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
- ชื่อเล่น เบรน
- 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


WHAT IS DOCKER?

What is Docker?

Packaging, deployment and execution tool

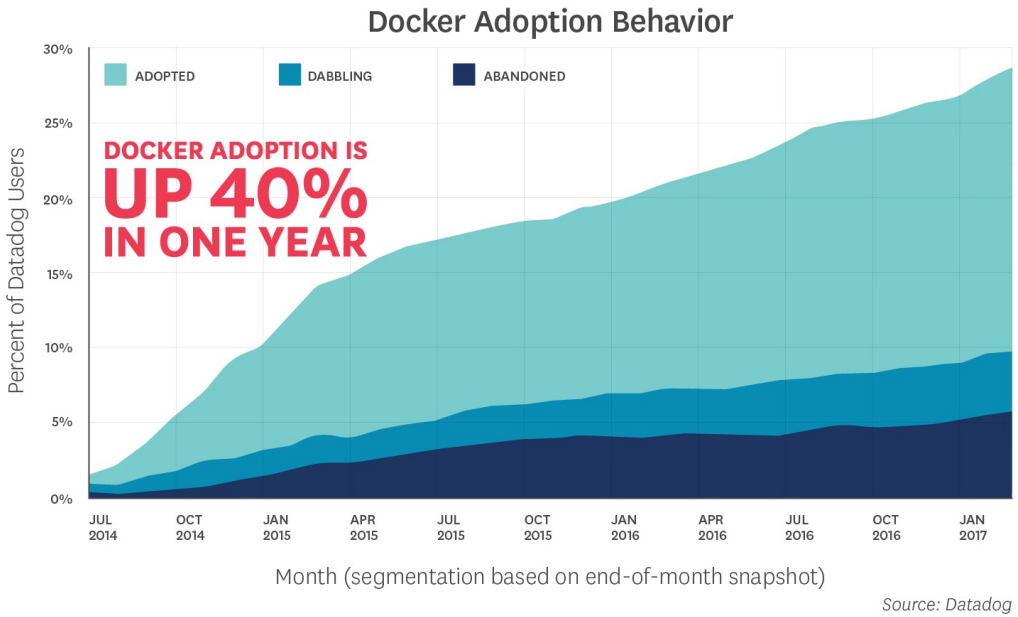
Problems

- Environmental differences
- Complex deployment processes
- Conflicting dependencies

Solution

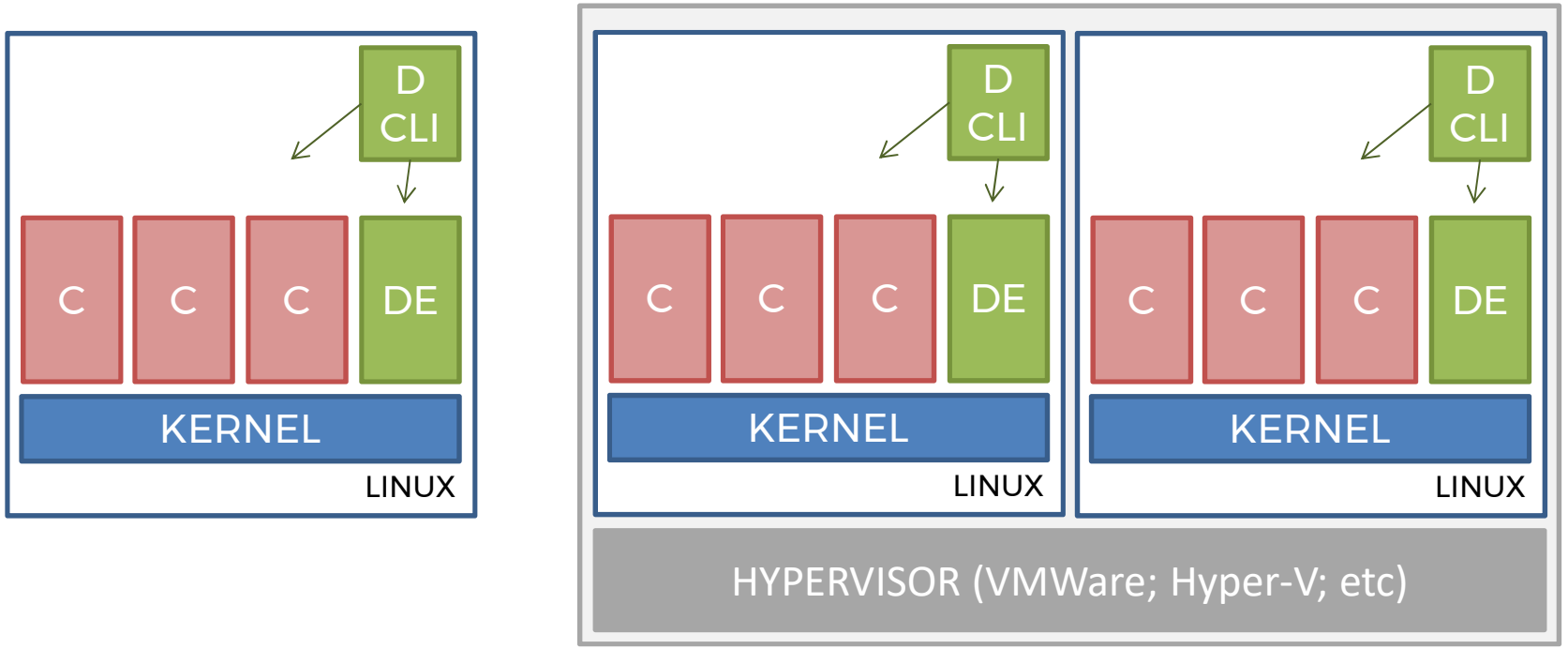
- Process isolation
- Bundle app and dependencies into containers
- Consistency and portability

