Move legacy apps to Windows Containers

(to take advantage of modern infrastructure and orchestration)

Regan Murphy
Software Engineer
Microsoft

@nzregs

Why **modernise?**

Reasons to modernise

- Aging infrastructure
  - Low efficiency and reliability
  - High operational costs and CapEx
  - Growing security/audit requirements
- Stagnant Architecture
  - Legacy stack and code
  - Long deployment times
  - Hard (or impossible) to add new functionality

Benefits of modernisation

- Turn CapEx into OpEx
- Increased Operational Efficiency
  - Get out of the data center business
  - Meet security and compliance requirements
  - Reduce time and budget spend on infrastructure management
- Rapid Innovation
  - Ship new capabilities faster
  - Achieve Scalability with confidence
  - Better collaboration across Business, Ops, IT, and Dev teams

Why containers?

Traditional (Virtual) Machines

- Complex deployments – multiple components share same server
- Highly inefficient – especially if you deploy app-per-VM
- Slow scale – takes time to add/remove instances

Benefits of containers

- Better agility – ship apps faster
- Portability – easily move workloads
- Density – achieve resource efficiency
- Rapid scale – scale easily to meet demand

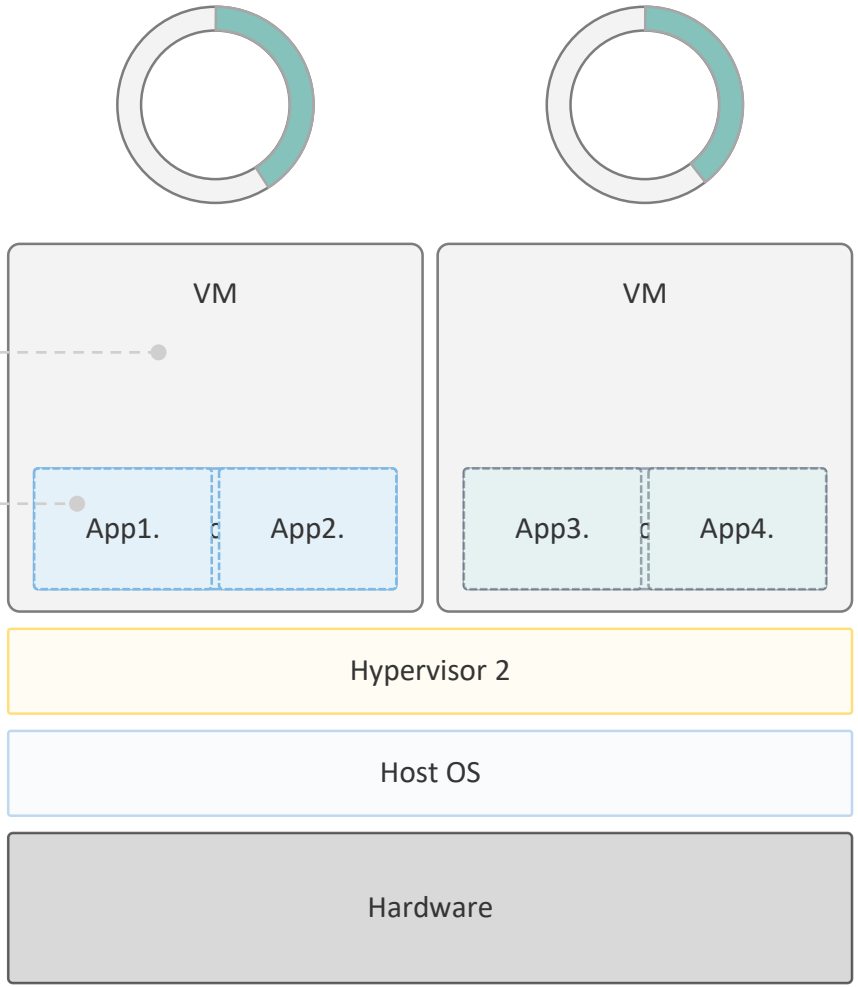
The container **advantage**

Traditional virtualized environment

Low utilization of resources

Apps and their dependencies separated for portability, collocated for density

App1 and App2 must share dependencies. App1 rollout will affect App2 stability



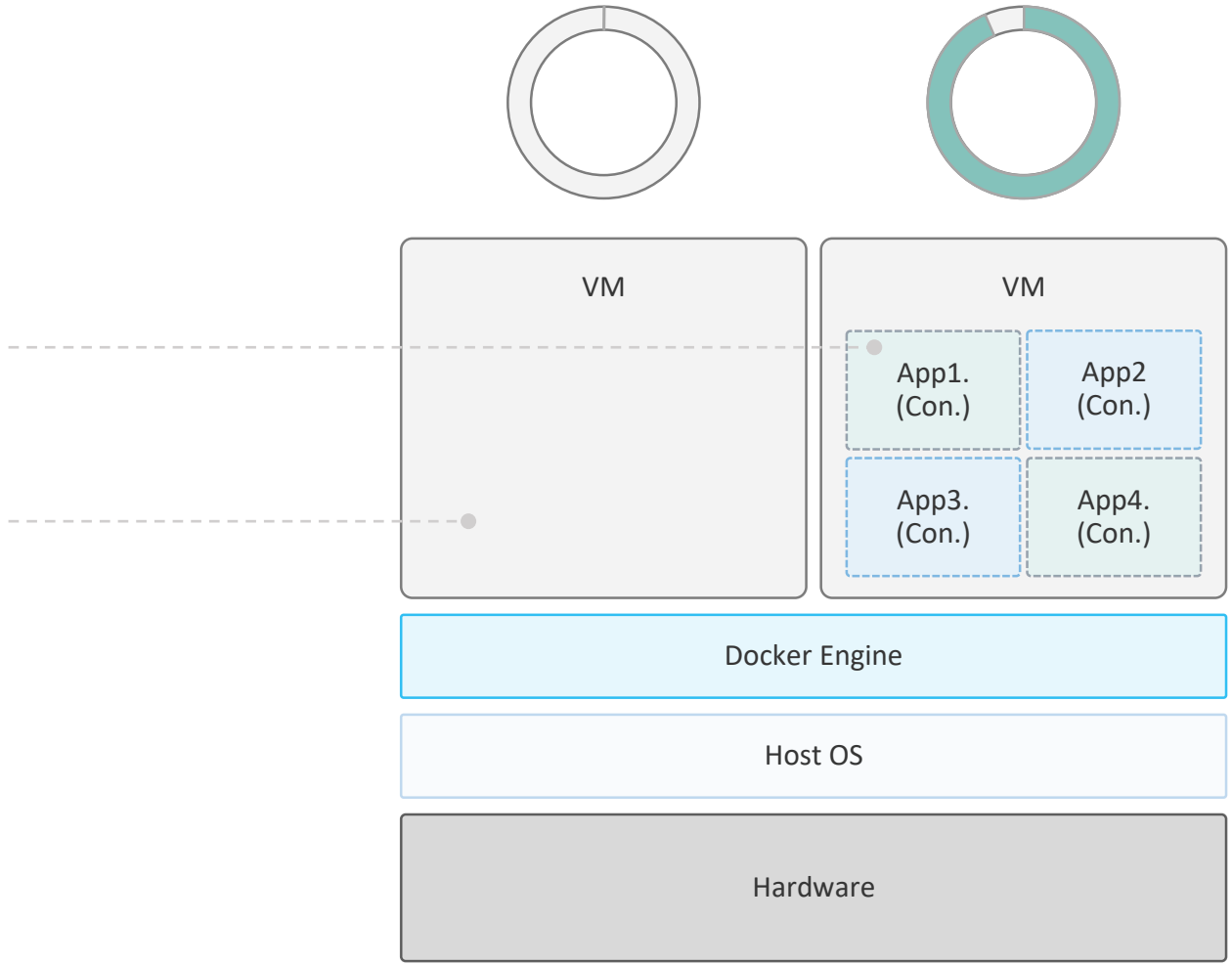
The container **advantage**

Containerized environment

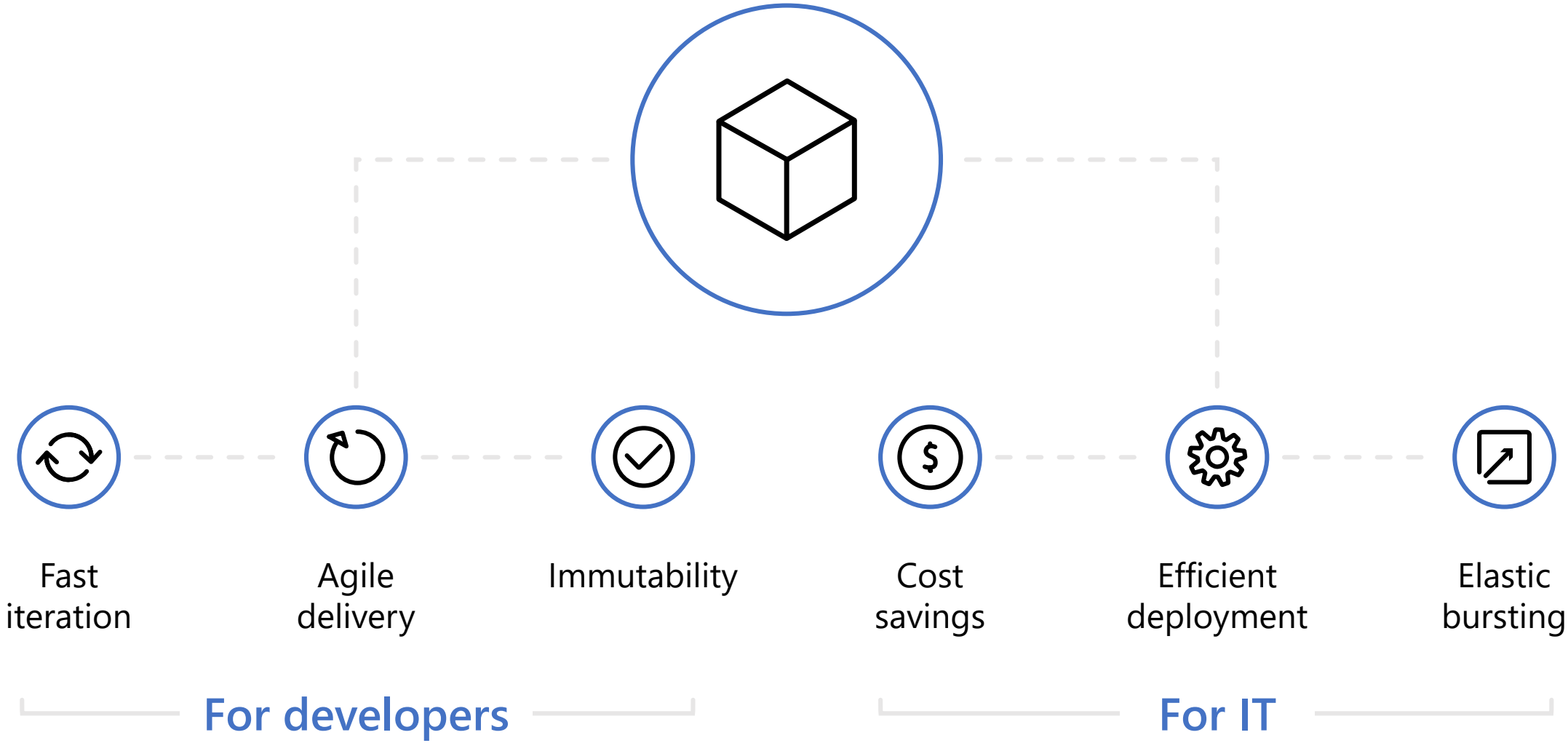
Migrate apps and their dependencies into containers and consolidate on underutilized VMs for improved density and isolation

