IT/Dev CONNECTIONS

ASP.NET Core & Docker

From 0 to Azure in 75 minutes

Marco De Sanctis
Visual Studio and Development Technologies MVP
info@marcodesanctis.it | *@crad77*

Delivered by KNect365 TMT an Informa business

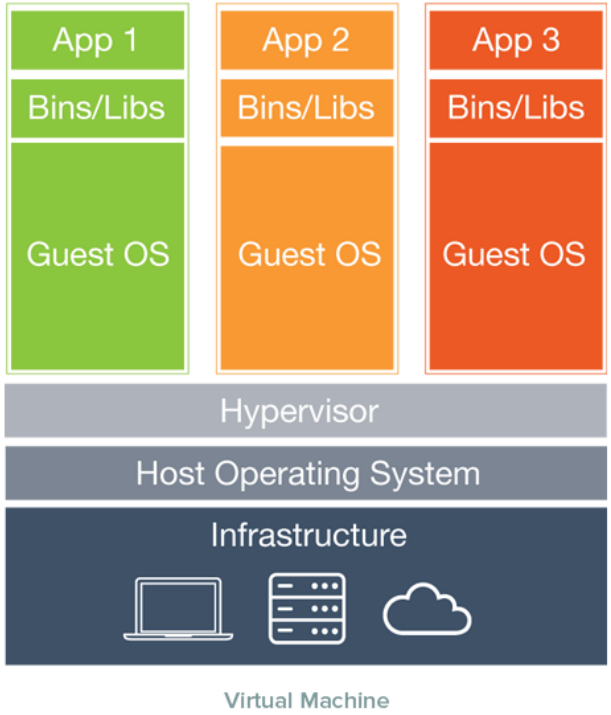
- What’s Docker and why should I care
- ASP.NET Core & Docker
- Let’s port a non-trivial project to Docker and AKS

DISCLAIMER

We are going to use some command line, but c'mon, it's not that scary, is it? ☺

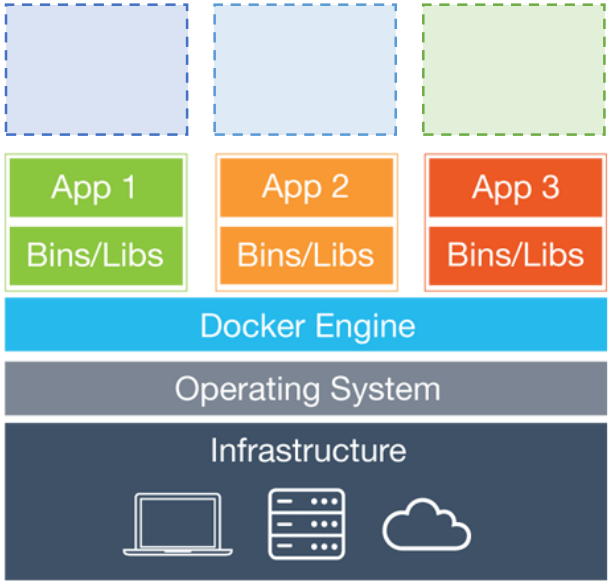
Anyway, don't worry: it's just a few commands, and I'll provide you a reference of everything we are going to use

HOW DID WE GET HERE?



Virtual Machine

Service density++ ☺

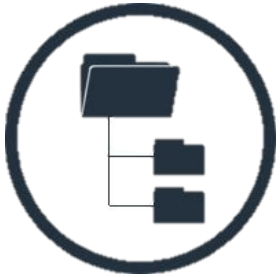


Container

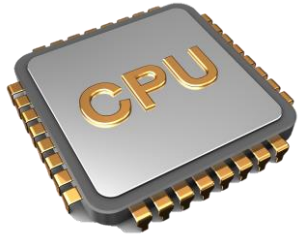
CONTAINERS == VIRTUALIZED OPERATING SYSTEM

IT/Dev CONNECTIONS

Kernel namespaces (Linux e Windows)