Adoption



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

Docker on Linux

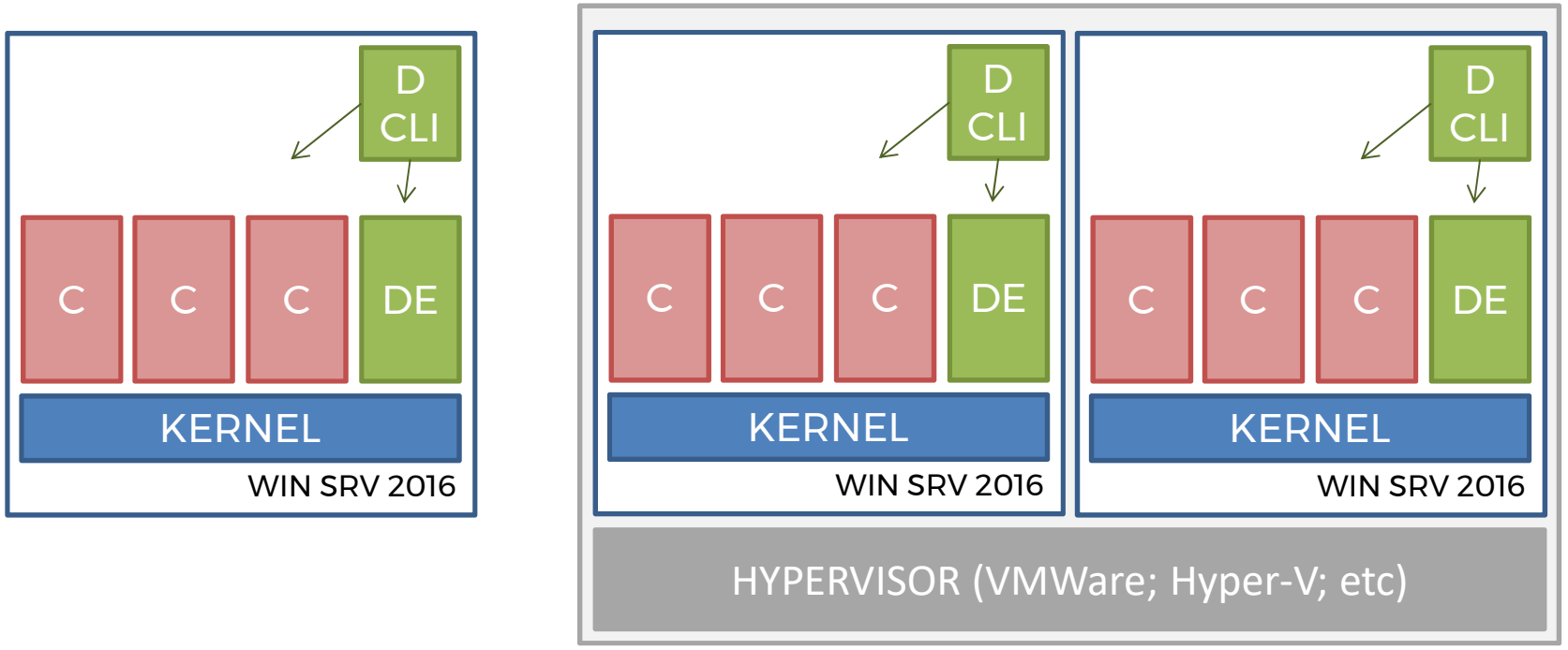


Docker on Windows

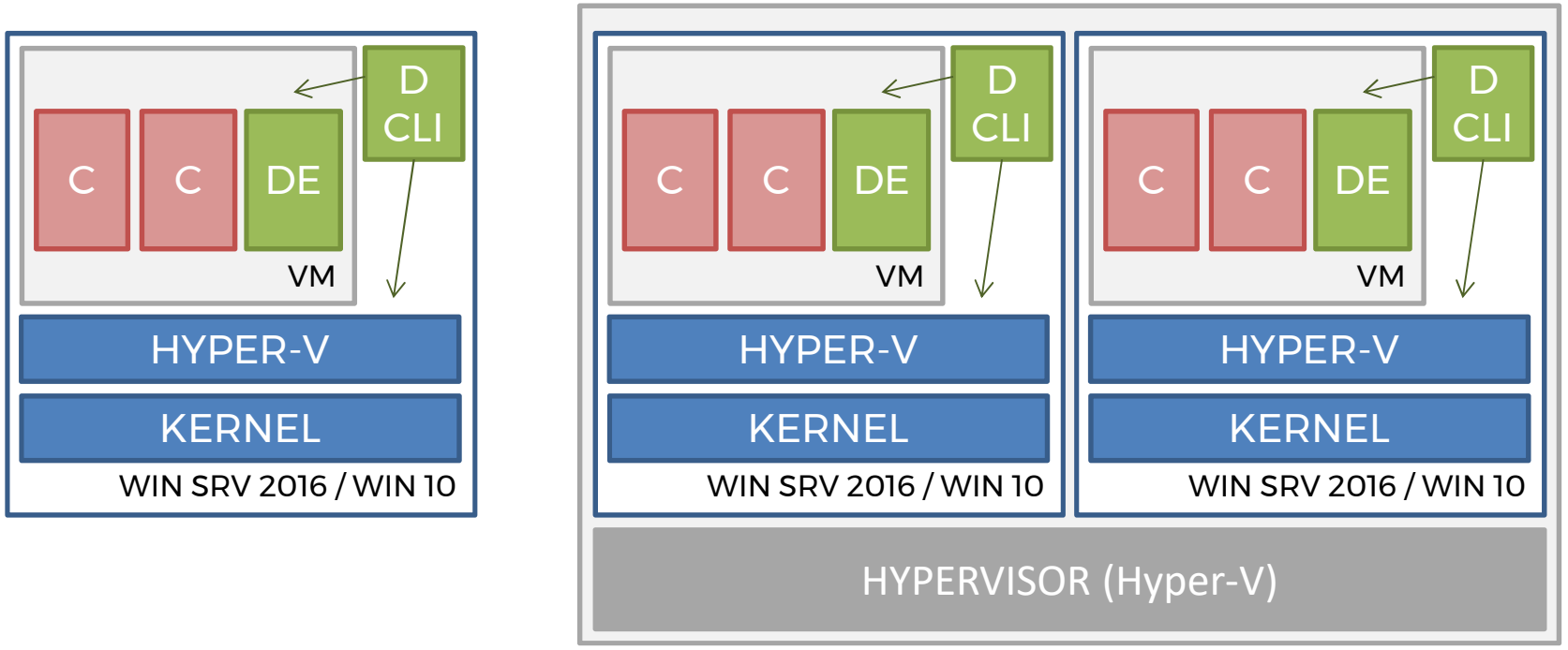
Docker on Windows – Two Models:

- **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 n virtualization techno y

