

## Pre-Flight Checks

Call for help if needed:

- Docker Installed?
- SQL Server Management Studio (SSMS) installed?
- Git installed?
- Can use git?
- Docker pulls done?
- Will join the AWS exercise?
  - Please let Bren know

Bitlocker fix:

- Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Policies\Microsoft\FVE" -Name FDVDenyWriteAccess -Value 0
- <https://github.com/docker/kitematic/wiki/Common-Issues-and-Fixes>

```
# Clone the repo!!
```

```
$ git clone  
https://github.com/brendonmatheson/docker_fu.git
```

```
Cloning into 'docker_fu'...
```

```
$ cd docker_fu  
$ git checkout agoda_ex1
```

# DOCKER FU

BRENDON MATHESON





## Your Presenter

- Brendon Matheson
- ชีวเล่น เบรอน
- Australian - 11yr Bangkok Resident



## Currently Working On

- Healthcare (Architect at Orion Health)
- Cloud / Multi-Tenant / SaaS
- Functions-as-a-Service (FaaS)

I Know Docker Fu



# Session Plan

- Basics
  - What is Docker?
  - Docker Basics
  - Serve static content in IIS (nanoserver)
- Real
  - Dockerize a .NET Core WebAPI microservice (nanoserver & linux)
- Production
  - Multi-container solution with docker-compose (linux)
  - Multi-container solution with docker-compose (nanoserver)
  - Scheduling a cluster with Docker Swarm
- Cloud
  - Deploying to AWS ECS
- Tooling Etc
  - Docker Cloud
  - Continuous Integration
  - Kitematic

A dark, monochromatic landscape photograph. In the background, a range of jagged, rocky mountains rises against a cloudy sky. Some snow or light-colored rock patches are visible on the mountain slopes. In the foreground, a calm body of water reflects the scene. To the left, a rocky shoreline with some sparse vegetation is visible. The overall tone is somber and atmospheric.

WHAT IS DOCKER?



# What is Docker?

Packaging, deployment and execution tool

## Problems

- Environmental differences
- Complex deployment processes
- Conflicting dependencies

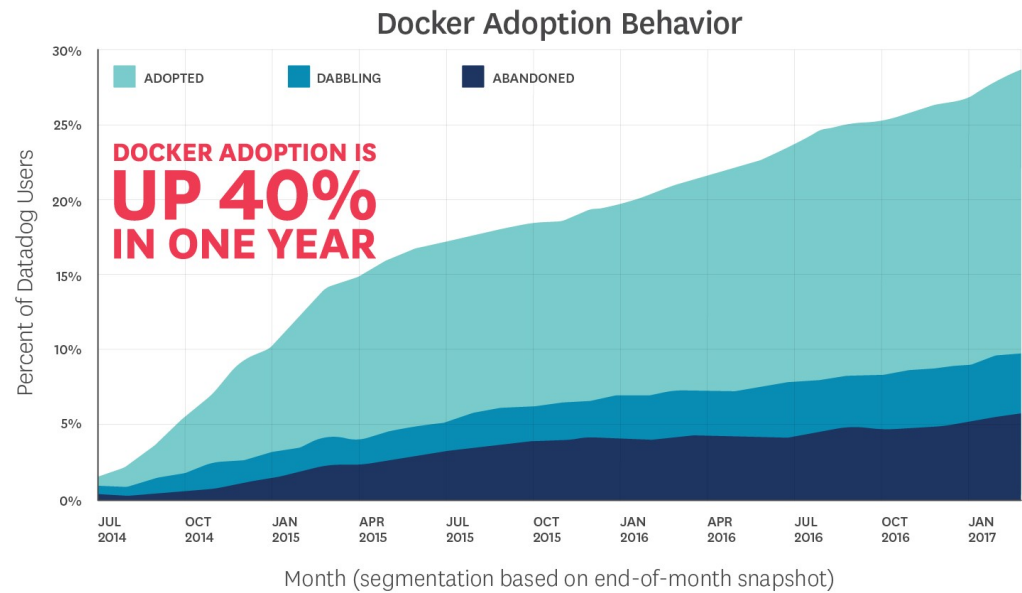
## Solution

- Process isolation
- Bundle app and dependencies into containers
- Consistency and portability





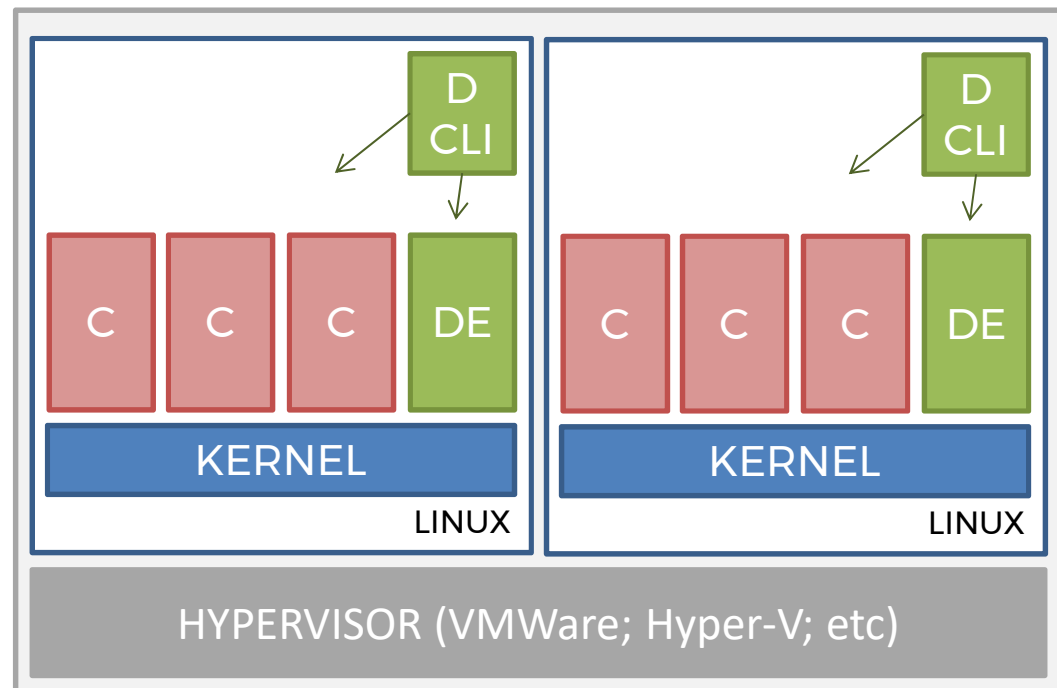
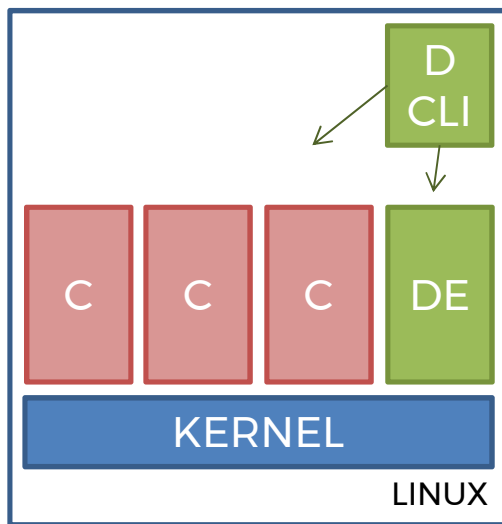
# Adoption