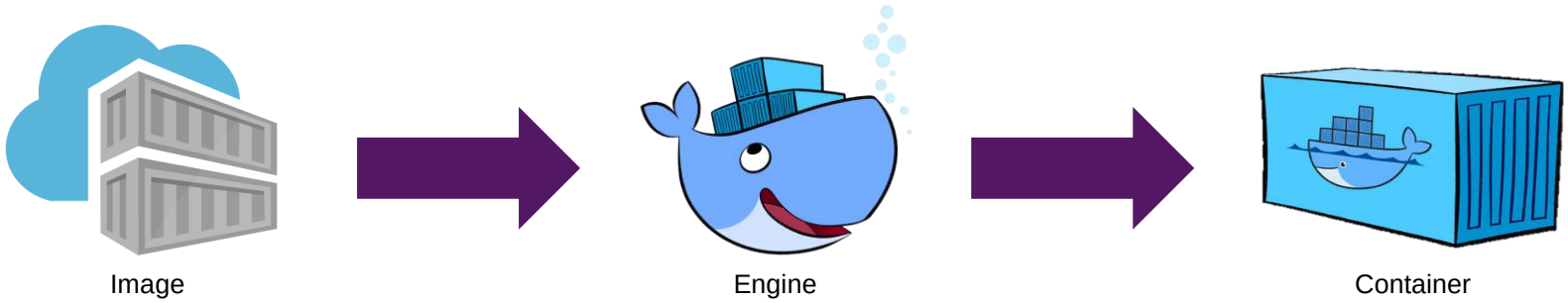
Virtual File System



Virtual Process Tree

Virtual Network

THE DOCKER WORKFLOW

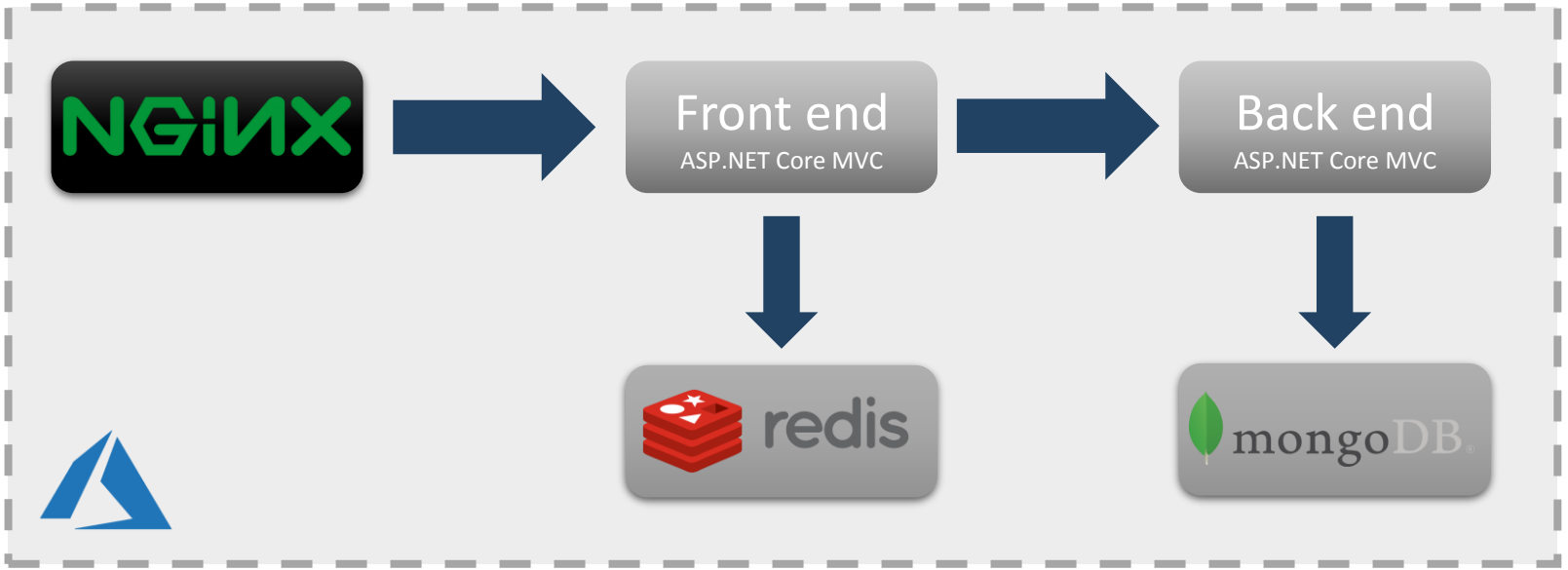


- Docker (aka Moby project) is free and open source, no limitations
- There's an enterprise edition (hosting, support, certification...)
- It's the de-facto standard for container-based applications