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


DOCKER BASICS

hello-world

- Run it!

```
docker run hello-world
```

- Review 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
```


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
```

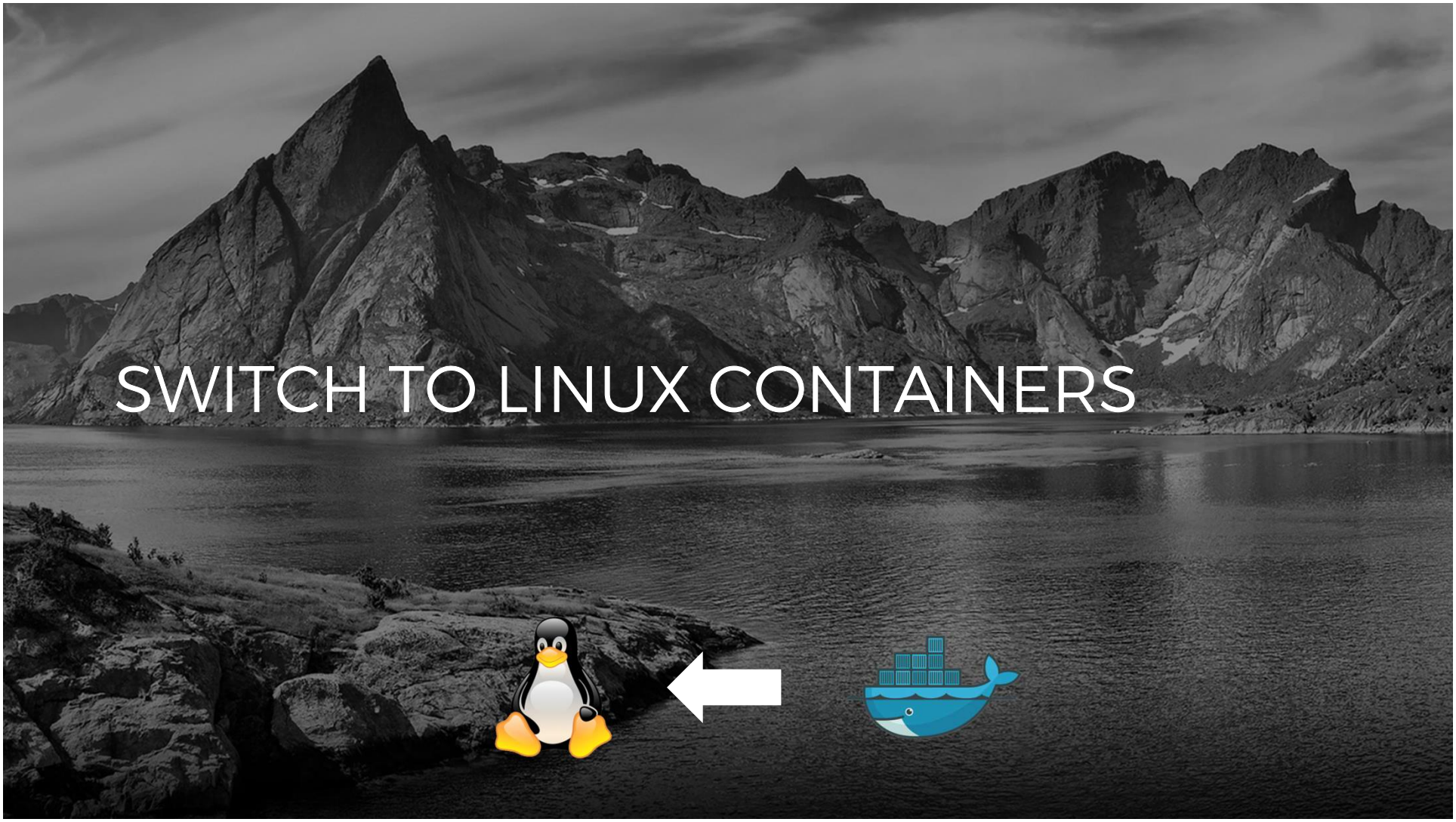
- Connect browse to <IP>:8000 – get IP from inspect:

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


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/mutli-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/

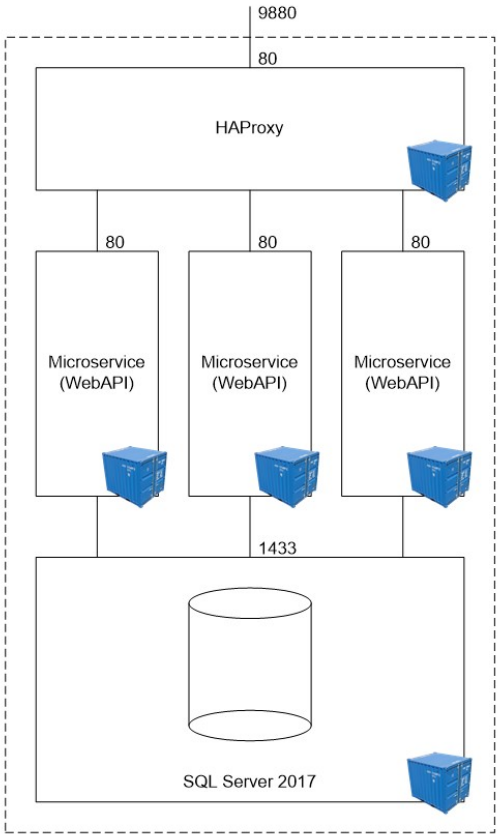

MULTI-CONTAINER SOLUTION
WITH DOCKER-COMPOSE (LINUX)

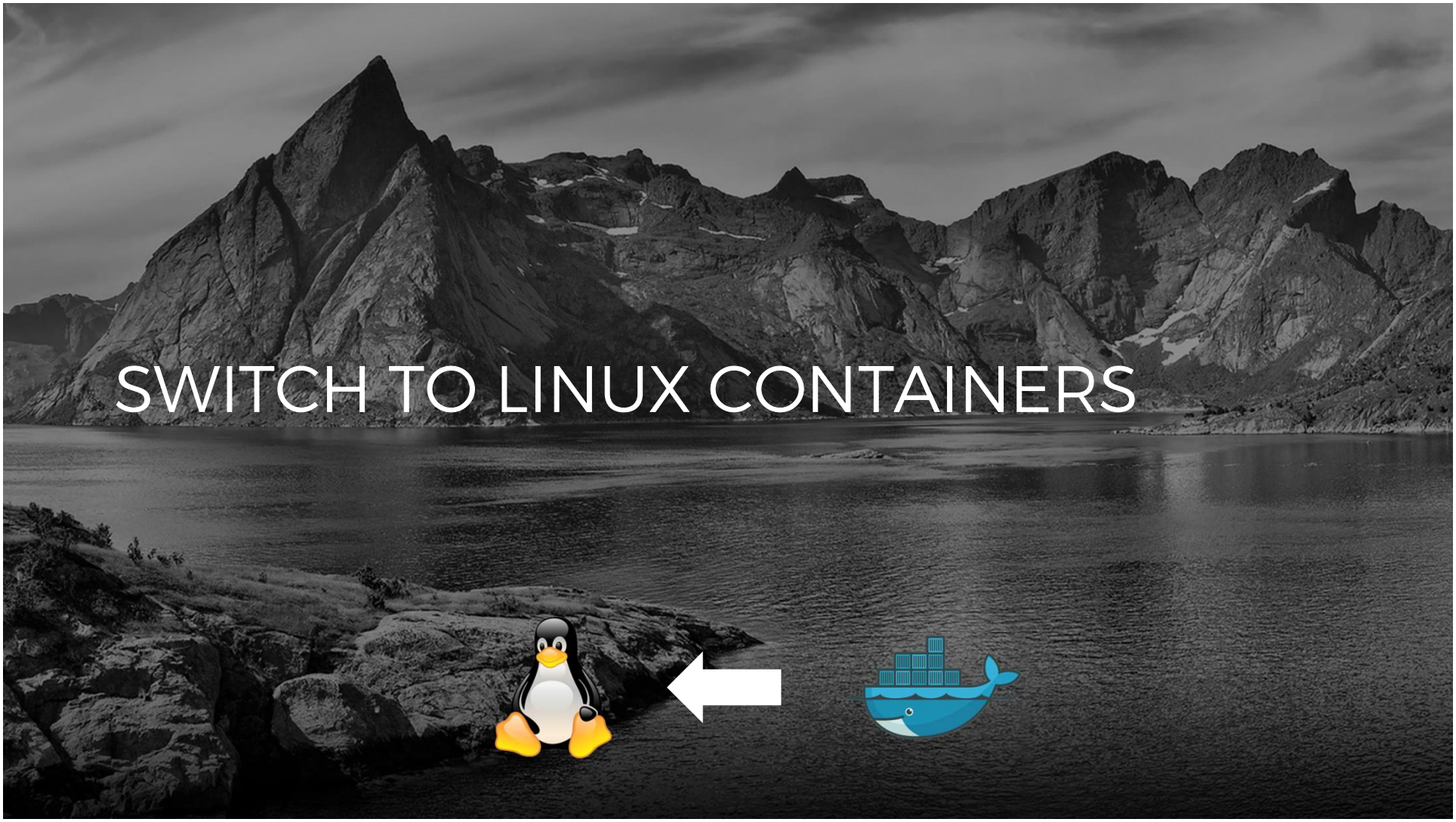
Multi-container solution with docker-compose (linux)