Decommission unused resources for efficiency gains and cost savings

Container is lighter weight and faster to scale dynamically



The container **advantage**



Works on **my machine!**

Azure Kubernetes Service

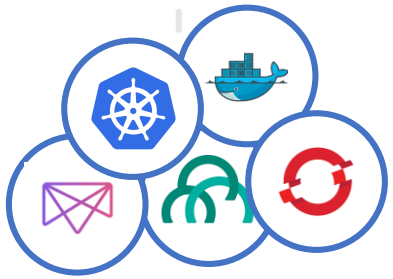
Azure Container Instances

Azure Batch

Service Fabric

Azure App Service

Virtual Machines (VM+ VMSS)



Partner Offerings

Works on **Microsoft Azure**

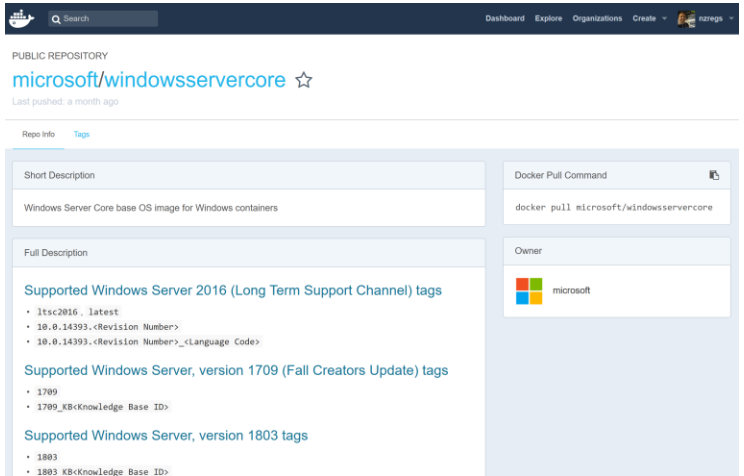
and more

Windows Server Containers

Windows Server container versions

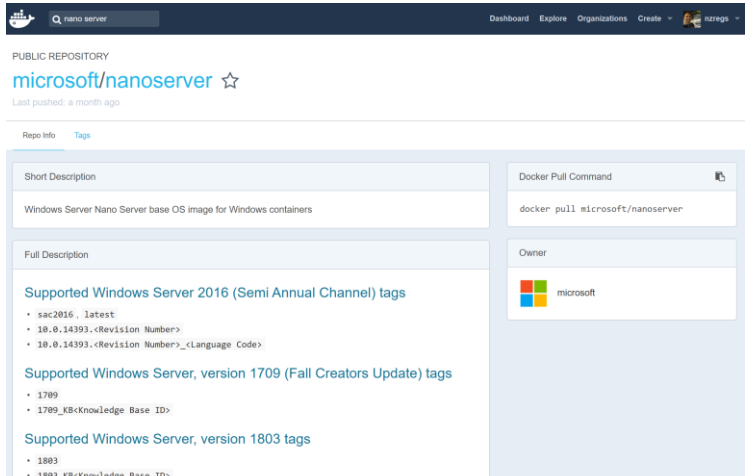
Windows Server Core

- Use for legacy application
- Includes full .NET framework
- Can run IIS and Windows Services
- Larger Container Sizes (GBs)



Nano Server

- Use for cloud-first applications
- Optimised for .NET Core
- Powershell Core, .NET Core, and WMI not included
- Available as container image only from 1709



Getting started

Migrate?
Rewrite?
Redeploy?
Maintain?

Before starting on any project to ‘containerize’ an application, weigh up the pros/cons of the various options.

Migrate – to windows containers

Rewrite – in modern version of .net, or .net core, then containterize

Redeploy - to a PaaS service such as Web Apps on Azure

Maintain – maybe keeping a legacy VM is the best path forward

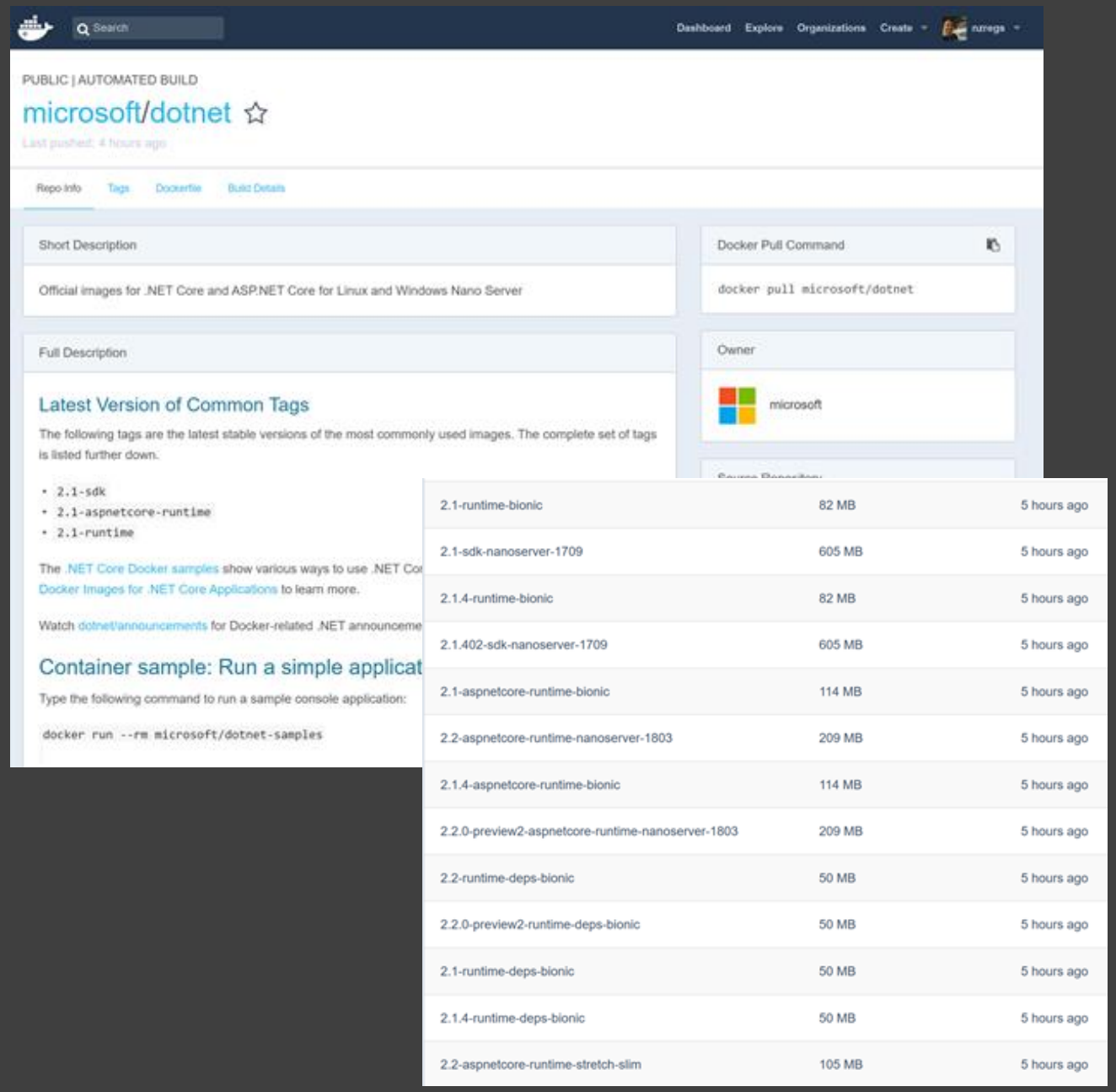
.net application?