Let’s get started ☺

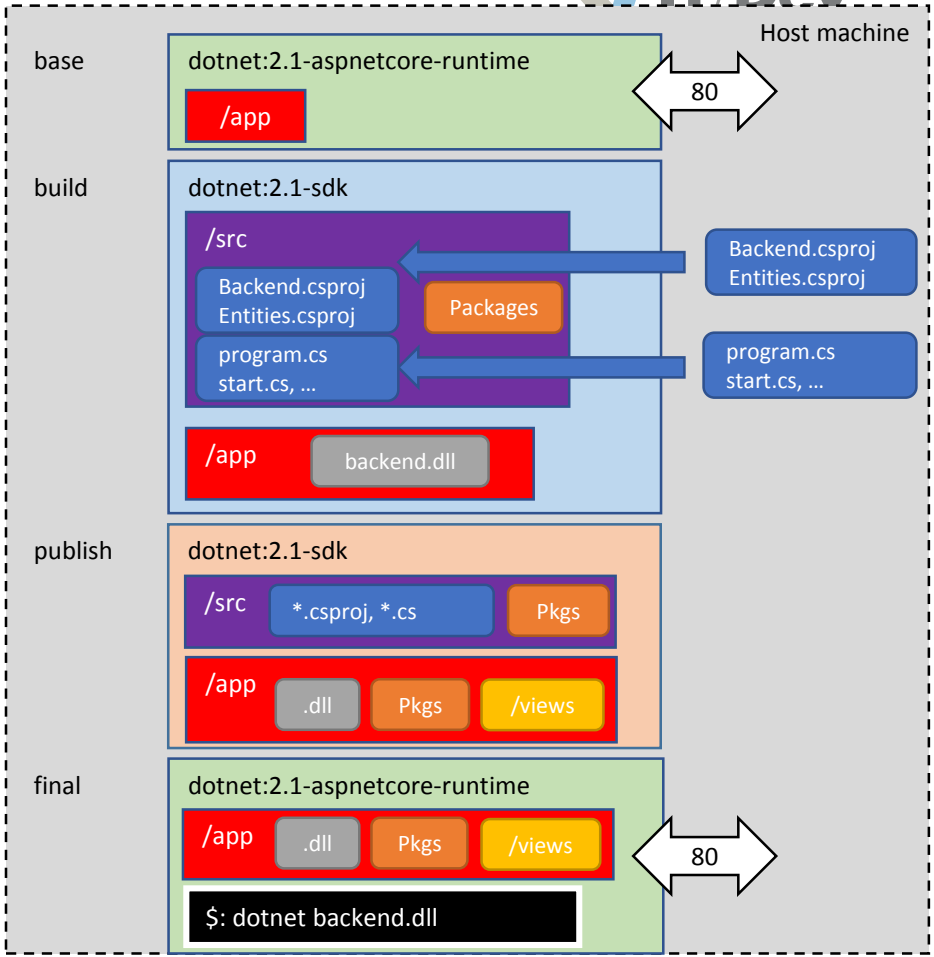
Demo

WHAT WE ARE GOING TO BUILD



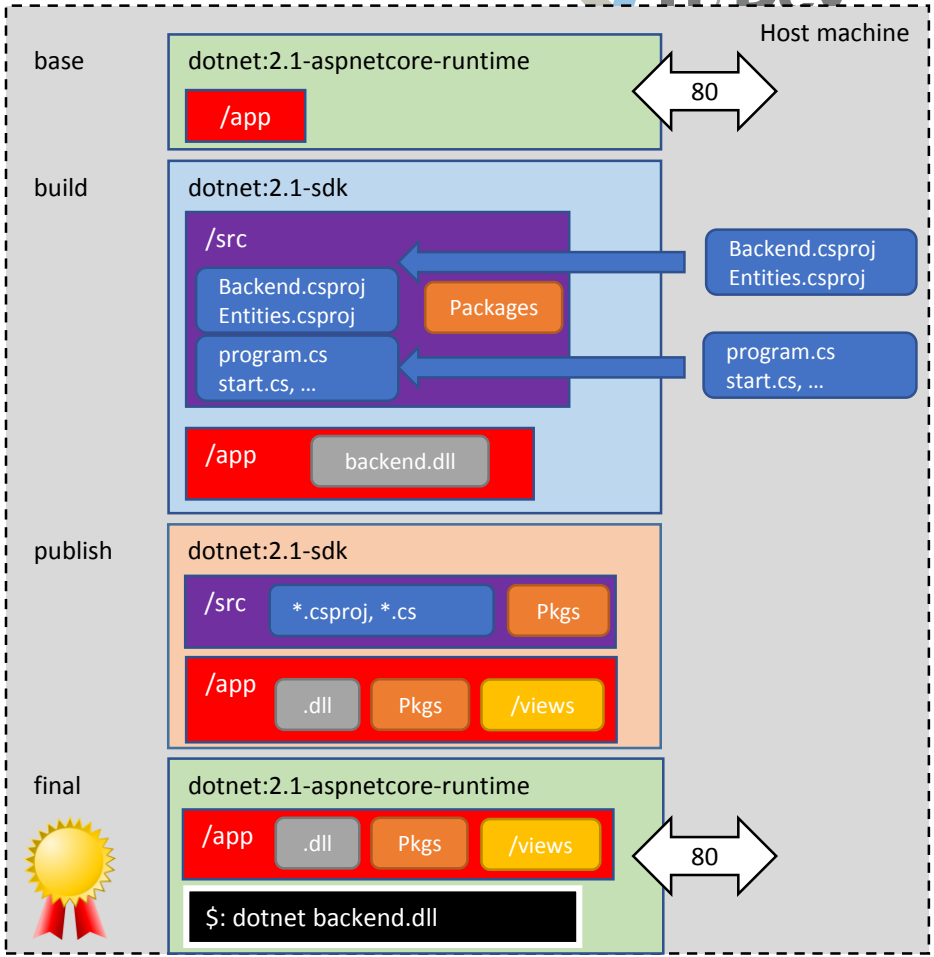
DOCKERFILE EXPLAINED

```
FROM ms/dotnet:2.1-runtime AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/dotnet:2.1-sdk AS build
WORKDIR /src
COPY Backend/Backend.csproj Backend/
COPY Entities/Entities.csproj Entities/
RUN dotnet restore Backend.csproj

COPY . .
WORKDIR /src/backend
RUN dotnet build -c Release -o /app

FROM build AS publish
RUN dotnet publish -c Release -o /app


FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "backend.dll"]
```