- 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

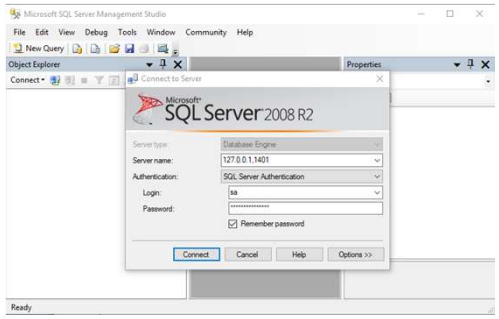
- Edit the microservice config to connect to a database
- Create the database on a temporary SQL Server instance
  - Create a volume for the data files

```
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

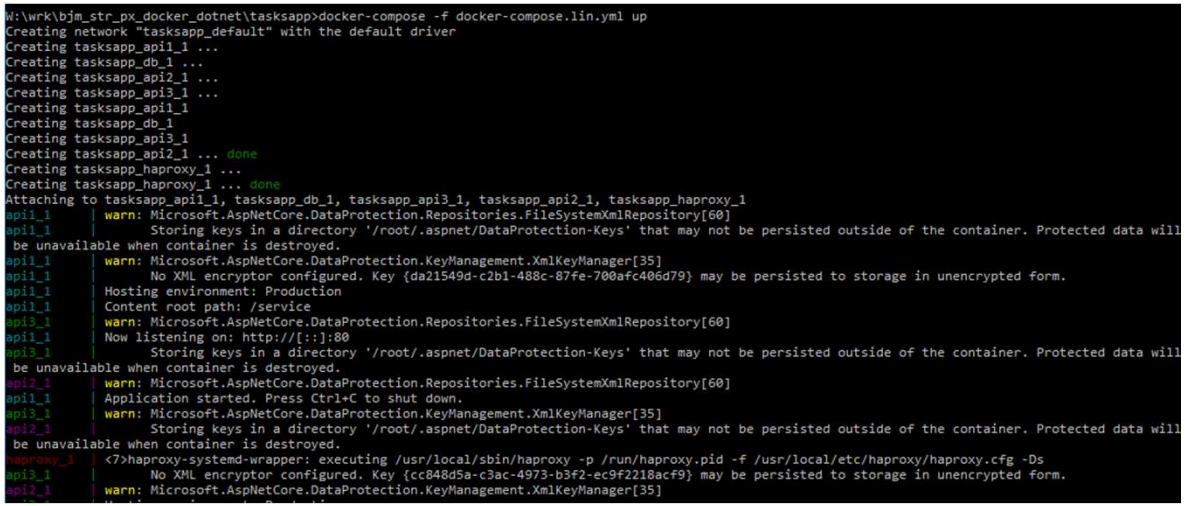
- Declare the external volume

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

Full System in One File

- Launch the system

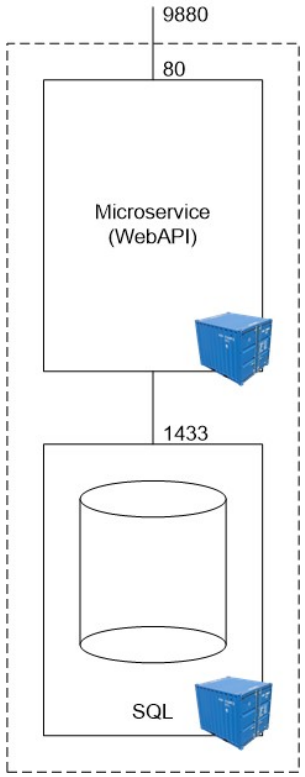
```
docker-compose -f docker-compose.lin.yml up
```



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


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


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


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/

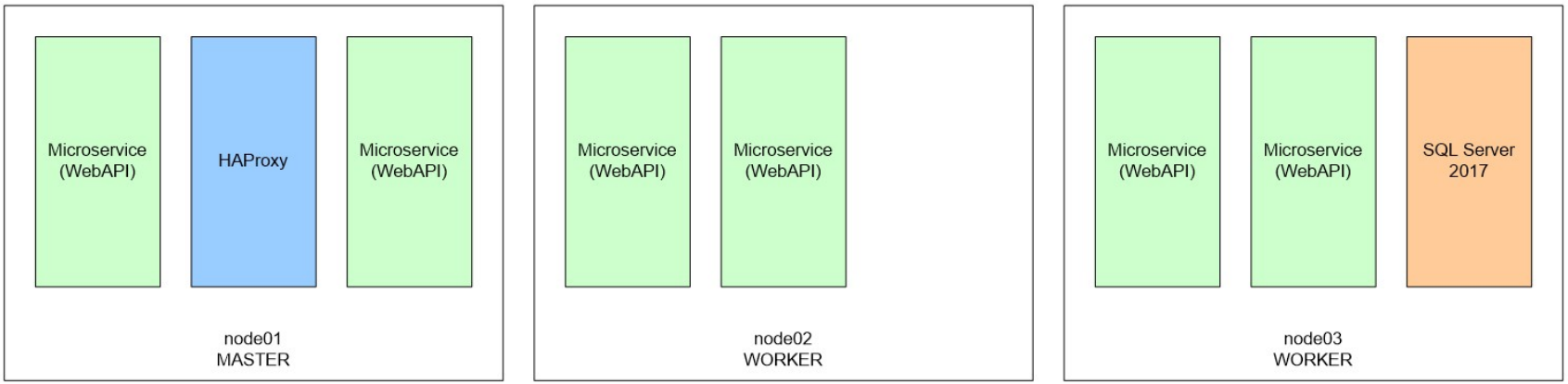
Plan

- 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

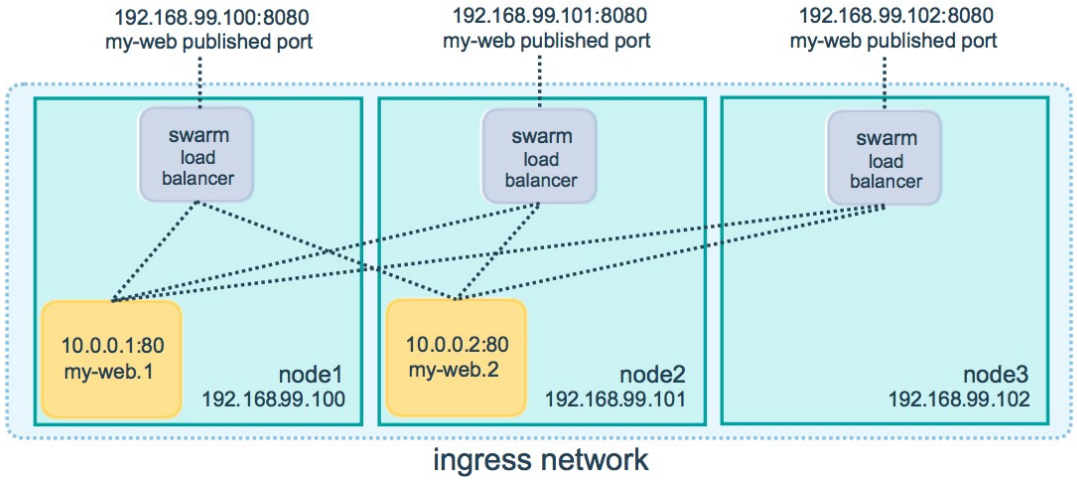
- Build and push the in-mem version of api for Linux
- Launch the api service with a single replica