Source: Datadog

- Source: <https://www.datadoghq.com/docker-adoption/>

# Docker on Linux

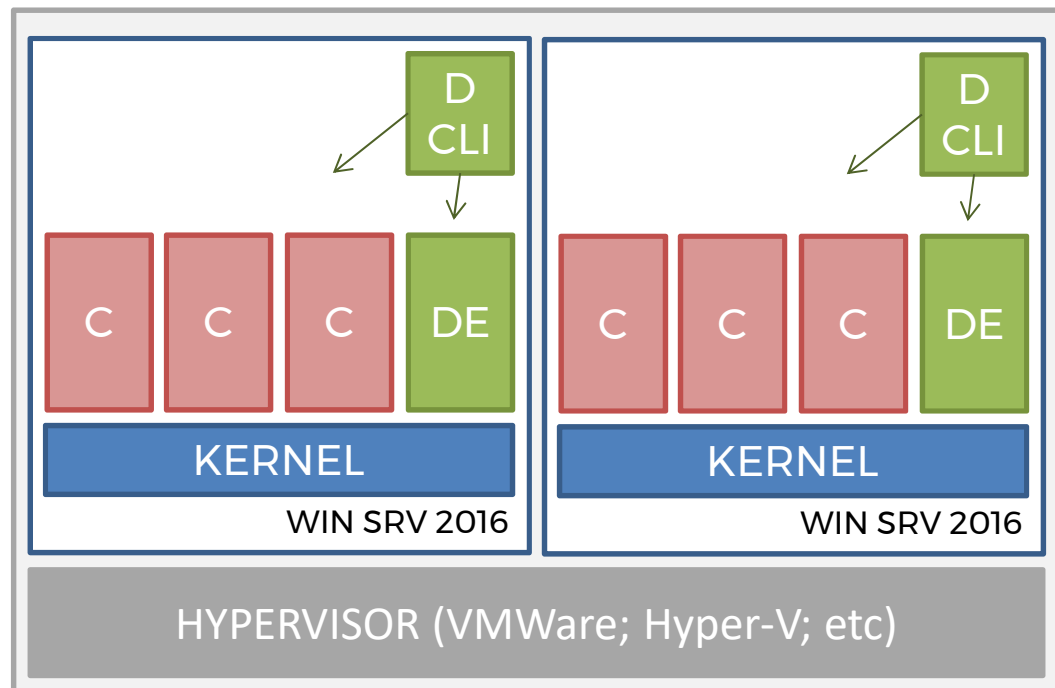
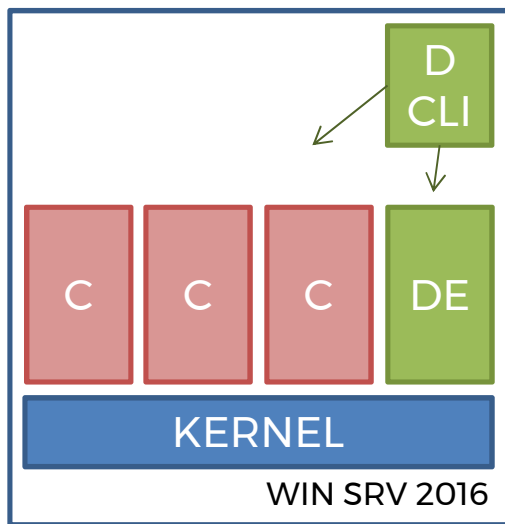


# Docker on Windows

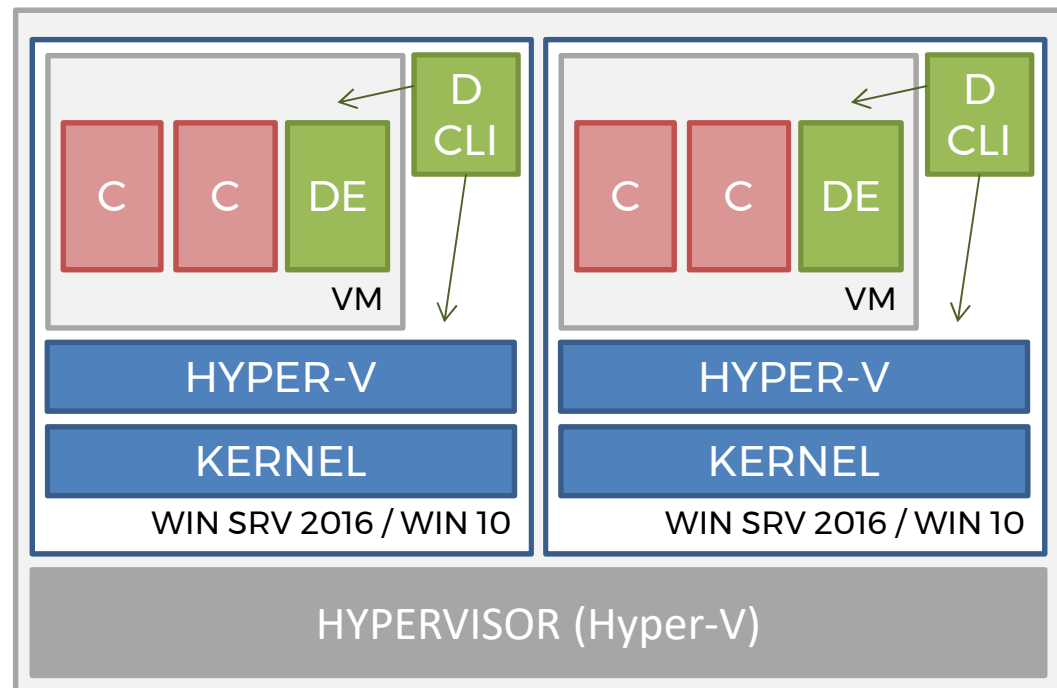
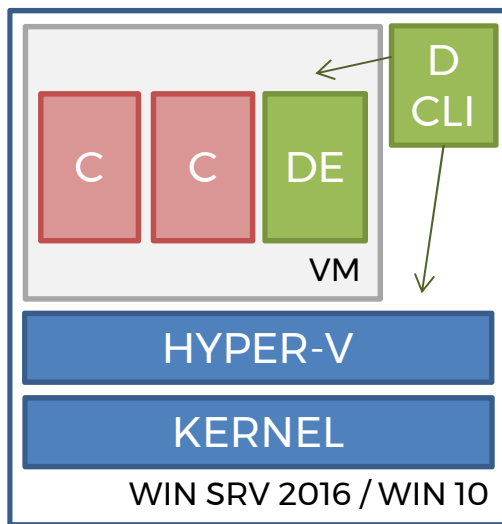
## Docker on Windows – Two Models:

