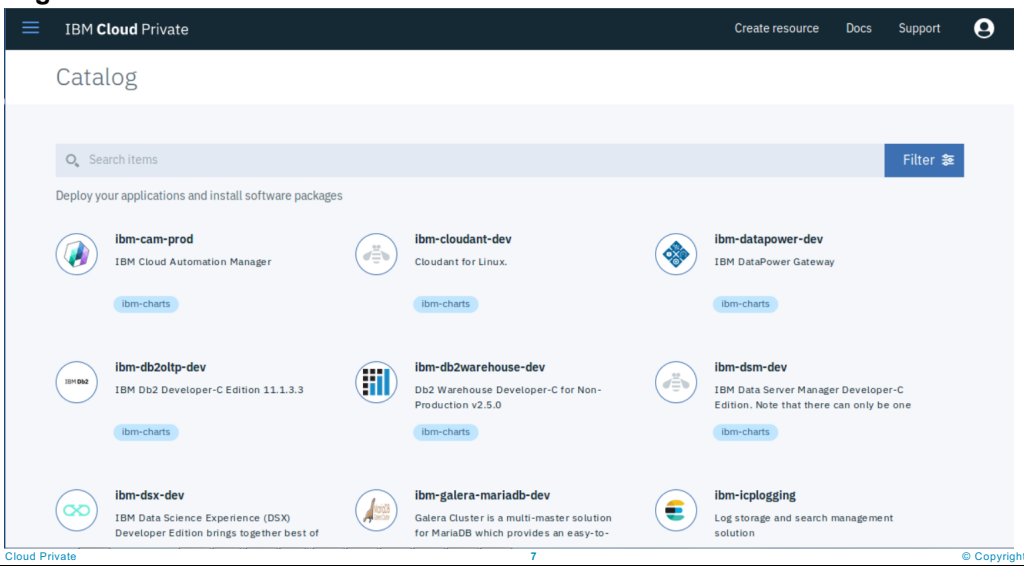
Tooling
- Web Terminal (open source)
- Skydive – network analyzer (open source)

HPC / HPDA
- IBM Spectrum LSF Community Edition
- IBM Spectrum Symphony Community Edition
- IBM Spectrum Conductor Tech Preview

IBM Training

Helm catalog

- **Catalog > Helm Charts**



Helm in IBM Cloud Private 7 © Copyright IBM Corporation 2018

IBM Training

Building and deploying applications with Kubernetes

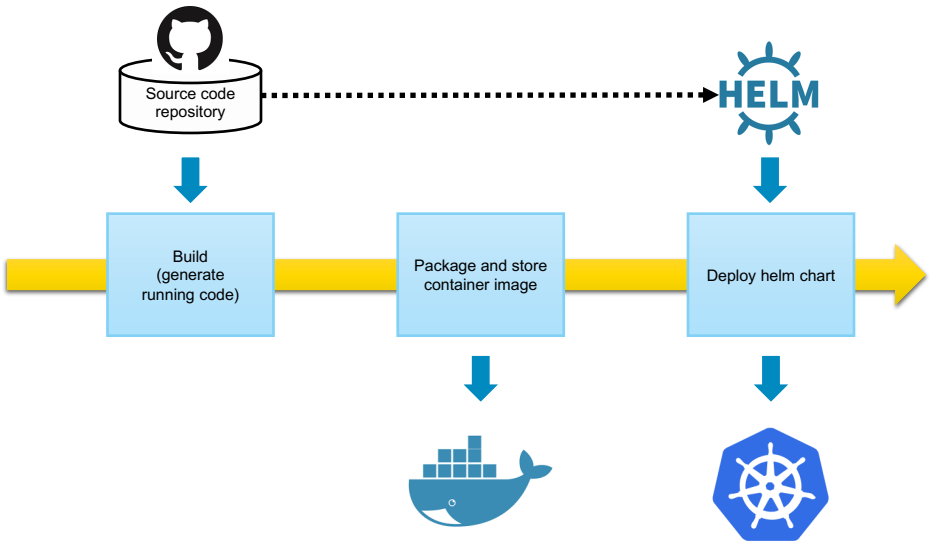
IBM Training

Objectives

- List the steps to build an application
- Identify steps to re-create and store the container image for an application
- Build a helm chart for an application
- Describe the process to redeploy a helm chart

IBM Training

Building and deploying in Kubernetes



Building and deploying applications with Kubernetes 3 © Copyright IBM Corporation 2018

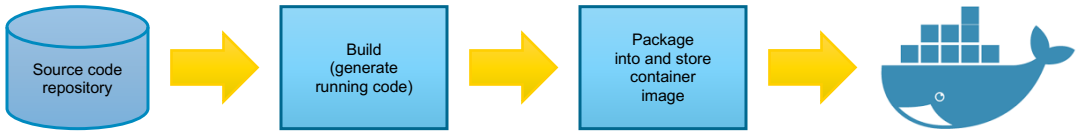
IBM Training

Build Process for Kubernetes

Building and deploying applications with Kubernetes 4 © Copyright IBM Corporation 2018