```
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

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

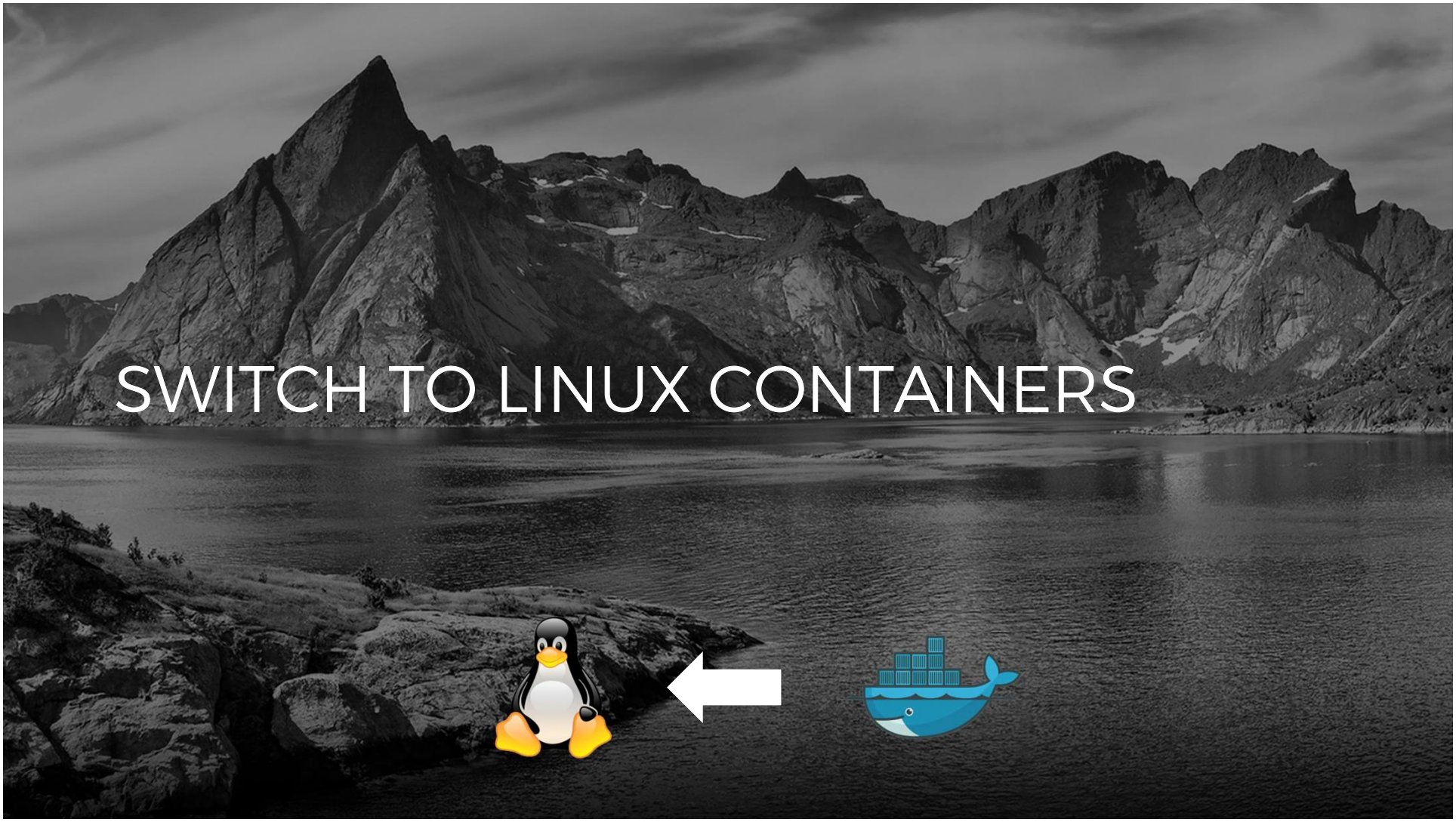
```
api:
        image: brendonmatheson/api:lin
        ports:
            - "9870:80"
        networks:
            - tasksbackend
        deploy:
            replicas: 5
```

