#GlobalAzure

#CloudGenVerona

@cloudgen_verona

Thanks to all the sponsors

CloudGen
ominidi verso l'evoluzione

PREMIUM SPONSOR

4WARD

KIRATECH®
WE DEVOPS IT

BASIC SPONSOR

CODICEPLASTICO

dgroove
Tecnologia emozionale.

CloudGen
ominidi verso l'evoluzione

TOPIC

ASP.NET Core, Kubernetes and Azure DevOps

Integrate containers and Kubernetes into your Azure DevOps build & release model and live happily ever after

Who’s this guy?

Marco De Sanctis

Visual Studio and Development Technologies MVP
info@marcodesanctis.it

@crad77

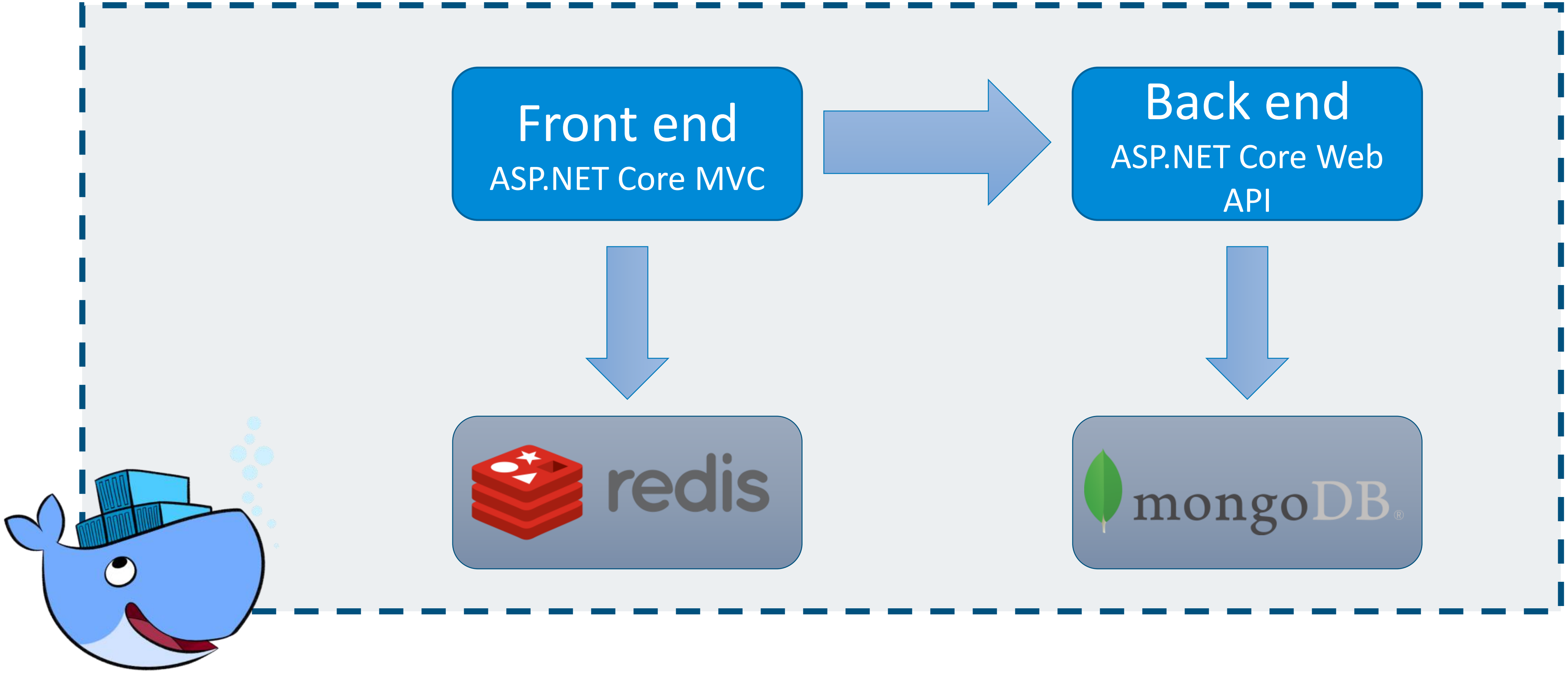
https://github.com/cradle77

https://uk.linkedin.com/in/desanctismarco