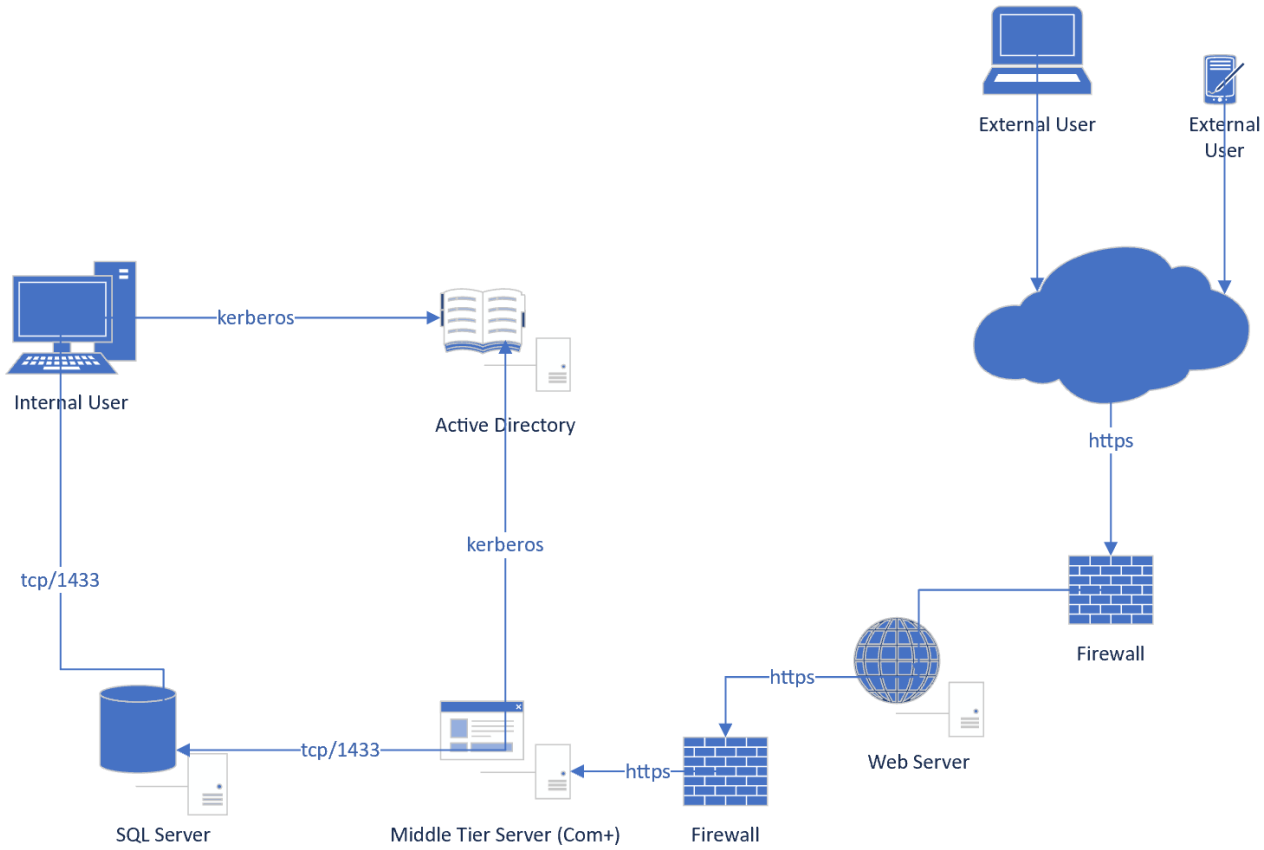
consider re-writing or re-targeting your application for .net core

you could then use .net core on either linux or windows nanoserver containers

Broad support for .net core!

- microsoft/dotnet images are maintained for both windows and linux
- there are SDK and runtime images for alpine and bionic linux distros, windows nanosever
- you can also build your own images





A Legacy Application

A traditional application consisting of:

- Client-Server “Fat Client” where the internal user connects directly to the SQL Server
- 3-Tier Application for Web Clients

This is a real CRM application, from approx. 2002 – and still deployed as-is today.

Which components into containers?

Active Directory Server

No.

SQL Server

This is not a good candidate for a number of reasons.

Middle Tier Application Server

The middle tier application can scale-out and is stateless.

Could be tricky as there is a COM+ component that relies on Windows Authentication.

Web Server

The web server is an excellent candidate for containerization.

There are multiple websites co-hosted on one server that could be split into separate containers

The web server can easily be scaled out (but be careful of session state in the websites).

Classic ASP? DCOM? No worries!