IBM Training

Build process



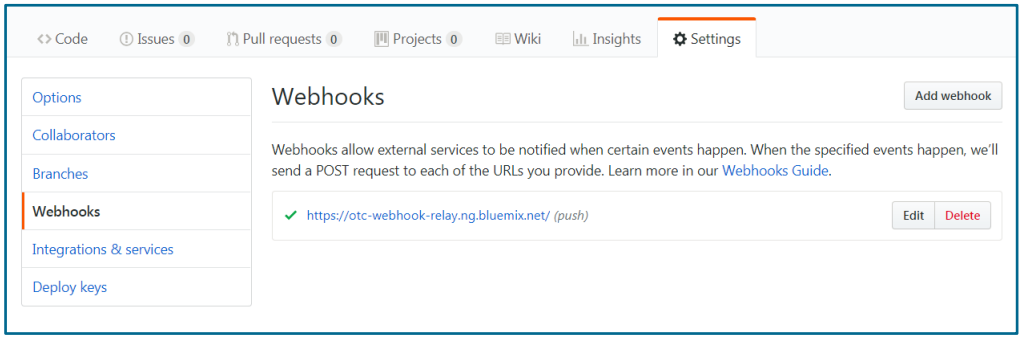
Git
SVN
SourceForge
. . .

maven
gradle
ant
make
. . .

docker build

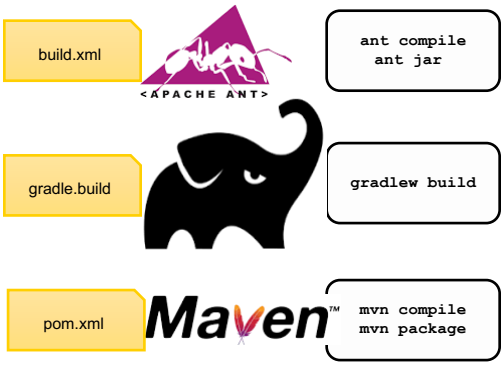
IBM Training

Source code repository



IBM Training

Build running code



Configuration file:

- Sets the project information (name, description etc)
- Defines target types and locations
- Specifies dependencies

Building and deploying applications with Kubernetes 7 © Copyright IBM Corporation 2018

IBM Training

Package container image

- FROM <initial image>
- RUN <a command in the image>
- LABEL <comment>
- EXPOSE <port>
- ENV <environment variable>
- ADD <a file from current path>
- COPY <a file from current path>
- VOLUME <volume to be created>
- USER <user to be used>[:<gid>]
- CMD <command to be executed>
- ENTRYPOINT <wrapper/shell to be executed>
- WORKDIR <initial working path>
- ARG <variables that can be passed for build>
- ONBUILD <instructions to be executed on future build>
- STOPSIGNAL <signal for container exit>
- HEALTHCHECK <cmd to check container health>
- SHELL <switch the default shell for RUN>

Dockerfile

```
docker build -t <image> <path>
```

Building and deploying applications with Kubernetes 8 © Copyright IBM Corporation 2018

IBM Training

Typical Dockerfile flow

- Load an existing image
  `FROM`
- Copy or add files into the image
  `ADD/COPY`
- Set an environment variables
  `ENV`
- Run commands to configure the image
  `RUN`
- Specify startup command and arguments
  `ENTRYPOINT/CMD`
- Set port to be available (mapped)
  `EXPOSE`
- Set initial working directory
  `WORKDIR`

```
FROM mysql

ADD scripts/load-data.sh load-data.sh
ADD scripts/load-data.sql load-data.sql
RUN chmod u+x load-data.sh

CMD ["mysqld"]
EXPOSE 3306
```