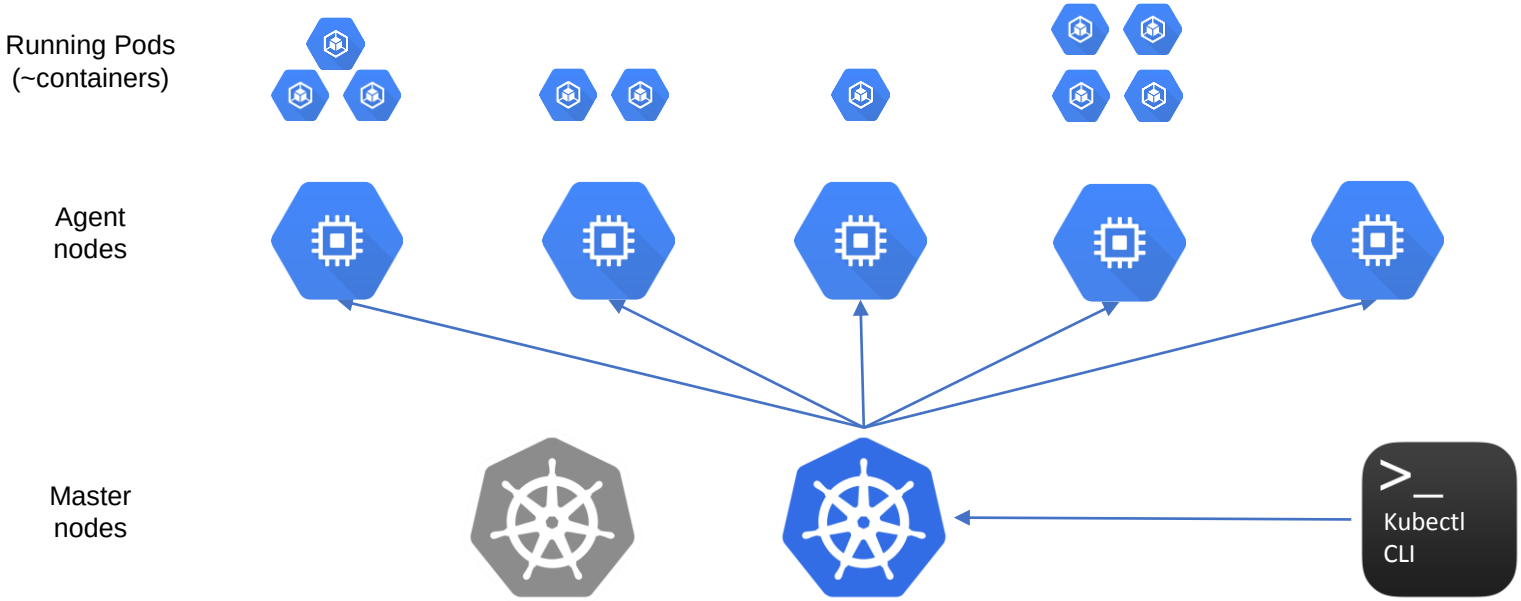


DOCKERFILE EXPLAINED

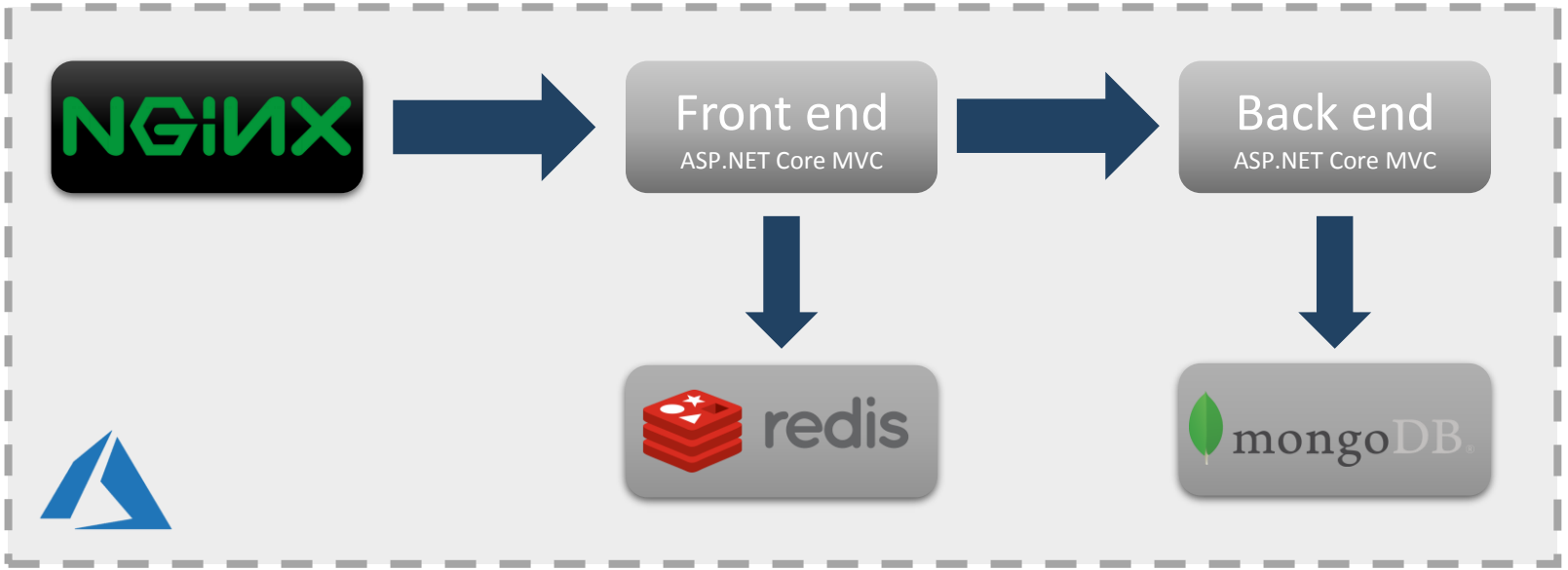
```
FROM ms/dotnet:2.1-runtime AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/dotnet:2.1-sdk AS build
WORKDIR /src
COPY Backend/Backend.csproj Backend/
COPY Entities/Entities.csproj Entities/
RUN dotnet restore Backend.csproj

COPY . .
WORKDIR /src/backend
RUN dotnet build -c Release -o /app

FROM build AS publish
RUN dotnet publish -c Release -o /app


FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "backend.dll"]
```