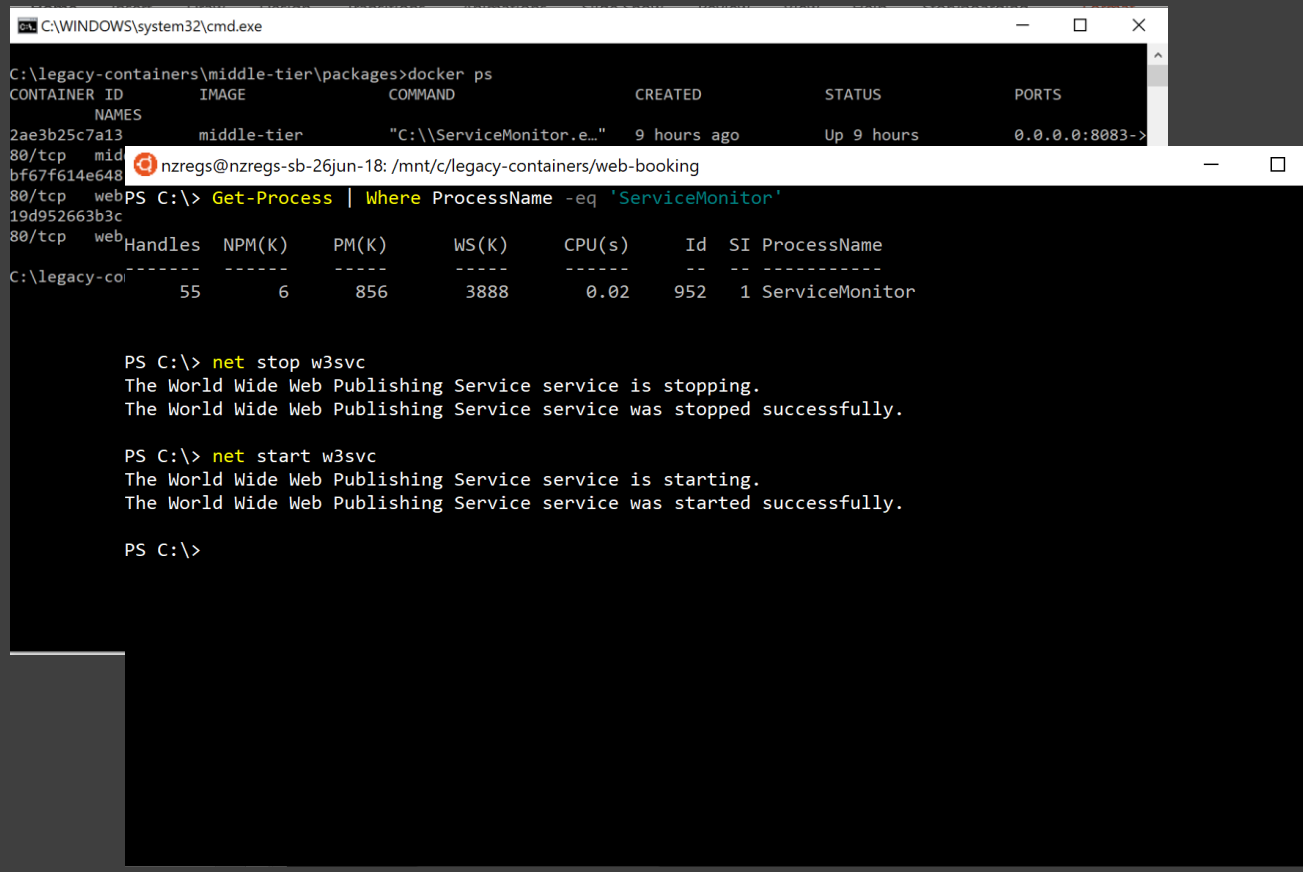
Get stuck in!

no GUI!

There is no GUI in a Windows Server container.

You can connect to a running container – e.g. **docker exec** or **kubectl exec** - and access a cmd or powershell shell – but this should be for development test/debug only.

Persist commands in a Dockerfile (or orchestrator equivalent) instead.



Learn cmd and powershell

Command Line Utils

- **tasklist:** show running tasks
- **icacls:** set file/folder permissions
- **appcmd.exe:** configure IIS site settings
- **dism.exe:** windows features installer
- **msiexec.exe:** installing packaged apps
- **regsvr32.exe:** register COM/DCOM components

Powershell

- **Add-WindowsFeature**, **Enable-WindowsOptionalFeature**: add/install windows features
- **Get-Process**: show running tasks
- **Start-Process**: start an app or cmd
- **Invoke-WebRequest**: downloading files or making REST calls

Deploying legacy packaged apps

A lot of legacy apps have installers wrapped up in a “setup.exe” file.

Need to learn how to silently install from setup.exe.

Sometimes the setup.exe doesn’t handle silent installs. You can then try and extract MSI packages and perform silent install from msiexec.

Installs go a lot smoother if you pre-install any pre-requisites – this avoids multi-step installs.

It helps to set logging to ‘verbose’ when testing/debugging installs

Extract MSI (multiple methods)

- `setup.exe /extract_all:”c:\temp\extract”`
- `setup.exe /extract:”c:\temp\extract”`
- `Try 7zip or other utility`

Install with verbose logs:

- `setup.exe /s /v”/qn L*V logfile.log”`
- `msiexec.exe /qn /lv “logfile.log”`

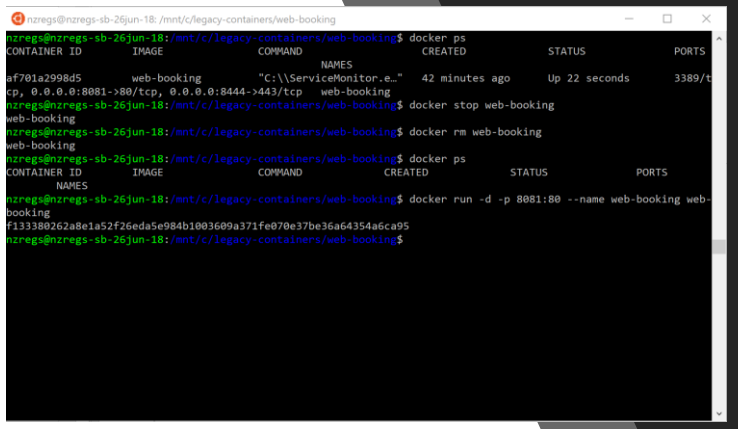
Dockerfile

- Declarative way of building container
- Containers should be immutable
- ENTRYPOINT is what runs when container is started up. Can be an exe or a script
- If your container starts and finishes a task, it will exit. We set the ENTRYPOINT to “ServiceMonitor.exe” in this example which will watch the IIS web worker and terminate if it stops
- Images build with “docker build” command