Building and deploying applications with Kubernetes 9

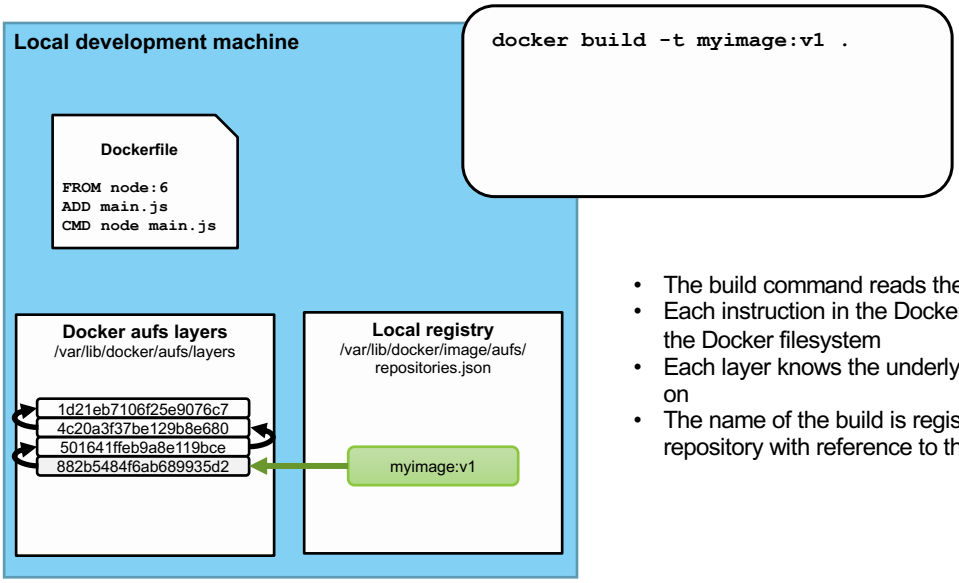
IBM Training

Working with a container image

Building and deploying applications with Kubernetes 10

IBM Training

Container image registry – 1

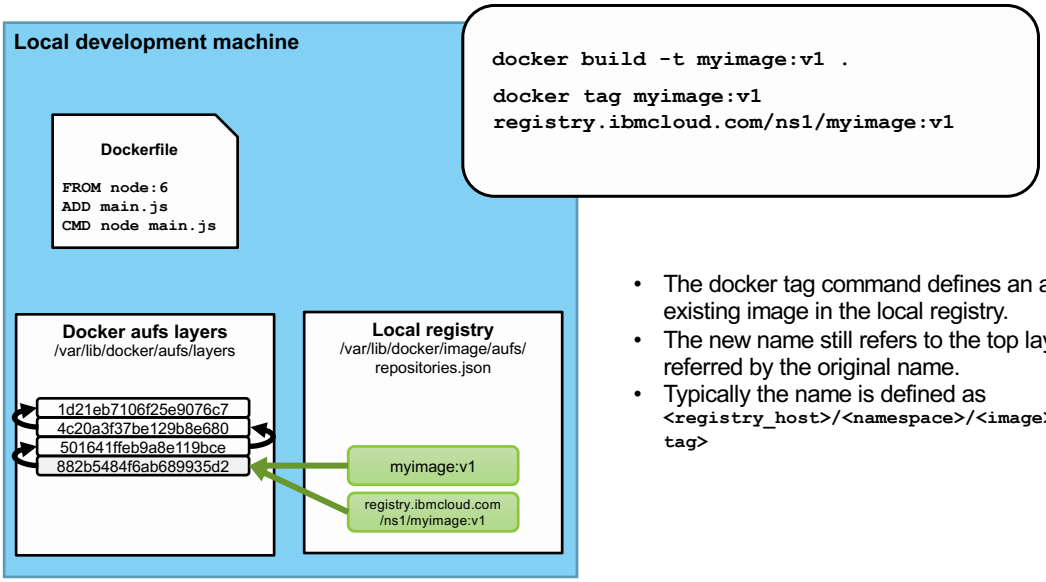


- The build command reads the Dockerfile
- Each instruction in the Dockerfile generates a layer in the Docker filesystem
- Each layer knows the underlying layer it is dependent on
- The name of the build is registered in the local repository with reference to the top layer

Building and deploying applications with Kubernetes 11 © Copyright IBM Corporation 2018

IBM Training

Container image registry – 2

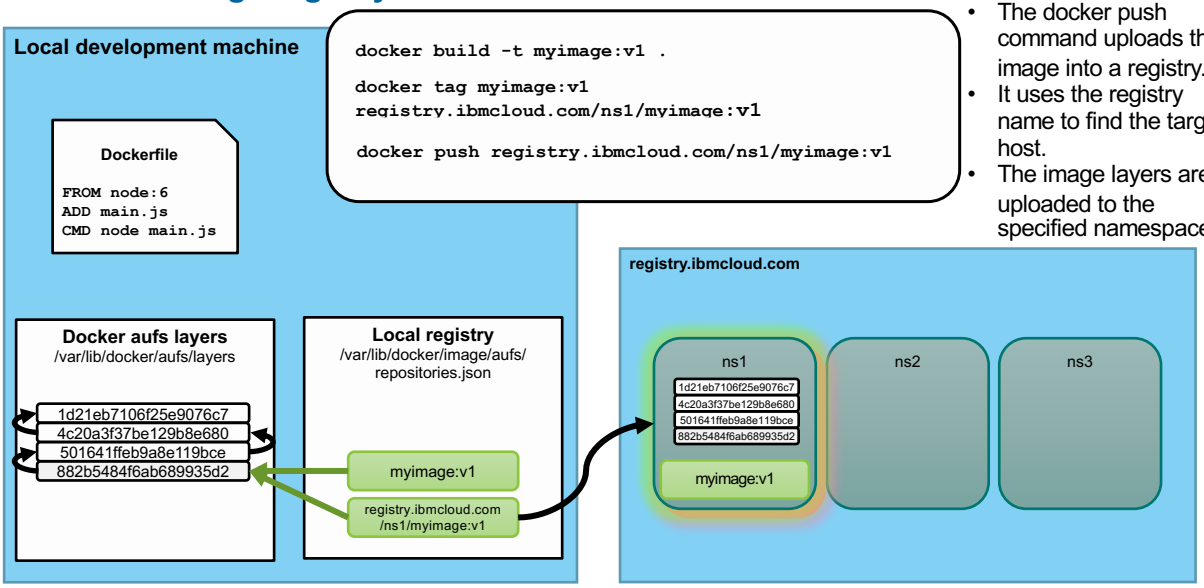


- The docker tag command defines an alias for an existing image in the local registry.
- The new name still refers to the top layers that was referred by the original name.
- Typically the name is defined as `<registry_host>/<namespace>/<image>:<version tag>`

Building and deploying applications with Kubernetes 12 © Copyright IBM Corporation 2018

IBM Training

Container image registry – 3

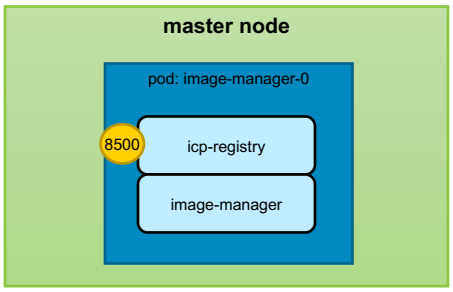


- The docker push command uploads the image into a registry.
- It uses the registry name to find the target host.
- The image layers are uploaded to the specified namespace.