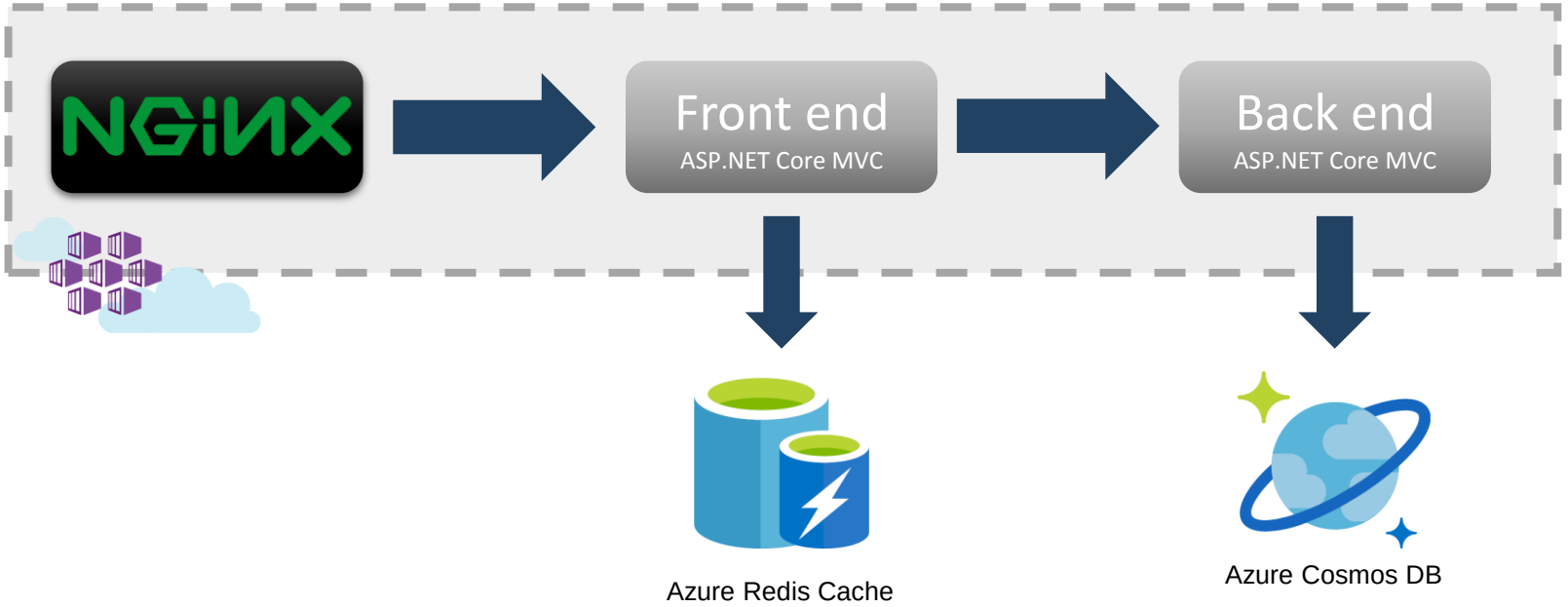
We need an orchestrator: enter Kubernetes and AKS



OUR APPLICATION SO FAR



LET'S LEVERAGE AZURE PAAS



A GLIMPSE INTO KUBERNETES'S OBJECT MODEL

IT/Dev CONNECTIONS

Deployment

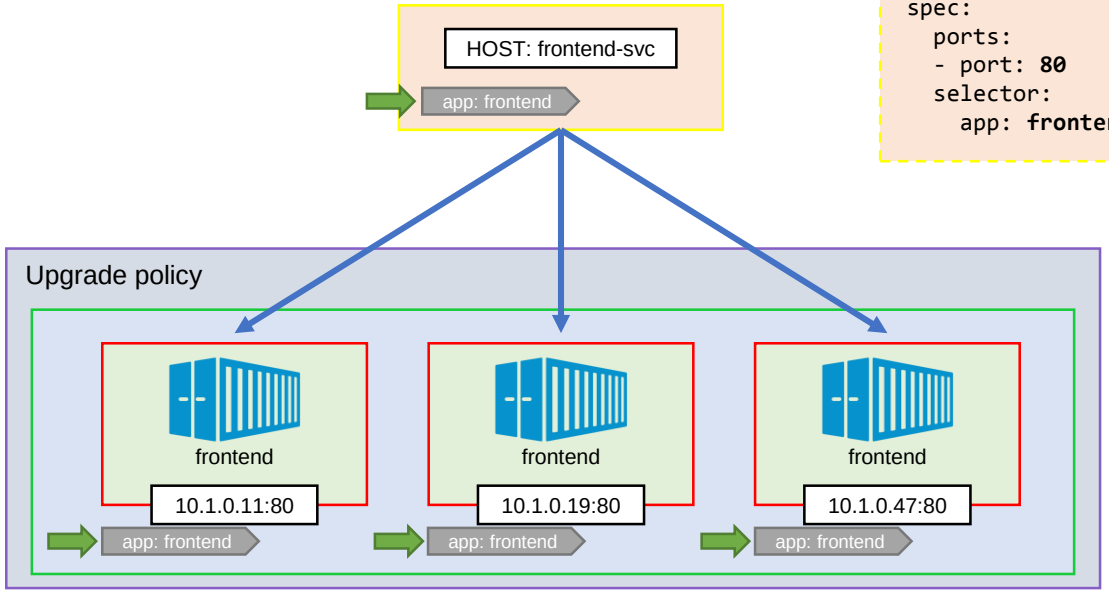
```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: frontend-dpy
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: myreg/frontend:latest
        ports:
        - containerPort: 80
```