Dockerfile

```
19
20  FROM microsoft/windowsservercore:1803
21
22  SHELL ["powershell", "-command", "$ErrorActionPreference = 'St
23
24  # Install IIS, add features, enable 32bit on AppPool
25  RUN Install-WindowsFeature -name Web-Server; `
26      Add-WindowsFeature Web-Static-Content, Web-ASP, WoW64-Supp
27      Import-Module WebAdministration; `
28      set-itemProperty IIS:\apppools\DefaultAppPool -name "enabl
29      Set-WebConfigurationProperty -PSPath MACHINE/WEBROOT/APPHO
30      Restart-WebAppPool "DefaultAppPool"
31
32  # Deploy service monitor to keep container running until(if) i
33  RUN powershell -Command `
34      Add-WindowsFeature Web-Server; `
35      Invoke-WebRequest -UseBasicParsing -Uri "https://dotnetbir
36
37  # Networking
38  EXPOSE 80
39
40  ENTRYPOINT ["C:\\ServiceMonitor.exe", "w3svc"]
41
42  # Copy local website files directory into container
43  COPY "website\." "c:\inetpub\wwwroot\."
44  RUN icacls "C:\inetpub\wwwroot\error" /grant IIS_IUSRS:M;
```

Build/Start/Connect/List/Stop/Remove container



Build image from Dockerfile in current directory and name/tag web-portal

- **docker build . –tag web-portal**

Start detached, map port 8081 to container port 80, give it name/tag web-portal, use v1 of the web-portal image

- **docker run –d –p 8081:80 –name web-portal web-portal:v1**

Connect to running container, hold “tty” for cmd/powershell:

- **docker exec –ti web-portal cmd.exe**
- **docker exec –ti web-portal powershell.exe**

Show running/all containers:

- **docker ps | docker ps –a**

Stop Container:

- **docker stop web-portal**

Remove container:

- **docker rm web-portal**

Remove image:

- **docker rmi web-portal**

Windows Server container
IIS + Classic ASP + ActiveX

DEMO

Kubernetes

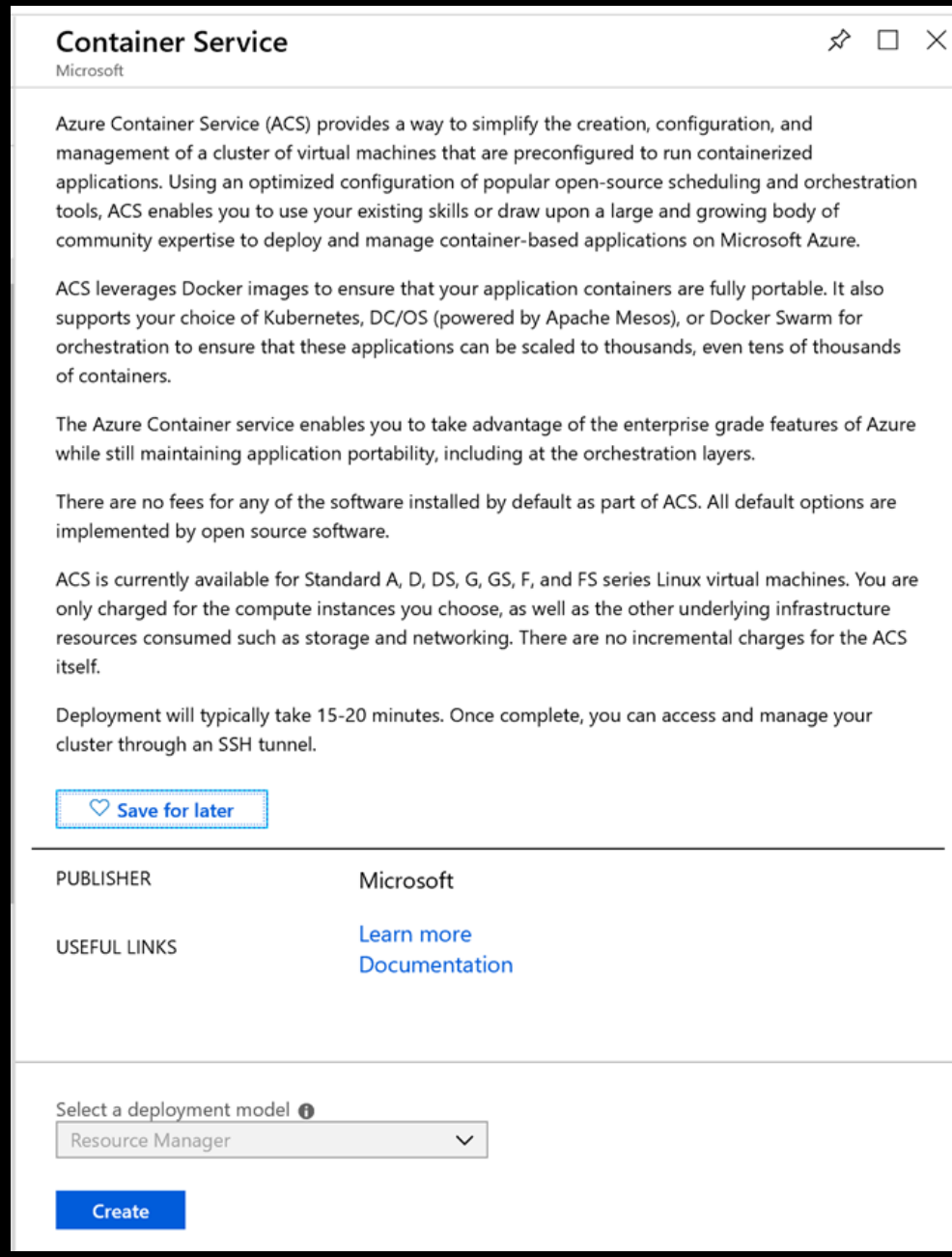
Why Kubernetes?