IBM Training

IBM Cloud Private registry

- Registry port: 8500
- How to access ICP registry:
- ICP certificate
  `master:/etc/docker/certs.d/<cluster_domain>\:8500/ca.crt`
  - Must be copied to any docker client in /etc/docker/certs.d/<cluster_domain>\:8500/
  - Run: `docker login`



IBM Training

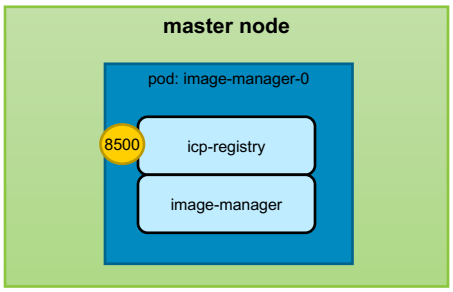
Using IBM Cloud Private registry

- Pulling image out from a private (not global registry entry)
- ImagePullSecrets
  - Create secret:

```
kubectl create secret docker-registry myregistrykey
--docker-server=<cluster_CA_domain>:8500
--docker-username=<user_name>
--docker-password=<user_password>
--docker-email=<user_email>
```

  - Use in template:

```
template:
  spec:
    imagePullSecrets:
    - name: myregistrykey
```



IBM Training

Using Helm charts for deployment

IBM Training

Helm chart components

```
helm create <chart_name>
```

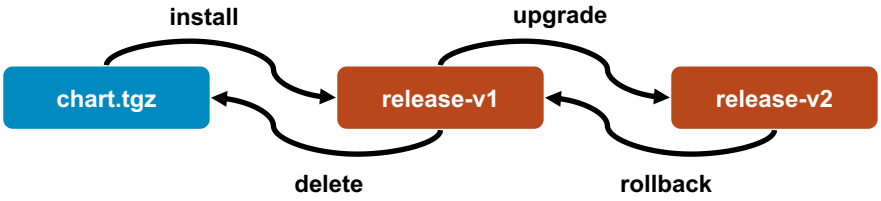
```
localuser@ubuntu-base:~$ tree catalog
catalog
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── ingress.yaml
│   ├── NOTES.txt
│   └── service.yaml
└── values.yaml

2 directories, 7 files
```

- The **`create`** sub-command build the directory structure
- Chart.yaml and values.yaml are used for variable substitution
- templates directory contains resources that this Helm chart built
- templates also contains lifecycle scripts (will be discussed later)

IBM Training

Helm chart life cycle



IBM Training

Chart Lifecycle Hooks

- pre-install
  - Executes after templates are rendered
  - Before any resources are created
- post-install
  - Executes after all resources are loaded
- pre-delete
  - Executes before any resources are deleted
- post-delete
  - Executes after all of the release's resources have been deleted
- pre-upgrade
  - Executes after templates are rendered
  - Before any resources are loaded
- post-upgrade
  - Executes after all resources have been upgraded
- pre-rollback
  - Executes after templates are rendered
  - Before any resources have been rolled back
- post-rollback
  - Executes after all resources have been modified

- A hook can be any Kubernetes resource
  - A hook is often a Kubernetes job
  - Goes in the templates directory
- Sample hook

```
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{ .Release.Name }}-initialize-tables"
  annotations:
    # This is what defines this resource as a hook. Without this line, the
    # job is considered part of the release.
    "helm.sh/hook": post-install
    "helm.sh/hook-weight": "30"
spec:
  template:
    metadata:
      name: {{ .Release.Name }}-initialize-tables
    spec:
      restartPolicy: Never
      containers:
      - name: initialize-tables
        image: "ibm-cloud-academy/ibmcloudcli:v01"
        command: [ "/bin/sh", "-c" ]
        args: ["git clone https://github.com/ibm-cloud-academy/tools;cd tools;./initdb.sh"]
        imagePullPolicy: IfNotPresent
```

Building and deploying applications with Kubernetes 19 © Copyright IBM Corporation 2018

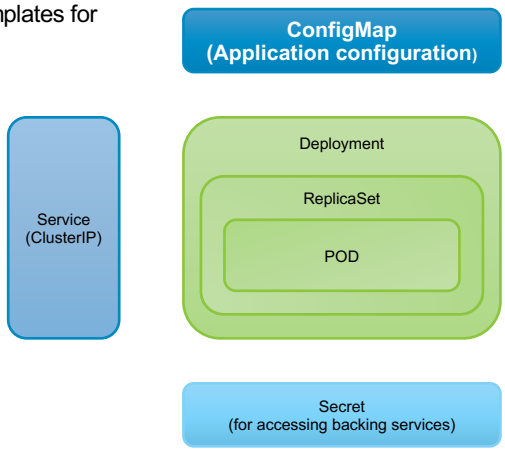
IBM Training

Microservice in Kubernetes – resources

The Helm chart must contain the templates for each of these Kubernetes resources

```
localuser@ubuntu-base:~$ tree catalog
catalog
├── charts
├── Chart.yaml
├── templates
│   ├── cleanup-job.yaml
│   ├── configmap.yaml
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── init-job.yaml
│   ├── NOTES.txt
│   ├── secret.yaml
│   ├── service.yaml
│   └── upgrade-job.yaml
└── values.yaml

2 directories, 11 files
```