- **Windows Containers** – Kernel-level support like Linux
  - Windows Server 2016
- **Hyper-V Isolation** – Virtualization-based shim
  - Windows Server 2016
  - Windows 10
    - Version 1511 / November 2016 Update / Build 10586
- References:
  - <https://docs.microsoft.com/en-us/virtualization/windowscontainers/quick-start/quick-start-windows-10>
  - <https://docs.docker.com/docker-for-windows/install/>

# Windows Containers



# Hyper-V Isolation



## Virtualization vs Containerization

Docker is a cool new virtualization technology



vmware®



# Virtualization vs Containerization

## Virtualization

- Virtual hardware
  - CPU
  - Disk
  - Memory
  - Devices
- Guest OS and software installed into VM

VM's => System-Oriented

## Containerization

- Native hardware – no hypervisor
- Host kernel is used by containerized process
- Allocate resources with control groups
- No additional OS install
- Supporting software and libraries are bundled into the container

Containers => Service-Oriented



A dark, monochromatic landscape photograph. In the background, a range of jagged, rocky mountains stretches across the horizon under a cloudy sky. In the foreground, a calm body of water reflects the light, with a rocky shoreline visible on the left. The text "DOCKER BASICS" is overlaid in the center-left in a white, sans-serif font.

# DOCKER BASICS

# hello-world

- Run it!

```
docker run hello-world
```

- Review [https://hub.docker.com/\\_/hello-world/](https://hub.docker.com/_/hello-world/)
- Pull

```
docker pull alpine:3.5
```

# hello-world

- Run an interactive session in Alpine Linux

```
docker run -i -t alpine:3.5 /bin/sh
```

- Linux kernel?
- Review Hyper-V Manager
- Run a detached nginx instance

```
docker run -d nginx:1.13-alpine
```

- Launch a shell process in the detached nginx instance

```
docker exec -it <id> /bin/sh
```

# hello-world

- **Housekeeping commands**

```
docker ps
```

```
docker stop
```

```
docker rm
```

```
docker images
```

```
docker rmi
```

```
docker container prune
```

```
docker image prune
```

# Externalities

- **Mounting file system volumes**

```
docker run -it -v W:\data:/data alpine:3.5 /bin/sh
```

- **Exposing ports**

```
docker run -it -p 8080:80 nginx:1.13-alpine
```

- **Environment variables**

```
docker run -it -e "FOO=bar" alpine:3.5 /bin/sh  
root@8e035b9c48d9:/# echo $FOO
```

A dark, moody landscape photograph of a mountain range reflected in a body of water. The mountains are rugged and rocky, with some snow patches visible. The water is calm, reflecting the mountains and the sky. The sky is dark and cloudy. The overall tone is somber and atmospheric.

# SERVE STATIC CONTENT IN IIS (NANOSERVER)

# SWITCH TO WINDOWS CONTAINERS





## Serve static content in IIS (nanoserver)

- Review the base IIS image at <https://hub.docker.com/r/microsoft/iis/>
  - Note: nanoserver vs windowsservercore
- Start with the tutorial Dockerfile:

```
FROM microsoft/iis:nanoserver-10.0.14393.1715
RUN mkdir C:\site
RUN powershell -NoProfile -Command \
    Import-module IISAdministration; \
    New-IISSite -Name "Site" -PhysicalPath C:\site -BindingInformation
    "*:8000:"
EXPOSE 8000
ADD content/ /site
```

## Serve static content in IIS (nanoserver)

- **Build**

```
docker build -t my/iis .
```

- **Run**

```
docker run --rm -it --name iis my/iis
```

- **Connect browse to <IP>:8000 – get IP from inspect:**

```
docker inspect iis
```

```
docker inspect -f "{{ .NetworkSettings.Networks.nat.IPAddress }}" iis
```

A dark, moody landscape photograph of a mountain range with a body of water in the foreground. The mountains are rugged and rocky, with some snow patches visible. The water is calm, reflecting the dark sky. The overall tone is somber and atmospheric.

# DOCKERIZE A .NET CORE WEBAPI MICROSERVICE (NANOSERVER & LINUX)

Dockerize a .NET Core WebAPI microservice (nanoserver & linux)

**Goal:** Create a ASP.NET Core WebAPI microservice that runs identically under both nanoserver and linux

SWITCH TO LINUX CONTAINERS



## Dockerize a .NET Core WebAPI microservice (nanoserver & linux)

- Review the tasksapp code
- Launch an interactive container for build

```
docker run --rm -it -v <your root path>\tasksapp\MyCo.Tasks:/build  
microsoft/aspnetcore-build:2.0.0 /bin/bash
```

- Build

```
dotnet clean  
dotnet restore  
dotnet publish -c Release -o out
```

- Exit the “build” container

# Dockerize a .NET Core WebAPI microservice (nanoserver & linux)

- Define the “runtime” container in a new Dockerfile

```
FROM microsoft/aspnetcore:2.0.0
```

```
RUN mkdir /service
```

```
WORKDIR /service
```

```
COPY MyCo.Tasks/out .
```

```
EXPOSE 80
```

```
ENTRYPOINT ["dotnet", "MyCo.Tasks.dll"]
```

- Build

```
docker build -t myco/api .
```



## Dockerize a .NET Core WebAPI microservice (nanoserver & linux)

- Run

```
docker run --rm -it --name api -p:9871:80 myco/api
```

- Browse to <http://localhost:9871/api/tasks>
- Kill the container

# Better Builds with Docker

- Dockerfile.build approach

```
FROM microsoft/aspnetcore-build:2.0.0
```

```
WORKDIR /build
```

```
RUN dotnet clean
```

```
RUN dotnet restore
```

```
RUN dotnet publish -c Release -o out
```

- Build the build container image - this actually runs the build

```
docker build -t myco/api-build -f Dockerfile.build .
```

- Retrieve the build result from the image

```
docker create --name tasks-build myco/api-build
```

```
docker cp tasks-build:/build/out .
```

```
docker rm tasks-build
```

# Better Builds with Docker

- From Docker 17.05 - multi-stage build approach in a single Dockerfile

```
# Build Stage
FROM microsoft/aspnetcore-build:2.0.0 AS tasks-build

RUN mkdir /build
WORKDIR /build
COPY MyCo.Tasks/ ./

RUN dotnet clean
RUN dotnet restore
RUN dotnet publish -c Release -o out

# Run Stage
FROM microsoft/aspnetcore:2.0.0

RUN mkdir /service
WORKDIR /service

COPY --from=tasks-build /build/out .

EXPOSE 80

ENTRYPOINT ["dotnet", "MyCo.Tasks.dll"]
```

- References

- <https://docs.docker.com/engine/userguide/eng-image/multistage-build/>
- <https://blog.alexellis.io/multi-stage-docker-builds/>

## Dockerize a .NET Core WebAPI microservice (nanoserver & linux)

- **Build and run**

```
docker build -t myco/api .  
docker run --rm -it --name api -p:9871:80 myco/api
```