- Originally came from Google, is now run by Cloud Native Computing Foundation (CNCF)
- Becoming the default orchestrator – and at a recent dockercon, Docker announced native support for Kubernetes alongside the existing support for their own orchestrator Docker Swarm
- First class support on public clouds. E.g:
  - Microsoft Azure Kubernetes Service- AKS
  - Google Kubernetes Service – GKS
  - Catalyst (as you heard this morning)
  - Amazon Elastic Container Service for Kubernetes
- Rich ecosystem of supporting tools – like Helm, Draft, Brigade, Kashti
- Open source

Create a kubernetes cluster with Windows Server nodes in Azure

Via the portal – look for “Container Service” and choose Kubernetes as orchestrator (other options are Swarm and DC/OS)

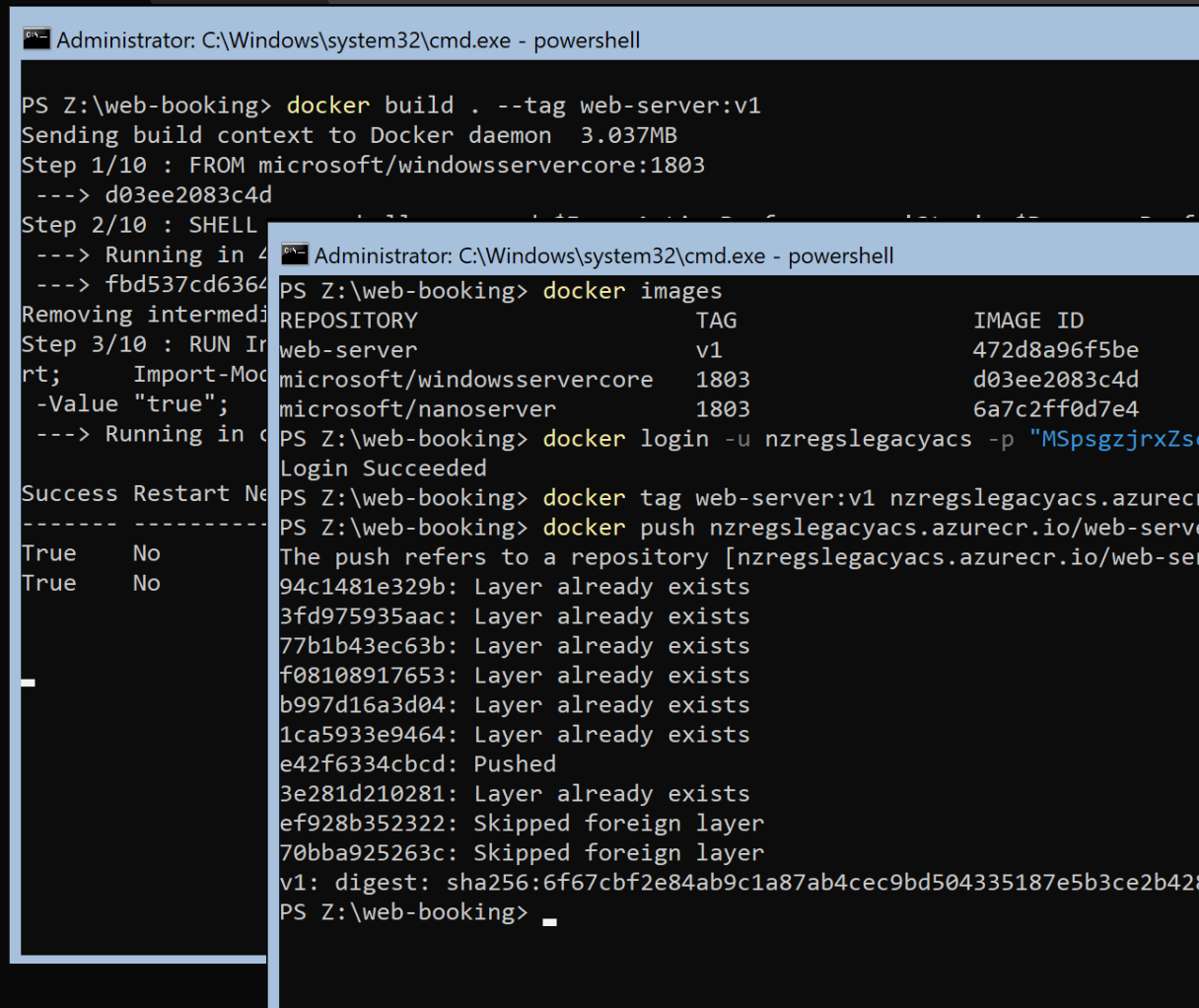
Via script using acsengine https://github.com/Azure/acs-engine/blob/master/docs/kubernetes/windows.md



Build and Push image to container registry

Pushing a windows server core container means sending ~5GB!