Building and deploying applications with Kubernetes 20 © Copyright IBM Corporation 2018

IBM Training

Deploying a helm chart

IBM Training

Checking chart

The **`lint`** sub-command checks and verifies the chart definitions.

```
helm lint <chart_name>
```

Options

```
      --namespace string    namespace to install the release into (only used if -
      --set stringArray     set values on the command line (can specify multiple
      --strict              fail on lint warnings
  -f, --values valueFiles   specify values in a YAML file (can specify multiple)
```

Options inherited from parent commands

```
      --debug                     enable verbose output
      --home string               location of your Helm config. Overrides $HELM_H
      --host string               address of Tiller. Overrides $HELM_HOST
      --kube-context string       name of the kubeconfig context to use
      --tiller-namespace string   namespace of Tiller (default "kube-system")
```

```
localuser@ubuntu-base:~$ helm lint catalog
==> Linting catalog
[INFO] Chart.yaml: icon is recommended

1 chart(s) linted, no failures
```

IBM Training

Packaging helm chart

```
localuser@ubuntu-base:~$ tree catalog
catalog
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── ingress.yaml
│   ├── NOTES.txt
│   └── service.yaml
└── values.yaml

2 directories, 7 files
```

helm package <chart> → <chart>.tgz

helm repo add <url>
helm install <chart>

helm serve

index.yaml

```
apiVersion: v1
entries:
  catalog:
  - apiVersion: v1
    appVersion: "1.0"
    created: 2018-03-04T17:56:03.183684258-05:00
    description: Catalog microservice chart
    digest: cfd006[illegible]520e04e...
    name: catalog
    urls:
    - http://127.0.0.1:8879/catalog-0.1.0.tgz
    version: 0.1.0
  customer:
  - apiVersion: v1
    appVersion: "1.0"
    created: 2018-03-04T17:56:03.184256872-05:00
    description: Customer microservice chart
    digest: a675ba78342f4b32...
    name: customer
    urls:
    - http://127.0.0.1:8879/customer-0.1.0.tgz
    version: 0.1.0
generated: 2018-03-04T17:56:03.182815694-05:00
```

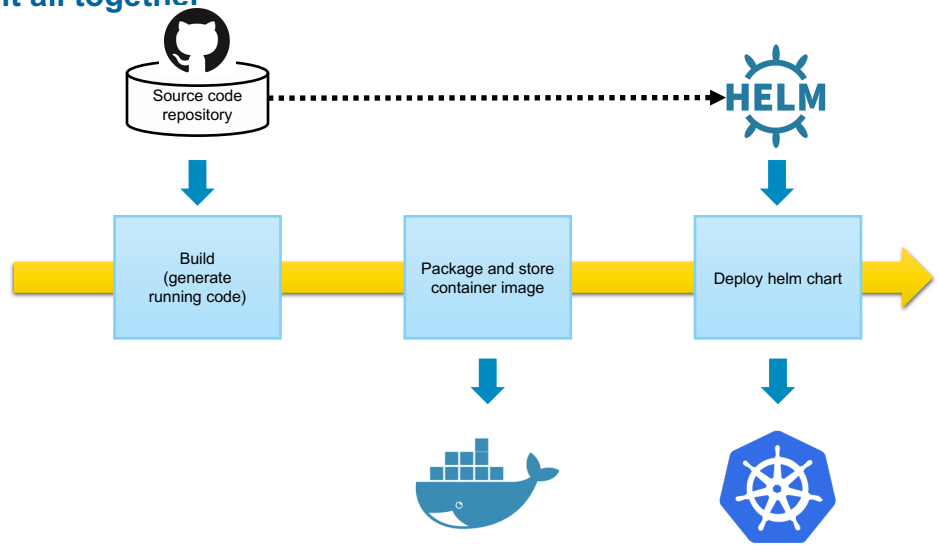
catalog-0.1.0.tgz

customer-0.1.0.tgz

Building and deploying applications with Kubernetes 23 © Copyright IBM Corporation 2018

IBM Training

Putting it all together



Building and deploying applications with Kubernetes 24 © Copyright IBM Corporation 2018

IBM Training IBM

DevOps Concepts

IBM Training IBM

Agenda

- What is DevOps
- Definitions - CI / CD
- Zero Downtime Deployment
- DevOps in Twelve-Factor Apps
- Implementing DevOps

IBM Training

What is DevOps?

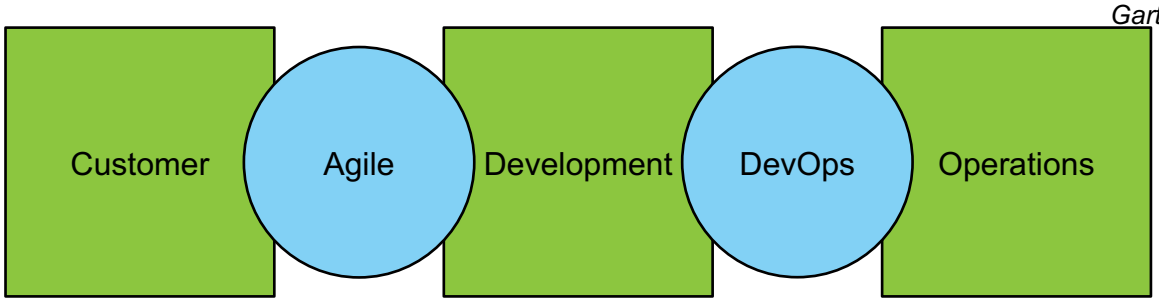
IBM Training

Definition

DevOps (*n.*) –

"DevOps is a philosophy, a **cultural shift** that **merges operations with development** and demands a **linked toolchain** of technologies to facilitate collaborative change. DevOps toolchains … can include dozens of non-collaborative tools, making the task of automation a technically complex and arduous one."