- Browse to <http://localhost:9871/api/tasks>
- Kill the container

# SWITCH TO WINDOWS CONTAINERS



# Dockerize a .NET Core WebAPI microservice (nanoserver & linux)

- **Build and run**

```
docker build -t myco/api .  
docker run --rm -it --name api -p:9871:80 myco/api
```

- **Find out the container IP – due to the Windows networking NAT limitation**

```
docker inspect tasks
```

- **Browse to <http://<ip address>:80/api/tasks>**
- **Kill the container**

## Windows NAT Limitation

- Linux can NAT through the loopback interface 127.0.0.1 / localhost
- Windows cannot (yet)
  - <https://blog.sixeyed.com/published-ports-on-windows-containers-dont-do-loopback/>
  - <https://blogs.technet.microsoft.com/virtualization/2016/05/25/windows-nat-winnat-capabilities-and-limitations/>
- Port forwards DO work from outside – i.e. limitation only applies locally
- Already fixed and in preview
  - <https://blogs.technet.microsoft.com/networking/2017/11/06/available-to-windows-10-insiders-today-access-to-published-container-ports-via-localhost127-0-0-1/>



A dark, monochromatic landscape photograph. In the background, a range of jagged, rocky mountains rises against a cloudy sky. The middle ground is dominated by a calm body of water, likely a fjord or a large lake, which reflects the light from the sky. In the foreground, on the left side, there are dark, craggy rocks. The overall tone is somber and atmospheric, with a high contrast between the dark rocks and the lighter sky and water.

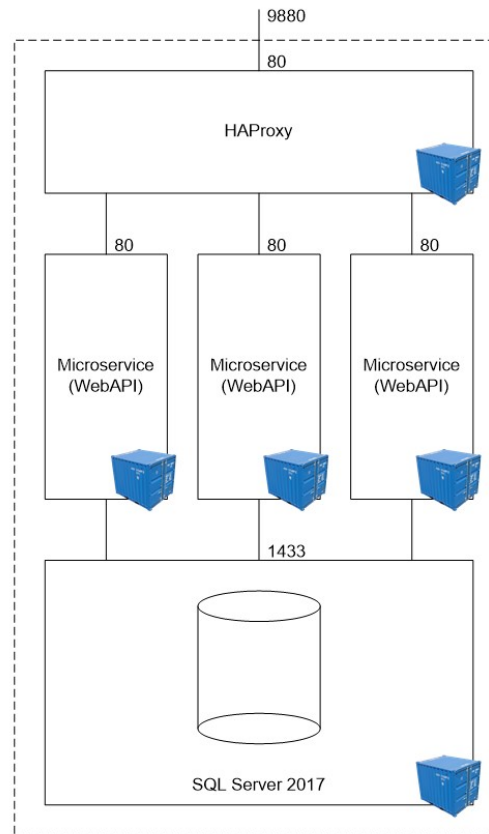
# MULTI-CONTAINER SOLUTION WITH DOCKER-COMPOSE (LINUX)

## Multi-container solution with docker-compose (linux)

- **docker-compose**
  - Create multi-container stacks
  - Define stack in a single YAML file
  - Single command to launch all containers in the stack

```
docker-compose -f mystack.yml up
```

# Multi-container solution with docker-compose (linux)



SWITCH TO LINUX CONTAINERS



# Multi-container solution with docker-compose (linux)

- **Initial docker-compose.yml**

```
version: "3"
services:

  api:
    build: .
```

- **Run and examine**

```
docker-compose up
```

- **Destroy**

```
docker-compose down
```

## Multi-container solution with docker-compose (linux)

- Map port

```
ports:  
  - "9871:80"
```

- Run again and browse <http://localhost:9870/api/tasks>

# Worker Nodes

- Build out three API worker nodes

```
version: "3"
services:

  api1:
    build: .
    ports:
      - "9871:80"

  api2:
    build: .
    ports:
      - "9872:80"

  api3:
    build: .
    ports:
      - "9873:80"
```

# Worker Nodes

- Explicitly label the images built by docker-compose:

```
version: "3"
services:

  api1:
    image: myco/api
    build: .
    ports:
      - "9871:80"

  api2:
    image: myco/api
    build: .
    ports:
      - "9872:80"

  api3:
    image: myco/api
    build: .
    ports:
      - "9873:80"
```



## Worker Nodes

- Browse to
  - <http://localhost:9871/api/tasks>
  - <http://localhost:9872/api/tasks>
  - <http://localhost:9873/api/tasks>

- Stop the stack

```
docker-compose -f docker-compose.lin.yml down
```

# Load Balancer

- **Add haproxy service**

```
haproxy:
  image: library/haproxy:1.7
  volumes:
    - haproxy_cfg:/usr/local/etc/haproxy
  ports:
    - "9880:80"
    - "9881:70"
  links:
    - api1
    - api2
    - api3
```

# Load Balancer

- **Declare the volume**

```
volumes:  
  haproxy_cfg:  
    external: true
```

- **Launch – observe volume error**

```
docker-compose up
```

```
ERROR: Volume haproxy_cfg declared as external, but could not be found.  
Please create the volume manually using `docker volume create --  
name=haproxy_cfg` and try again.
```

## Load Balancer

- Create the volume to hold the haproxy configuration:

```
docker volume create haproxy_cfg
```

- Import the haproxy.cfg file into the volume via a temporary container:

```
scripts\haproxy_copy_config.cmd
```

# Load Balancer

- Run and observe round-robin load balancing

```
docker-compose up
```

- Browse to <http://localhost:9880/api/tasks>
- Each node has its own in-mem data, so each refresh will be different
- Drop the worker-node ports

# Load Balancer

- Create `docker-compose.lin.dev.yml` to mixin worker-node ports for debugging

```
version: "3"
services:

  api1:
    ports:
      - "9871:80"

  api2:
    ports:
      - "9872:80"

  api3:
    ports:
      - "9873:80«
```

- Launch stack with ports mixed in

```
docker-compose -f docker-compose.lin.yml -f docker-compose.lin.dev.yml up
```

## Add SQL Server 2017

- Create a data directory - e.g mine is W:\data
- Launch a SQL Server 2017 instance - in Docker!

```
docker run ^  
    -e "ACCEPT_EULA=Y" ^  
    -e "MSSQL_SA_PASSWORD=p@ssw0rz!@#" ^  
    -p 1401:1433 ^  
    -v W:\data:/var/opt/mssql ^  
    --name sql1 ^  
    -d ^  
    microsoft/mssql-server-linux:2017-GA
```

- Shortcut scripts\start\_sql\_linux.cmd

# Add SQL Server 2017

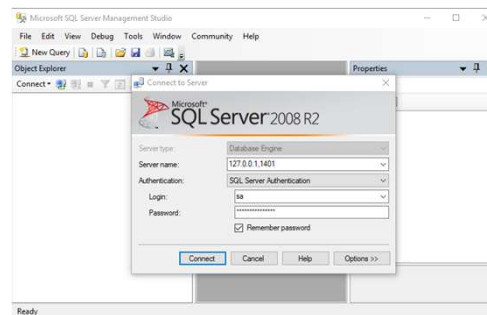
- Edit the microservice config to connect to a database
- Create the database on a temporary SQL Server instance
  - Create a volume for the data files

```
docker volume create tasks-db
```