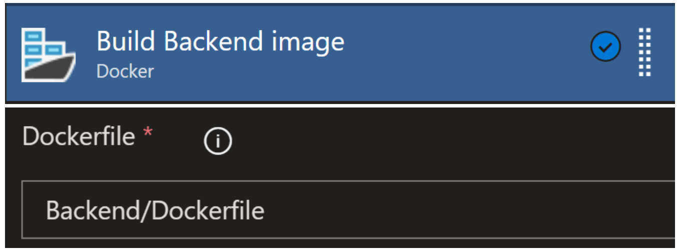
Agenda

- The simplest build pipeline possible
- Release to Azure Kubernetes Service
- Add testing to the pipelines
- Going to production

Our application for today



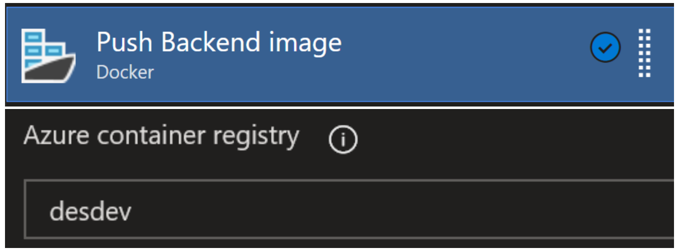
The simplest build pipeline possible



Build Backend image
Docker

Dockerfile *

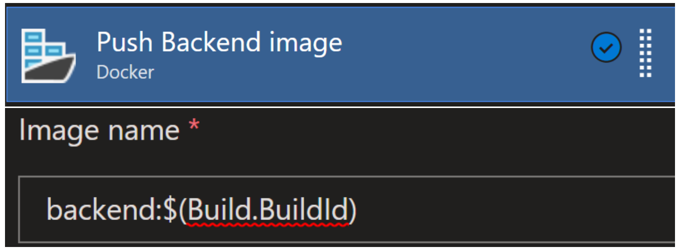
Backend/Dockerfile



Push Backend image
Docker

Azure container registry

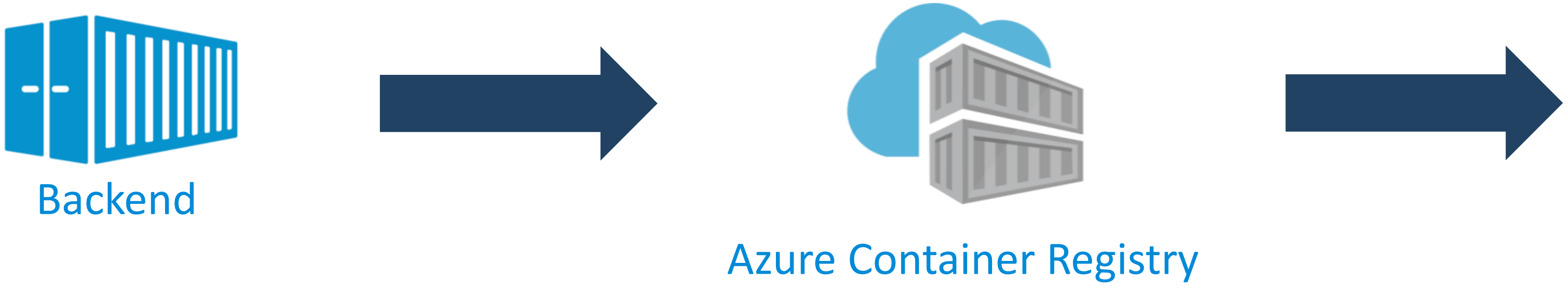
desdev

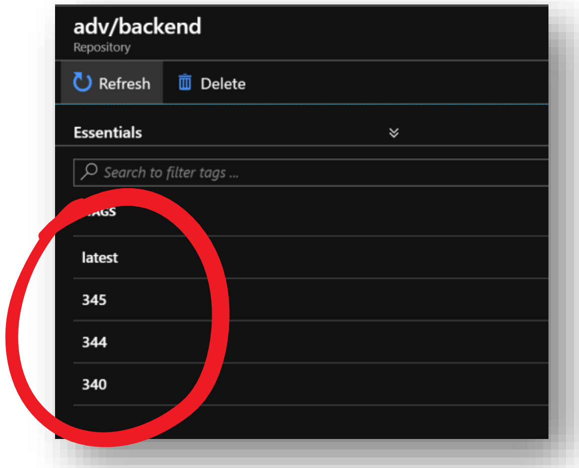


Push Backend image
Docker

Image name *

backend:$(Build.BuildId)