Gartner



IBM Training

Every organization has critical business apps

Trading halted for half a day on the biggest US exchange for financial options following an outage caused by software problems.

Airline canceled more than 700 flights and another 765 flights are delayed due to a software outage – Blamed ticketing partner while the real problem was on their end

and in today's competitive world, those apps can never go DOWN!!!

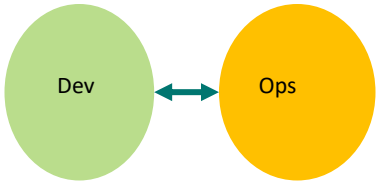
Software problem led to two days of downtime at the largest bank in Europe has tarnished their image as the most reliable banking website.

Not surprisingly, many angry customers poured out their wrath via social networking after the largest video streaming company had a software outage for more than 20 hours

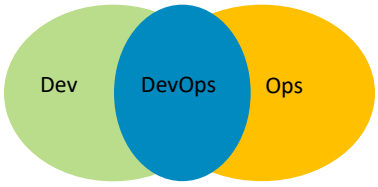
DevOps Concepts 5 © Copyright IBM Corporation 2017

IBM Training

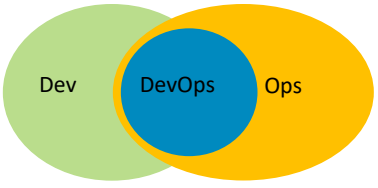
Evolving perspective on DevOps

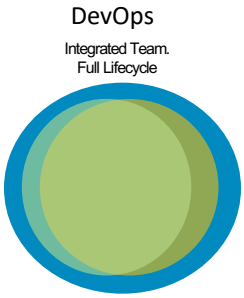


Large Enterprises
Traditional Workloads
(historical view)



Large Enterprises with
Traditional Workloads
Moving to Cloud



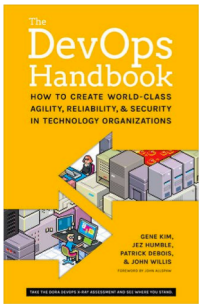


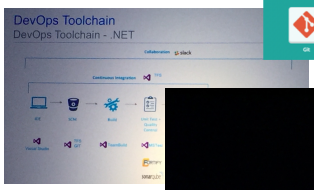
Newer companies with
Cloud-Native
Workloads

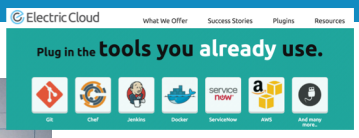
DevOps Concepts 6 © Copyright IBM Corporation 2017

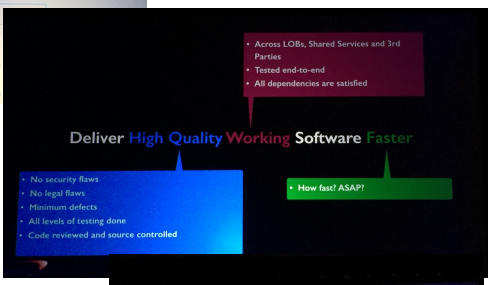
IBM Training

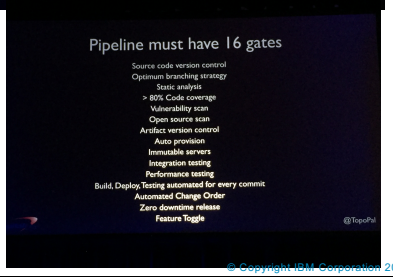
Newer approaches becoming mainstream











2015

Large Enterprises experimenting with full integrated DevOps lifecycle for cloud-native workloads with small innovation teams.

How to scale to the enterprise?

2016 / 2017

Several testimonials of large enterprises applying integrated DevOps to real projects.

Message expanding beyond just Continuous Integration / Continuous Delivery to also include reliability, security and insights.

Containers and microservice architectures more visible

DevOps toolchains enter mainstream

DevOps Concepts 7 © Copyright IBM Corporation 2017

IBM Training

Definitions - CI / CD

8 © Copyright IBM Corporation 2017

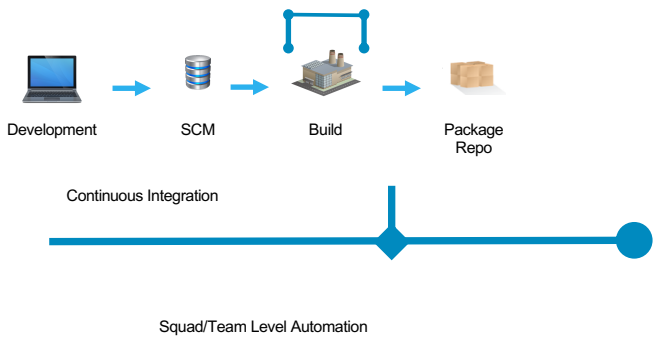
IBM Training

Definitions

- **CI / CD**
  - Acronym for **Continuous Integration** plus **Continuous Deployment**
- **Continuous integration**
  - Creating packaged builds of the latest code changes as soon as they're available
- **Continuous deployment**
  - Progressing each new packaged build through the deployment lifecycle stages as rapidly as possible, resulting in that package getting into production
- **Continuous delivery**
  - Continuous integration + continuous deployment
- **Delivery pipeline**
  - An automated sequence of steps to perform CI / CD