- Review and run the script:

```
scripts\start_sql_linux.cmd
```

- Connect via SSMS





## Add SQL Server 2017

- Create a new database with the script at Database\tasks\_database.sql
- Terminate SQL Server

```
docker stop sql1
```

## Add SQL Server 2017

- Add a new “db” service to docker-compose.lin.yml:

```
db:
  image: microsoft/mssql-server-linux:2017-GA
  environment:
    ACCEPT_EULA: "Y"
    MSSQL_SA_PASSWORD: "p@ssw0rz!@#"
  volumes:
    - tasks-db:/var/opt/mssql
  expose:
    - "1433"
  ports:
    - "1402:1433"
```

## Add SQL Server 2017

- Declare the external volume

```
volumes:  
  haproxy_cfg:  
    external: true  
tasks-db:  
  external: true
```

# Full System in One File

- Launch the system

`docker-compose -f docker-compose.lin.yml up`

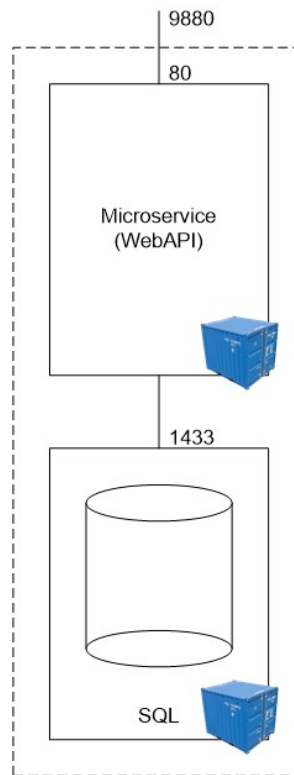
```
W:\wrk\bjm_str_px_docker_dotnet\tasksapp>docker-compose -f docker-compose.lin.yml up
Creating network "tasksapp_default" with the default driver
Creating tasksapp_api1_1 ...
Creating tasksapp_db_1 ...
Creating tasksapp_api2_1 ...
Creating tasksapp_api3_1 ...
Creating tasksapp_api1_1
Creating tasksapp_db_1
Creating tasksapp_api3_1
Creating tasksapp_api2_1 ... done
Creating tasksapp_haproxy_1 ...
Creating tasksapp_haproxy_1 ... done
Attaching to tasksapp_api1_1, tasksapp_db_1, tasksapp_api3_1, tasksapp_api2_1, tasksapp_haproxy_1
api1_1 | warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]
api1_1 |   Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be persisted outside of the container. Protected data will
api1_1 |   be unavailable when container is destroyed.
api1_1 | warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
api1_1 |   No XML encryptor configured. Key {da21549d-c2b1-488c-87fe-700afc406d79} may be persisted to storage in unencrypted form.
api1_1 |   Hosting environment: Production
api1_1 |   Content root path: /service
api3_1 | warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]
api1_1 |   Now listening on: http://[::]:80
api3_1 |   Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be persisted outside of the container. Protected data will
api3_1 |   be unavailable when container is destroyed.
api2_1 | warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]
api1_1 |   Application started. Press Ctrl+C to shut down.
api3_1 | warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
api2_1 |   Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be persisted outside of the container. Protected data will
api2_1 |   be unavailable when container is destroyed.
haproxy_1 | <?haproxy-systemd-wrapper: executing /usr/local/sbin/haproxy -p /run/haproxy.pid -f /usr/local/etc/haproxy/haproxy.cfg -Ds
api3_1 |   No XML encryptor configured. Key {cc848d5a-c3ac-4973-b3f2-ac9f2218acf9} may be persisted to storage in unencrypted form.
api1_1 | warn: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[35]
```

Browse to <http://localhost:9880/api/tasks>

A dark, monochromatic landscape photograph. In the background, a range of jagged, rocky mountains rises against a cloudy sky. In the foreground, a calm body of water reflects the light, with some rocky terrain visible on the left side. The overall tone is dark and atmospheric.

# MULTI-CONTAINER SOLUTION WITH DOCKER-COMPOSE (NANOSERVER)

# Multi-container solution with docker-compose (nanoserver)



# SWITCH TO WINDOWS CONTAINERS



## Create Database

- Create the managed volume

```
docker volume create tasks-db
```

- Launch SQL Server 2017 on Windows via helper script

```
scripts\start_sql_win.cmd
```



## Define Single Worker Node

```
version: "3"
services:

  api:
    build: .
    image: myco/api
    environment:
      ASPNETCORE_ENVIRONMENT: "Production"
    ports:
      - "9871:80"
```

# Define Database

- Database service:

```
db:
  image: microsoft/mssql-server-windows-developer:2017
  environment:
    ACCEPT_EULA: "Y"
    SA_PASSWORD: "p@ssw0rz!@#"
    ATTACH_DBS: "[{'dbName': 'Tasks', 'dbFiles': ['C:\\\\data\\\\Tasks.mdf',
'C:\\\\data\\\\Tasks.ldf']}]]"
  volumes:
    - "tasks-db:C:\\data"
  ports:
    - "1401:1433"
```

- Declare volume

```
volumes:
  tasks-db:
    external: true
```

## Launch Stack

```
docker-compose -f docker-compose.win.yml up
```

```
W:\wrk\bjm_str_px_docker_dotnet\tasksapp>docker-compose -f docker-compose.win.yml up
Creating tasksapp_api_1 ...
Creating tasksapp_db_1 ...
Creating tasksapp_api_1
Creating tasksapp_db_1 ... done
Attaching to tasksapp_api_1, tasksapp_db_1
api_1 | Hosting environment: Production
api_1 | Content root path: C:\service
api_1 | Now listening on: http://[::]:80
api_1 | Application started. Press Ctrl+C to shut down.
db_1 | VERBOSE: Starting SQL Server
db_1 | VERBOSE: Changing SA login credentials
db_1 | VERBOSE: Attaching 1 database(s)
db_1 | VERBOSE: Invoke-Sqlcmd -Query IF EXISTS (SELECT 1 FROM SYS.DATABASES WHERE NAME
db_1 | = 'Tasks') BEGIN EXEC sp_detach_db [Tasks] END;CREATE DATABASE [Tasks] ON
db_1 | (FILENAME = N'C:\data\Tasks.mdf'),(FILENAME = N'C:\data\Tasks.ldf') FOR ATTACH;
db_1 | VERBOSE: Started SQL Server.
db_1 |
```

## Multi-Container Solution Further Reading

- SQL Server for Linux
  - <https://docs.microsoft.com/en-us/sql/linux/sql-server-linux-release-notes>

A dark, monochromatic landscape photograph of a mountain range with a body of water in the foreground. The text "CLUSTERS AND SCHEDULERS" is overlaid in white.