ReplicaSet

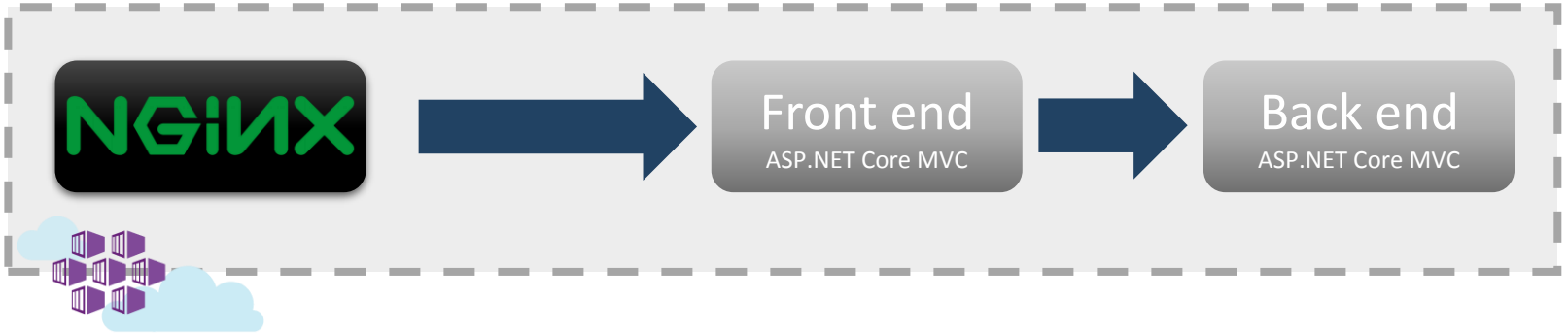
Pod

Service

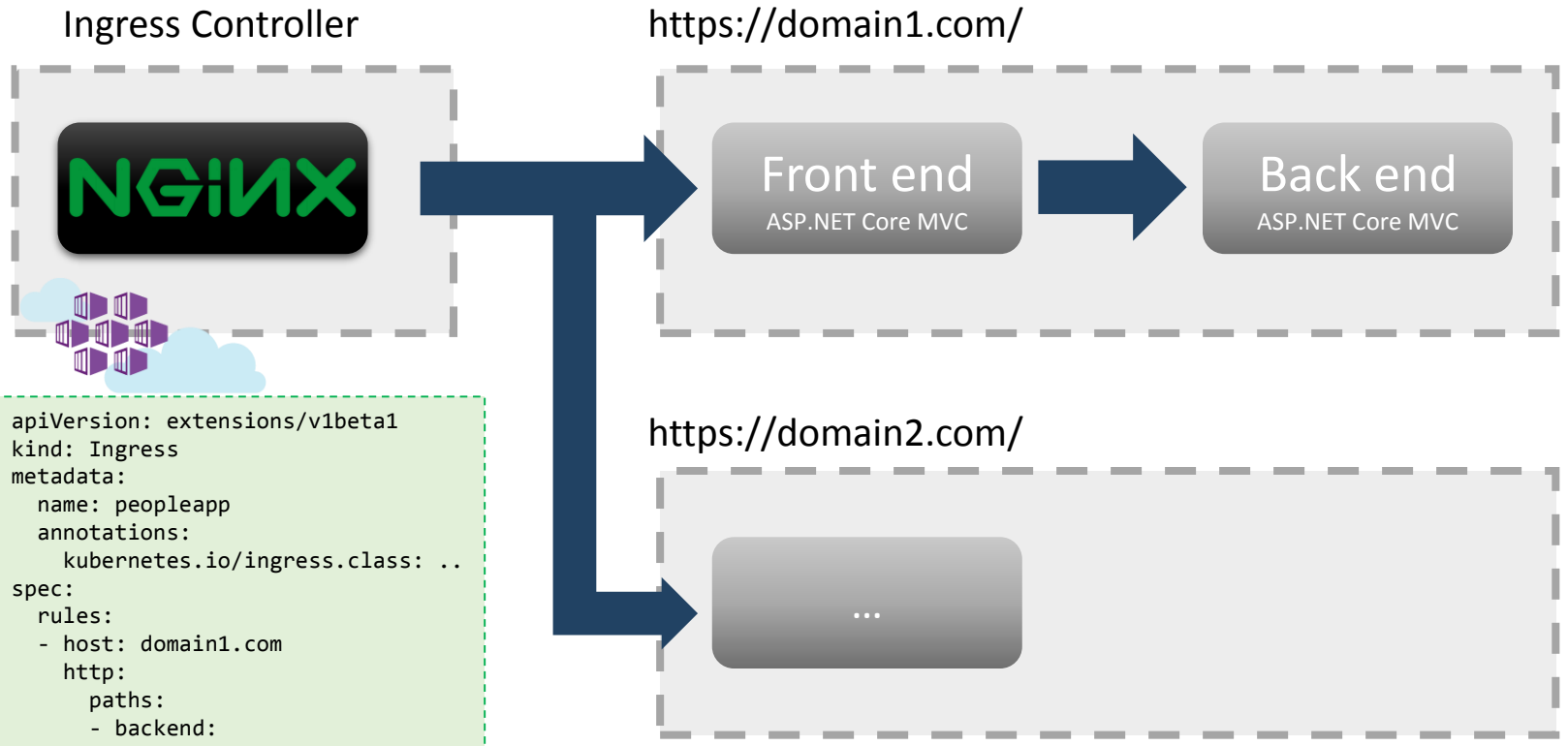
```
apiVersion: v1
kind: Service
metadata:
  name: frontend-svc
spec:
  ports:
  - port: 80
  selector:
    app: frontend
```



INGRESS CONFIGURATION



INGRESS CONFIGURATION




```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: peopleapp
  annotations:
    kubernetes.io/ingress.class: ..
spec:
  rules:
  - host: domain1.com
    http:
      paths:
      - backend:
          serviceName: frontend
          servicePort: 80
        path: /
```

RECAP

- Run **Mongo** & **Redis** as Docker containers
- Dockerised **frontend** and **backend**
- Added **nginx**
- Described the whole system on **docker-compose**
- Run all of it on our laptop
- Saved the images on **Azure Container Registry**
- Configured a CI/CD pipeline for them
- Deployed on a Kubernetes cluster in **Azure Container Service (AKS)**
- Set up a dashboard and an alert in **Log Analytics**

RECAP - COMMANDS

`docker run –p 8080:80 wordpress`	Starts a container from an image, exponing it on 8080
`docker ps`	Lists of all running containers (-a includes stopped ones)
`docker start containerName`	Starts (and stops) a container
`docker rm -f $(docker ps -q)`	Removes all running containers