IBM Training

Continuous Integration



Frequently performing all of these steps in sequence:

- Development
  - Rapidly implementing changes in small, tested batches
- SCM (Source Code Management)
  - Merging changes from multiple developers
  - Tools like GitHub, SVN, etc
- Build
  - Creating new deployment artifacts
  - Tools like Jenkins, Gradle, Maven, etc
- Package
  - Installing builds into runtimes
  - Releasing runtimes as immutable images
  - Cloud Foundry push, Building container images

IBM Training

Continuous integration – package step

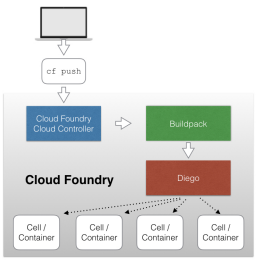
In the package step, an immutable image is created

- An immutable image does not get changed – only used for deploying instances
- If it needs to change, you delete it and create a new one

Examples

- Docker containers
  - A container image is created from a Dockerfile
  - After that, it is only deployed as a container, not changed
  - If you need to make changes, make a new container image by creating a new build and running the Dockerfile again

- Cloud Foundry
  - When an app is pushed to Cloud Foundry, the buildpack creates a droplet – from that CF creates Garden container instances that run the app
  - If you need to make changes, create a new build and push it



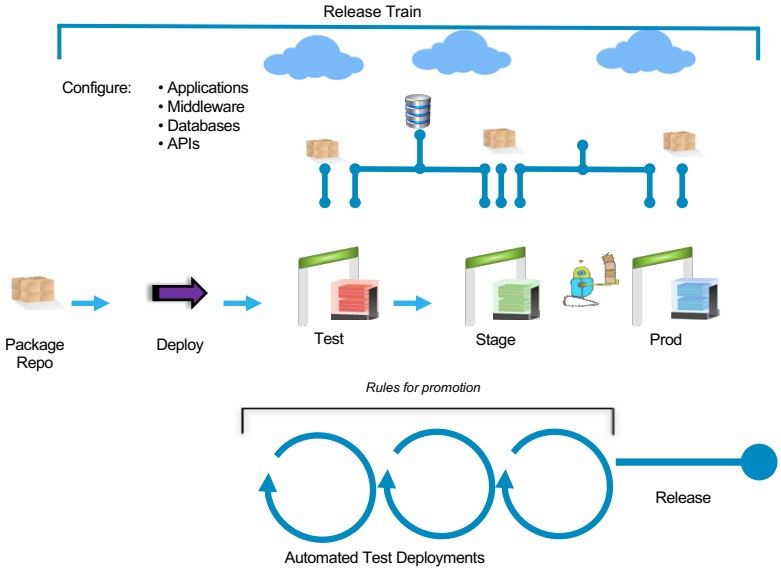
DevOps Concepts 11 © Copyright IBM Corporation 2017

IBM Training

Continuous Deployment

Rapidly progressing the latest packaged build through the test lifecycle stages and into production

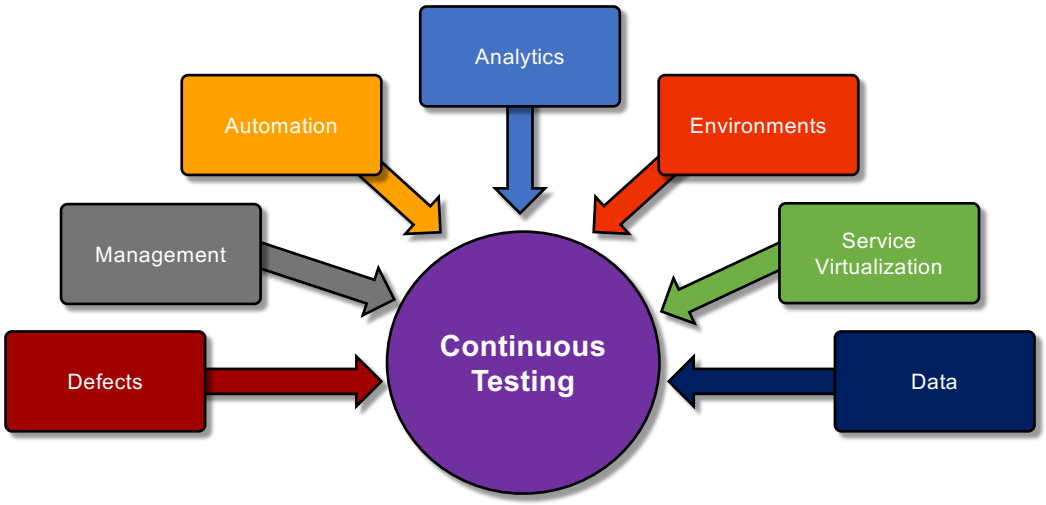
- Deploy to Test
  - Perform functional testing
  - Automated use of test tools
- Deploy to Stage
  - Rehearse production deployment
  - Perform integration testing
- Deploy to Prod
  - Make the build available to users