- 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

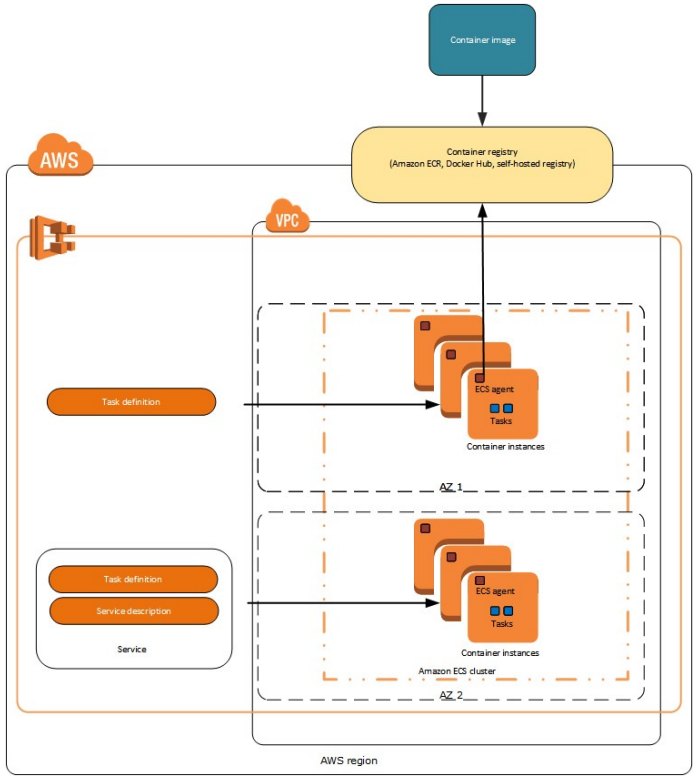
Swarm Further Reading

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


DEPLOYING TO AWS ECS

Deploying to AWS ECS

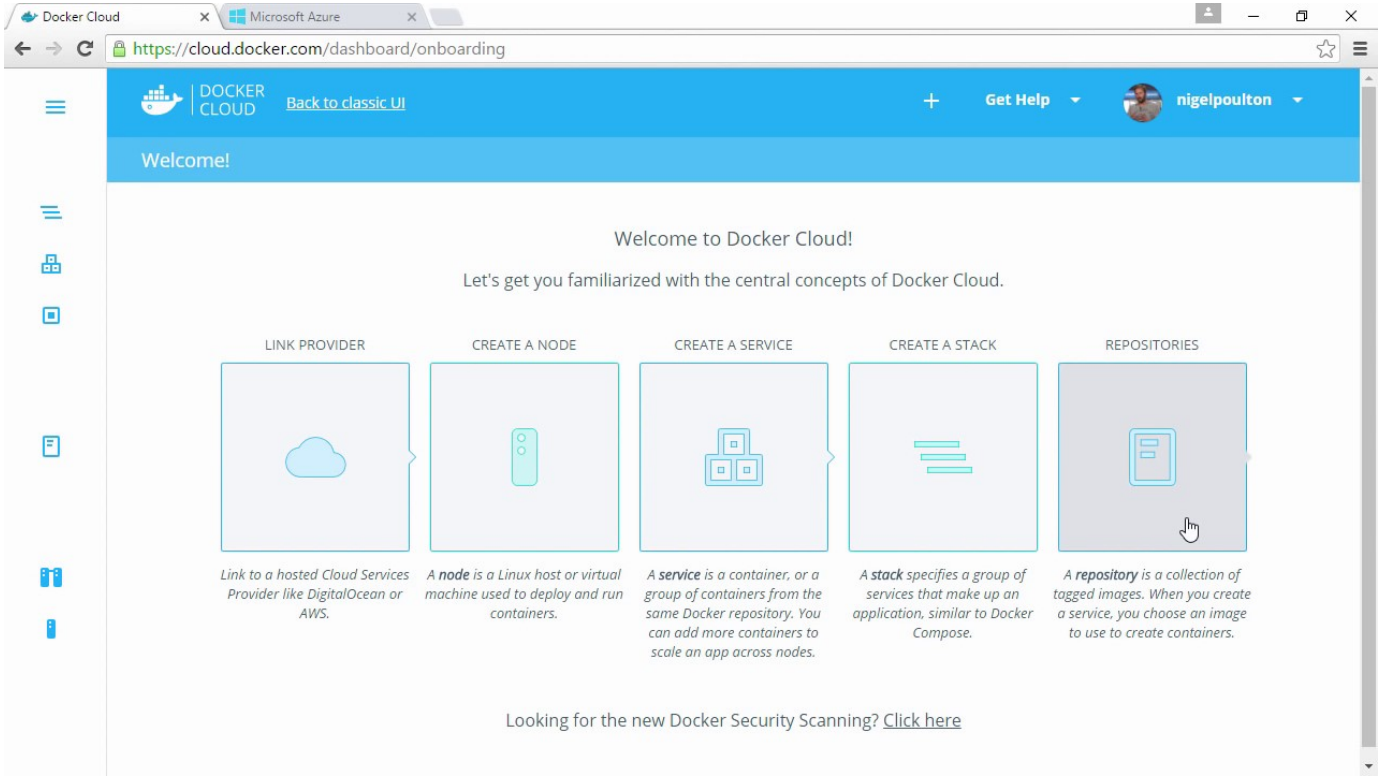
- **Registry** - ECR
  - Host our images on AWS
  - Alternative to Docker Hub
- **Cluster**
  - Set of virtual machines for running container workloads
  - Launch from wizard – runs CloudFormation template
- **Task**
  - Image plus execution meta-data
- **Service**
  - Clustered set of instances of a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