# CLUSTERS AND SCHEDULERS

## Clusters and Schedulers

- A scheduler automatically deploys and runs containers according to your specifications
- Health-checks and auto-healing
- Higher-order conceptual unit based on containers
- Many options
  - Kubernetes
    - Rancher
  - Swarm
  - Mesos
  - DC/OS
  - AWS ECS

A dark, monochromatic landscape photograph. In the background, a range of jagged, rocky mountains rises against a cloudy sky. The middle ground is filled with a calm body of water, which reflects the light from the sky. In the foreground, the dark, silty banks of the water are visible, with some rocks and sparse vegetation. The overall tone is somber and atmospheric.

# SCHEDULING A CLUSTER WITH DOCKER SWARM

## Scheduling a cluster with Docker Swarm

- Docker Swarm is a Docker-native clustering system that exposes the same API as the standalone Docker Engine
- Health checks
- Launch a set number of containers and scale up or down
- Rolling updates
- References:
  - <https://docs.docker.com/compose/swarm/>



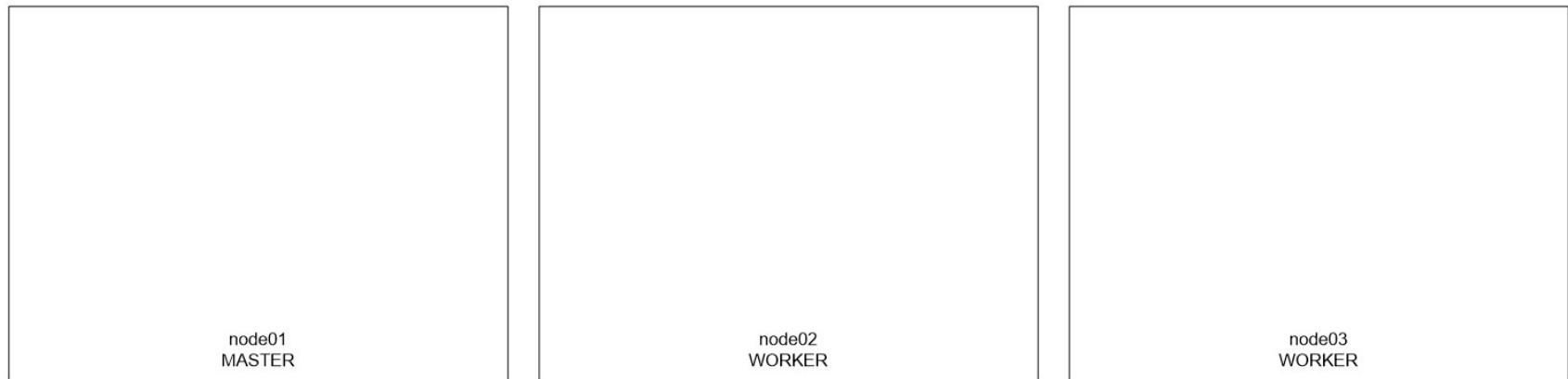
## Plan

- Multi-Node Swarm – hands-off due to complexity
  - Manual – Step-by-Step
  - Automated with Compose
- Single-Node Swarm – hands-on
- Try the multi-node swarm yourself after our workshop

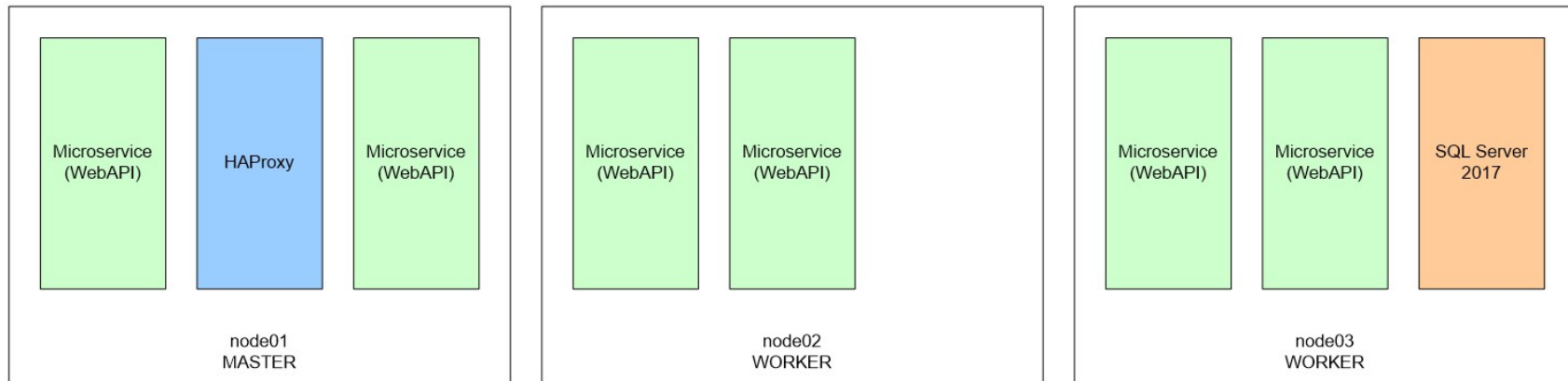


# MULTI-NODE SWARM

# Physical Architecture



# Physical Architecture



# Multi-Node Swarm Manual Setup – Build Cluster

- Create an external vswitch in Hyper-V named “External”
- Create a cluster using docker-machine:

```
docker-machine create --driver hyperv --hyperv-virtual-switch "External" manager1
docker-machine create --driver hyperv --hyperv-virtual-switch "External" worker1
docker-machine create --driver hyperv --hyperv-virtual-switch "External" worker2
```

- Reconfigure manager1 to have 4GB RAM since we'll run SQL Server on that later
- Check machines and note their IP's:

```
docker-machine ls
```

- Turn these separate nodes into a Swarm cluster

```
docker-machine ssh manager1
docker swarm init --advertise-addr <manager1's ip>
docker-machine ssh worker1/2 # paste the join command from manager1
```

- Use docker-machine to point local Docker CLI at manager1

```
docker-machine env manager1
```

## Multi-Node Swarm Manual Setup – Trial API Workers

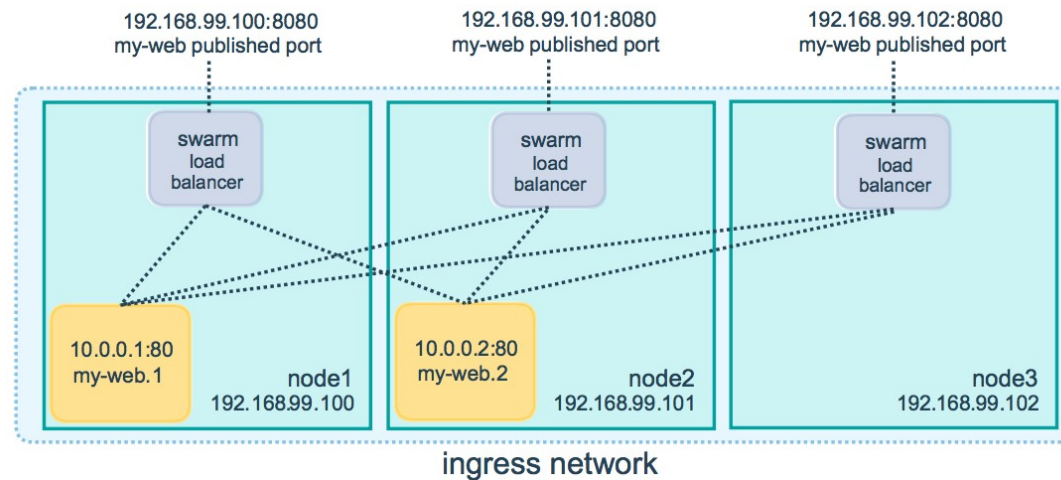
- Build and push the in-mem version of api for Linux
- Launch the api service with a single replica

```
docker service create -d --name api -p 9870:80 --replicas 1  
brendonmatheson/api:lin
```