DevOps Concepts 12 © Copyright IBM Corporation 2017

IBM Training

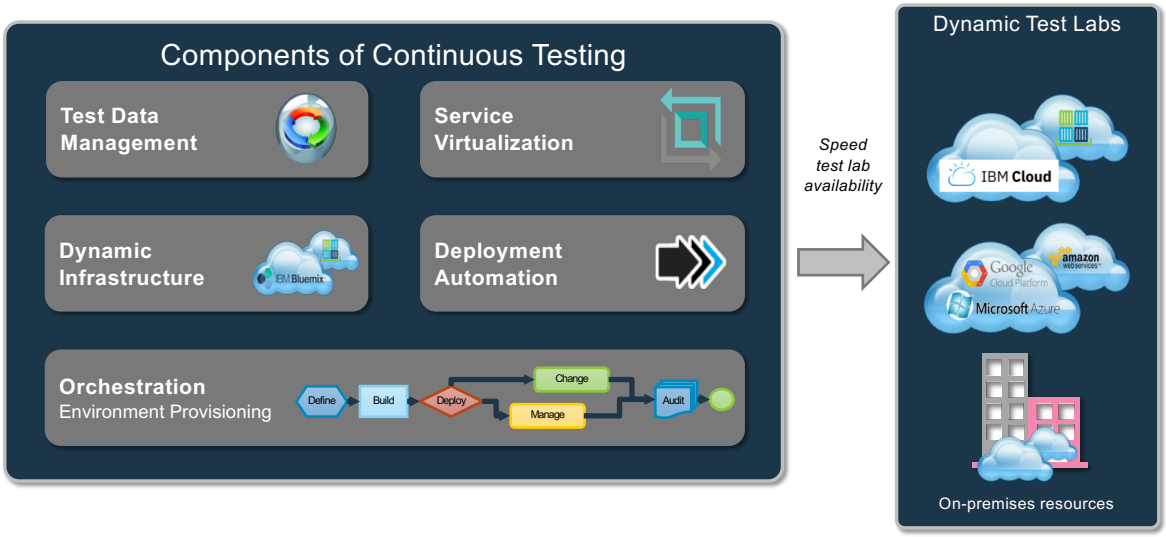
Building a continuous testing culture requires people, process, tools & time



DevOps Concepts 13 © Copyright IBM Corporation 2017

IBM Training

Test continuously in realistic production-like test labs

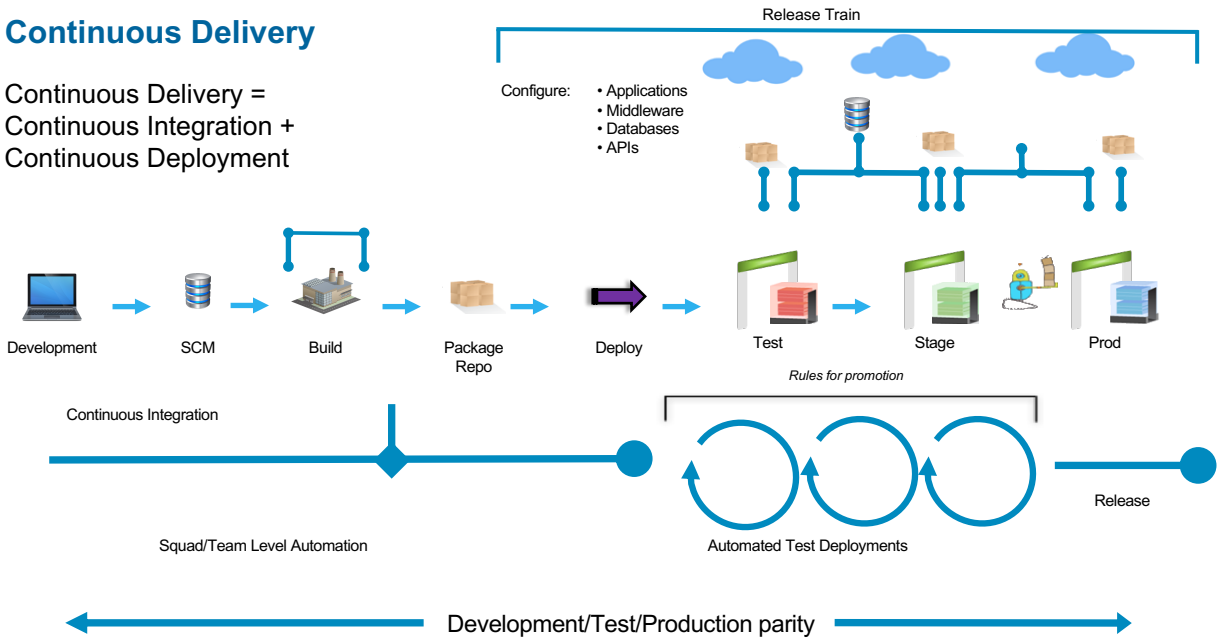


DevOps Concepts 14 © Copyright IBM Corporation 2017

IBM Training

Continuous Delivery

Continuous Delivery =
Continuous Integration +
Continuous Deployment



DevOps Concepts 15 © Copyright IBM Corporation 2017

IBM Training

Zero Downtime Deployment

16 © Copyright IBM Corporation 2017