Container image
Container registry
(Amazon ECR, Docker Hub, self-hosted registry)
AWS
VPC
Task definition
ECS agent
Tasks
Container instances
AZ 1
Task definition
Service description
Service
ECS agent
Tasks
Container instances
Amazon ECS cluster
AZ 2
AWS region

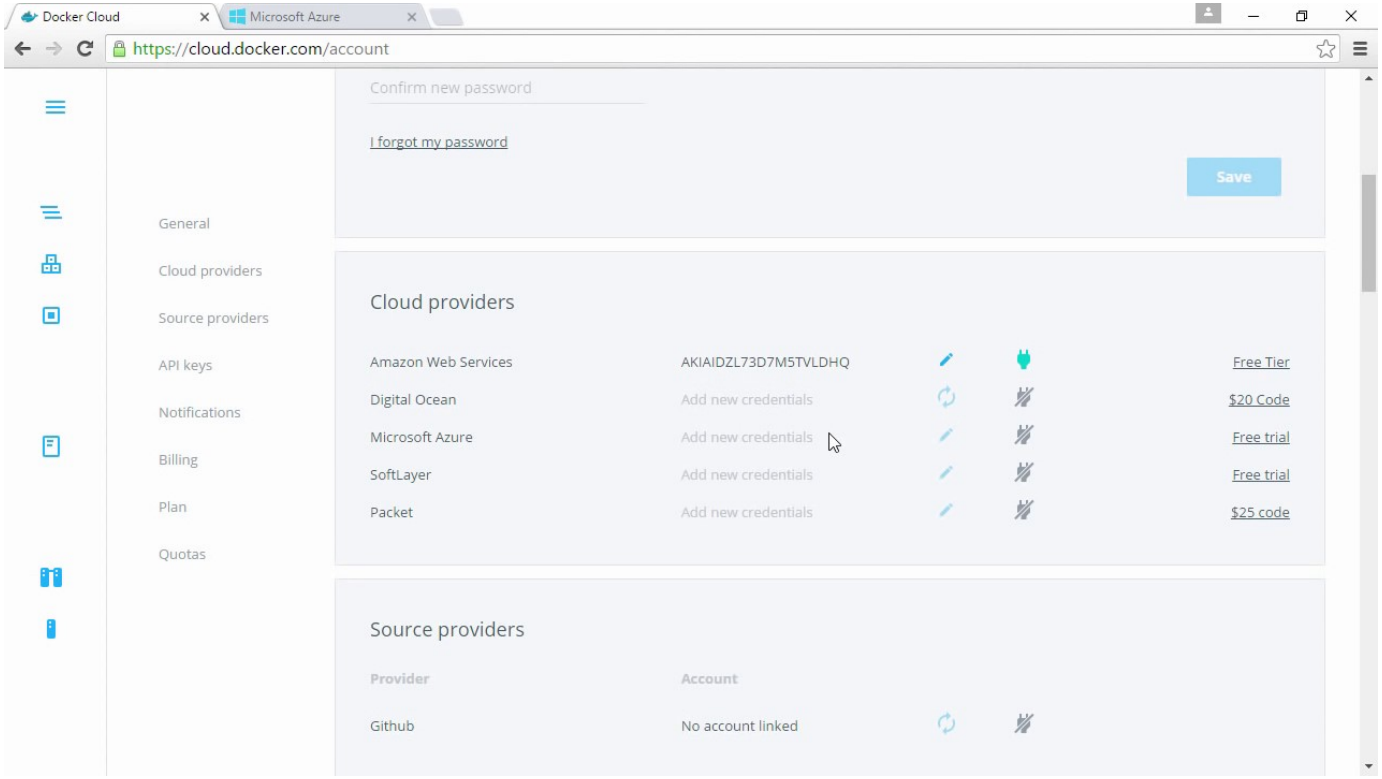

TOOLING ETC


DOCKER CLOUD

Docker Cloud



Docker Cloud




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

- 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/
    - 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/
    - 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