`docker images`	Lists all images
`docker rmi imageName`	Removes an image
`docker login myregistry.azurecr.io`	Logs in Azure Container Registry
`docker tag frontend myregistry.azurecr.io/frontend`	Tags an image for Azure Container Registry
`docker push myregistry.azurecr.io/frontend`	Pushes an image to Azure Container Registry
`kubectl apply -f filename`	Creates the kubectl objects specified in the file
`kubectl get deployments/services/pods`	Lists all the deployments/services/pods in the cluster
`kubectl proxy --address="0.0.0.0"`	Tunnels the cluster's dashboard to localhost

RECAP – ON AZURE

Azure Container Registry

- Our private repository where we can store Docker images
- `docker login myregistry.azurecr.io`

Azure Web App for Containers

- They can run only one container at a time
- Sort of CD supported via Azure Container Registry and web hooks

Azure Container Service (AKS)

- Fully managed orchestrator
- Based on Kubernetes
- https://docs.microsoft.com/en-us/azure/aks/

Container monitoring solution

- Based on Azure LogAnalytics
- Uses a DaemonSet to track cluster events
- https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-containers

THANK YOU!

IT/Dev CONNECTIONS

@crad77
info@marcodesanctis.it

Get the code at
https://github.com/cradle77/DockerGettingStarted