- Browse the running service
- Review networking and ingress routing

```
docker service ls  
docker service inspect tasks  
docker ps  
docker inspect <container_id>
```

# Ingress Routing



- Reference:
  - <https://docs.docker.com/engine/swarm/ingress/#publish-a-port-for-a-service>

## Multi-Node Swarm Manual Setup – Trial API Workers

- Update the service to have 4 replicas

```
docker service update --replicas 4 tasks  
docker scale tasks=4
```

- Review again
- Kill a machine observe Swarm react
- Restore the machine and force a rebalance

```
docker service update --force
```

- Tear down the service

```
docker service rm tasks
```



## Multi-Node Swarm Manual Setup – Create Database

- Create the volume to hold the database files

```
docker volume create tasks-db
```

- Launch a temporary SQL Server instance and create the Tasks database

```
scripts\start_sql_lin.cmd
```

- Connect via SSMS and use db\tasks\_database.lin.sql to create the database
- Throw away the temporary SQL Server instance:

```
docker stop sql1
```

# Multi-Node Swarm Manual Setup – DB Service

- Create a network for API to talk to database

```
docker network create --driver overlay tasksbackend
```

- Launch database service on Swarm connected only to backend network

```
docker service create \  
  -d \  
  -e "ACCEPT_EULA=Y" \  
  -e "MSSQL_SA_PASSWORD=p@ssw0rz@#" \  
  --network tasksbackend \  
  --mount type=volume,source=tasks-db,destination=/var/opt/mssql \  
  --name db \  
  --replicas 1 \  
  --constraint "node.role==manager" \  
  microsoft/mssql-server-linux:2017-GA
```

- References:

- [https://docs.docker.com/engine/reference/commandline/service\\_create/#specify-service-constraints-constraint](https://docs.docker.com/engine/reference/commandline/service_create/#specify-service-constraints-constraint)

## Multi-Node Swarm Manual Setup – API Service

- Build and push the database version of api for Linux
- Launch the api service with four replicas and connected to the backend network

```
docker service create \  
    -d \  
    --name api \  
    -p 9870:80 \  
    --network tasksbackend \  
    --replicas 4 \  
    brendonmatheson/api:lin
```

- Browse the running service
- Review networking and routing again
- Shell into one of the API containers to further examine VIP's and DNS resolution

# Multi-Node Swarm – Automated with Compose

```
version: '3'
services:
  db:
    image: microsoft/mssql-server-linux:2017-GA
    environment:
      ACCEPT_EULA: "Y"
      MSSQL_SA_PASSWORD: "p@ssw0rz@#"
    volumes:
      - tasks-db:/var/opt/mssql
    networks:
      - tasksbackend
    deploy:
      replicas: 1
      placement:
        constraints:
          - "node.role==manager"

volumes:
  tasks-db:
    external: true

networks:
  tasksbackend:
    driver: overlay
```

# Multi-Node Swarm – Automated with Compose

```
api:
  image: brendonmatheson/api:lin
  ports:
    - "9870:80"
  networks:
    - tasksbackend
  deploy:
    replicas: 5
```

- **Deploy the stack:**

```
docker stack deploy --compose-file docker-compose.stack.yml ts
```

- **Reference:**

- <https://docs.docker.com/engine/swarm/stack-deploy/>



# SINGLE-NODE SWARM

SWITCH TO LINUX CONTAINERS



## Single-Node Swarm

- Create your single-node swarm

```
docker swarm init
```

- Deploy the stack – identical config to our multi-node exercise!

```
docker stack deploy --compose-file docker-compose.stack.lin.yml tasks
```

- Inspect the components of the running stack and make sure you understand them



## Swarm Further Reading

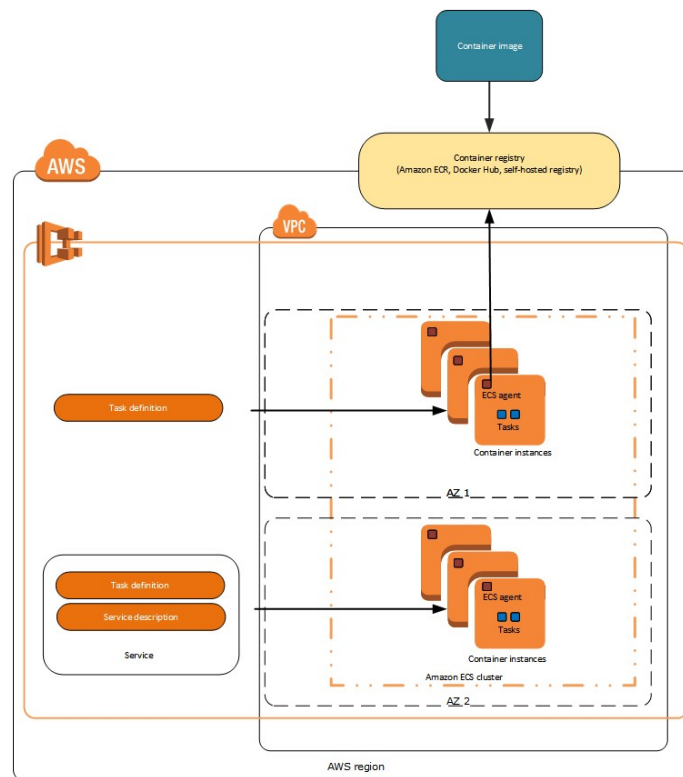
- Rolling Updates -  
<https://docs.docker.com/engine/swarm/swarm-tutorial/rolling-update/>

A dark, monochromatic landscape photograph. In the background, a range of jagged, rocky mountains rises against a cloudy sky. Some snow or light-colored patches are visible on the mountain slopes. In the foreground, a calm body of water reflects the scene. To the left, a rocky shoreline with some sparse vegetation is visible. The overall tone is somber and atmospheric.