Push to a private registry



Create YAML Deployment

Because we are pulling from a private registry that requires a login, first create the secret on kubernetes (using the service principal that was deployed with k8s in this case):

```
kubectl create secret docker-registry
regcred --docker-server
nzregslegacyacs.azurecr.io --docker-
username 88d6ca05-****-****-****-
75df****d4f2 --docker-password
ejhhewe********//plkZbLcmxUKcSeQ+JF7+Uo= --
docker-email regan.murphy@example.com
```

```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: web-booking
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: web-booking
    spec:
      containers:
      - name: web-booking
        image: nzregslegacyacs.azurecr.io/web-bo
        ports:
        - containerPort: 80
          protocol: TCP
      imagePullSecrets:
      - name: regcred
---
apiVersion: v1
kind: Service
metadata:
  name: web-booking
spec:
  selector:
    app: web-booking
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

Apply deployment to Kubernetes Cluster

Apply the deployment file

Watch the pods and services

Useful commands:

```
kubectl get service --watch
kubectl get pods
kubectl describe pod <PODNAME>
```

```
nzregs@nzregs-sb-26jun-18: /mnt/c/legacy-containers/web-booking
nzregs@nzregs-sb-26jun-18:/mnt/c/legacy-containers/web-booking$ kubectl get service -
NAME          TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
kubernetes    ClusterIP      10.0.0.1       <none>         443/TCP        2h
web-booking   LoadBalancer   10.0.185.104   <pending>      80:31611/TCP   3m
web-booking   LoadBalancer   10.0.185.104   52.187.229.208   80:31611/TCP   4m
^Cnzregs@nzregs-sb-26jun-18:/mnt/c/legacy-containers/web-booking$ kubectl get pod
NAME                           READY     STATUS              RESTARTS   AGE
web-booking-143594665-ztfjx    0/1       ContainerCreating   0          4m
nzregs@nzregs-sb-26jun-18:/mnt/c/legacy-containers/web-booking$ kubectl describe pod
Name:           web-booking-143594665-ztfjx
Namespace:      default
Node:           b6e4bacs9000/10.240.0.5
Start Time:     Tue, 07 Aug 2018 02:01:30 +1200
Labels:         app=web-booking
                pod-template-hash=143594665
Annotations:    kubernetes.io/created-by={"kind":"SerializedReference","apiVersion":"
plicaSet","namespace":"default","name":"web-booking-143594665","uid":"37b77ed7-9981-1
Status:         Pending
IP:
Created By:     ReplicaSet/web-booking-143594665
Controlled By:  ReplicaSet/web-booking-143594665
Containers:
  web-booking:
    Container ID:
    Image:          nzregslegacyacs.azurecr.io/web-booking:v1
    Image ID:
    Port:           80/TCP
    State:          Waiting
      Reason:       ContainerCreating
    Ready:          False
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-452g1 (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          False
  PodScheduled   True
Volumes:
  default-token-452g1:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-452g1
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     <none>
Events:
  Type    Reason                 Age   From                  Message
  ----    ------                 ----  ----                  -------
  Normal  Scheduled              5m    default-scheduler     Successfully assigned w
to b6e4bacs9000
  Normal  SuccessfulMountVolume  5m    kubelet, b6e4bacs9000  MountVolume.SetUp succe
ken-452g1"
  Normal  Pulling                5m    kubelet, b6e4bacs9000  pulling image "nzregsle
ing:v1"
```

Container orchestration
using kubernetes

DEMO

Authentication

Domain join inside of containers is not supported.

When Windows Authentication needed:

- Can use gMSA – group managed service account
- Must domain-join container host
- Supply extra argument to start container
- All “NETWORK SERVICE” will assume the gMSA
- Still a “roadmap” item for k8s – possible with docker swarm

Enable gMSA on Container Host

```
# ON THE CONTAINER HOSTS

## Install the gMSA

Enable-WindowsOptionalFeature -FeatureName ActiveDirectory-Powershell -online -all

Get-ADServiceAccount -Identity container_gmsa

Install-ADServiceAccount -Identity container_gmsa

Test-AdServiceAccount -Identity container_gmsa


## Create credential spec file

Invoke-WebRequest "https://raw.githubusercontent.com/Microsoft/Virtualization-Documentation/live/windows-server-container-tools/ServiceAccounts/CredentialSpec.psm1" -UseBasicParsing -OutFile $env:TEMP\cred.psm1

import-module $env:temp\cred.psm1

New-CredentialSpec -Name Gmsa -AccountName container_gmsa -Domain (Get-ADDomain -Current LocalComputer)

##This will return location and name of JSON file

Get-CredentialSpec
```

Start a container with gMSA credentials

```
docker run -d -p 8080:80 --security-opt "credentialspec=file://Gmsa.json" --name web-booking nzregslegacy.azurecr.io/web-booking:v3
```

To verify if the gMSA is working inside a container, the commands are useful:

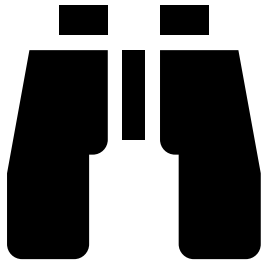
```
docker exec –it web-booking cmd
nltest /parentdomain
```

There is a good walkthrough of gMSA available here:

```
https://github.com/artisticcheese/artisticcheesecontainer/wiki/Using-Group-Managed-Service-Account-(GMSA)-to-connect-to-AD-resources
```

Whats coming up?

Running Linux Containers on Windows with LinuxKit



"Starting with Docker for Windows version 18.03.0-ce-win59 the Linux Containers on Windows (LCOW) is available as an experimental feature"

https://blogs.msdn.microsoft.com/premier_developer/2018/04/20/running-docker-windows-and-linux-containers-simultaneously/

Roadmap: Windows Server 2019

(H2 2018)

Improved container support

Kubernetes support—improvements to compute, storage and networking components of a Kubernetes cluster

Red Hat open-shift container platform

Optimized images for Server Core and Nano Server

Improved Linux support

Linux containers on Windows host

Support for tools such as open SSH, Curl, Tar

Windows Subsystem for Linux

Next steps?

If you're new to Windows containers, you might like to check out the **Windows Containers Workshop** built by my colleague Paul Bouwer for Container Camp AU & UK:

https://github.com/paulbouwer/windows-containers-workshop

If you're new to Containers and/or Kubernetes, then lookout for a "Containers Open Hack" near you. The next one is in Auckland, NZ, on November 14-16. There are also events coming up in US and Europe and there was a recent one in Sydney

http://aka.ms/NZopenhack

Move legacy apps to Windows Containers

(to take advantage of modern infrastructure and orchestration)

Regan Murphy
Software Engineer
Microsoft

@nzregs

Microsoft

NDC { Sydney }