# DEPLOYING TO AWS ECS

## Deploying to AWS ECS

- **Registry** - ECR
  - Host our images on AWS
  - Alternative to Docker Hub
- **Cluster**
  - Set of virtual machines for running container workloads
  - Launch from wizard – runs CloudFormation template
- **Task**
  - Image plus execution meta-data
- **Service**
  - Clustered set of instances of a Task



A dark, monochromatic landscape photograph. In the background, a range of jagged, rocky mountains stretches across the horizon. The central peak is particularly prominent, with a sharp, pointed summit. Patches of snow or light-colored rock are visible on the mountain slopes. In the foreground, a calm body of water reflects the dark sky and the silhouettes of the mountains. The water's surface shows subtle ripples. On the left side, a rocky shoreline with some sparse vegetation is visible. The overall mood is somber and majestic. The text 'TOOLING ETC' is superimposed in white, sans-serif capital letters on the left side of the image, over the water and the lower part of the mountains.

TOOLING ETC



DOCKER CLOUD

# Docker Cloud






The screenshot shows the Docker Cloud onboarding dashboard in a web browser. The browser's address bar displays the URL <https://cloud.docker.com/dashboard/onboarding>. The page features a blue header with the Docker Cloud logo, a 'Back to classic UI' link, a '+ Get Help' button, and a user profile for 'nigelpoulton'. A 'Welcome!' banner is positioned below the header. The main content area is titled 'Welcome to Docker Cloud!' and 'Let's get you familiarized with the central concepts of Docker Cloud.' It contains five interactive cards: 'LINK PROVIDER' (cloud icon), 'CREATE A NODE' (server icon), 'CREATE A SERVICE' (stack of containers icon), 'CREATE A STACK' (stack of lines icon), and 'REPOSITORIES' (document icon). Each card includes a brief definition. The 'REPOSITORIES' card is highlighted with a mouse cursor. At the bottom, a link points to 'Looking for the new Docker Security Scanning? Click here'.

DOCKER CLOUD [Back to classic UI](#) + Get Help nigelpoulton

Welcome!

Welcome to Docker Cloud!

Let's get you familiarized with the central concepts of Docker Cloud.

LINK PROVIDER	CREATE A NODE	CREATE A SERVICE	CREATE A STACK	REPOSITORIES
				
<i>Link to a hosted Cloud Services Provider like DigitalOcean or AWS.</i>	<i>A <b>node</b> is a Linux host or virtual machine used to deploy and run containers.</i>	<i>A <b>service</b> is a container, or a group of containers from the same Docker repository. You can add more containers to scale an app across nodes.</i>	<i>A <b>stack</b> specifies a group of services that make up an application, similar to Docker Compose.</i>	<i>A <b>repository</b> is a collection of tagged images. When you create a service, you choose an image to use to create containers.</i>

Looking for the new Docker Security Scanning? [Click here](#)

# Docker Cloud

The screenshot shows the Docker Cloud account management interface. The browser address bar displays `https://cloud.docker.com/account`. The left sidebar contains navigation links: General, Cloud providers, Source providers, API keys, Notifications, Billing, Plan, and Quotas. The main content area is divided into three sections: a password confirmation section at the top, a 'Cloud providers' table in the middle, and a 'Source providers' table at the bottom.

Confirm new password

[I forgot my password](#)

[Save](#)

### Cloud providers

Provider	Account	Refresh	Remove	Tier
Amazon Web Services	AKIAIDZL73D7M5TVLDHQ			<a href="#">Free Tier</a>
Digital Ocean	<a href="#">Add new credentials</a>			<a href="#">\$20 Code</a>
Microsoft Azure	<a href="#">Add new credentials</a>			<a href="#">Free trial</a>
SoftLayer	<a href="#">Add new credentials</a>			<a href="#">Free trial</a>
Packet	<a href="#">Add new credentials</a>			<a href="#">\$25 code</a>

### Source providers

Provider	Account	Refresh	Remove
Github	No account linked		





# CONTINUOUS INTEGRATION WITH DOCKER

## Continuous Integration with Docker

- Naive approach
  - Add a docker build to the end of your build configuration
- Better approach
  - Use multi-stage Builds for your whole build process
  - CI just runs docker build and docker push
  - Repeatable locally
- Also
  - Run build infrastructure on Docker
  - `docker run -v /var/run/docker.sock:/var/run/docker.sock`



KITEMATIC

Kitematic

**KITEMATIC™**

Kitematic

The logo for Kitematic, featuring the word "KITEMATIC" in a bold, white, sans-serif font. The letter "K" is stylized with a vertical line through its center. A small "TM" trademark symbol is located at the top right of the letter "C". The logo is centered on a solid blue rectangular background.

**KITEMATIC<sup>TM</sup>**

- GUI Tool for Mac and Windows
- May be useful for simple cases when learning

# Brens Recommended Docker Learning Curve

- Single-Container Use-Cases
  - Services
    - CI system
      - <https://hub.docker.com/r/jetbrains/teamcity-server/>
      - <https://hub.docker.com/r/jetbrains/teamcity-agent/>
      - [https://hub.docker.com/\\_/jenkins/](https://hub.docker.com/_/jenkins/)
    - VCS – gitlab; git+ssh
      - <https://hub.docker.com/r/gitlab/gitlab-ce/>
    - JetBrains Upsource - <https://hub.docker.com/r/jetbrains/upsource/>
    - SonarQube - [https://hub.docker.com/\\_/sonarqube/](https://hub.docker.com/_/sonarqube/)
    - Plex(!) - <https://github.com/plexinc/pms-docker>
  - Dev / test environments
    - Build configurations
- Study Networking!
  - Networking - <https://docs.docker.com/engine/userguide/networking/>
  - Embedded DNS - <https://docs.docker.com/engine/userguide/networking/configure-dns/>
- Clustered Use-Cases – Your Apps
  - Test / Staging environments
  - Production environments




# I want more!


- Basics
  - What is Docker?
  - Docker Basics
  - .NET Core hello-world in Docker (nanoserver & linux)
  - Serve static content in IIS (nanoserver)
  - Visual Studio .NET Framework hello-world (linux / windowsservercore)
  - Dockerize a .NET Framework hello-world (nanoserver)
- Real
  - Dockerize nginx and CIFS (linux)
  - Dockerize a .NET Core WebAPI microservice (nanoserver & linux)
  - Dockerize a .NET Framework WebAPI microservice (windowsservercore)
- Production
  - Multi-container solution with docker-compose (linux)
  - Multi-container solution with docker-compose (nanoserver)
  - Clusters and Scheduling
  - Private Docker Registry
  - Scheduling a cluster with Docker Swarm
  - Scheduling a cluster with Kubernetes
  - Deploying across heterogeneous container engines
- Cloud
  - Deploying to AWS ECS
  - Deploying on Azure ACS
  - Deploying to Google Container Engine
  - Deploying to Digital Ocean
- Tooling Etc
  - Docker Cloud
  - Continuous Integration
  - Testing Containers
  - Visual Studio 2017 Support for Docker
  - Kitematic
  - Customize the dotnetcore SDK container

Follow YouTube (<http://u.bren.cc/youtube>) and Twitter (<http://u.bren.cc/twitter>) for new content:



 b@bren.cc

 <http://u.bren.cc/github>

 brendon.matheson

 <http://u.bren.cc/linkedin>

 <http://u.bren.cc/youtube>

 <http://u.bren.cc/twitter>