adv/backend
Repository

AGS

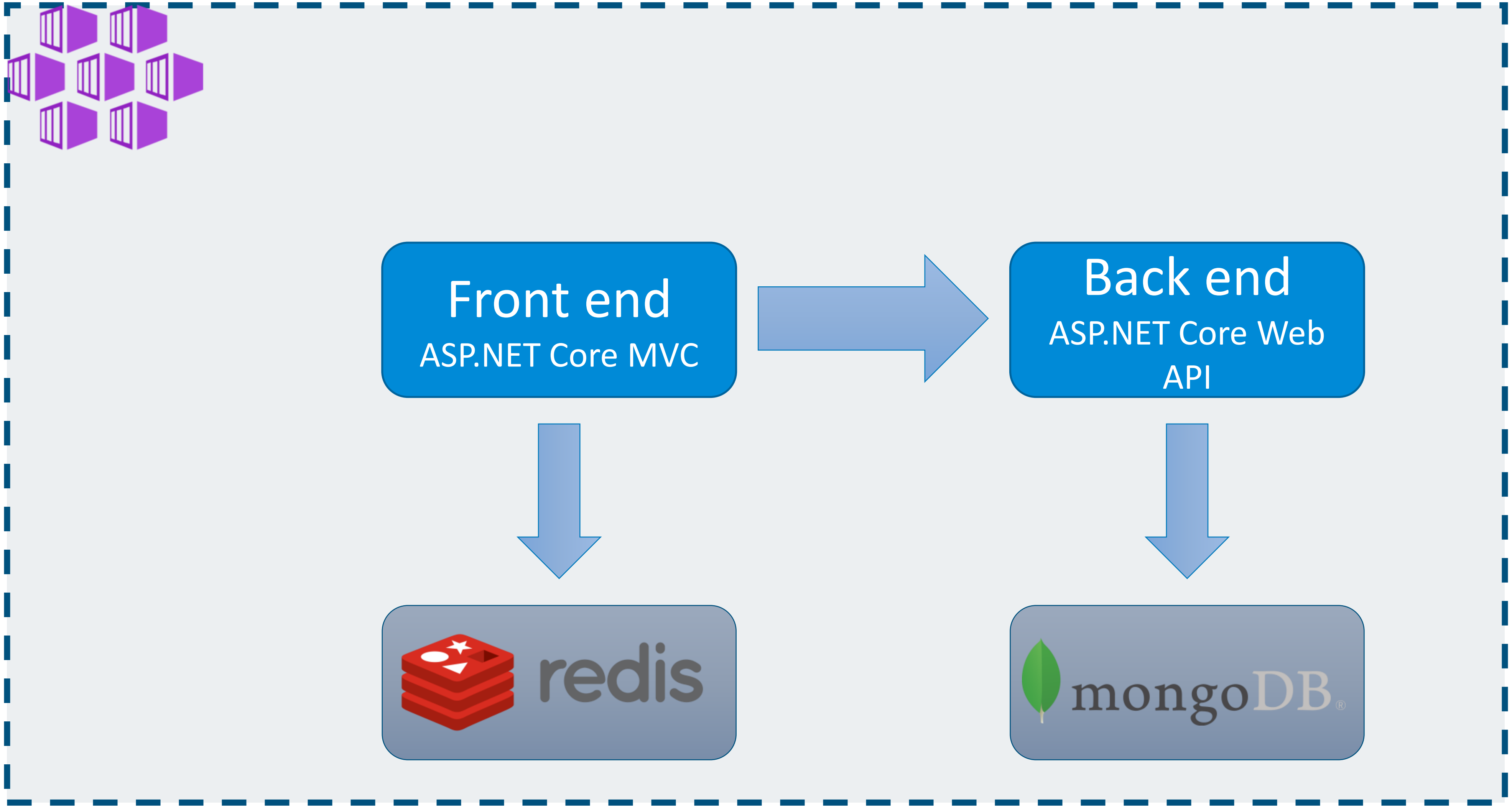
latest

345

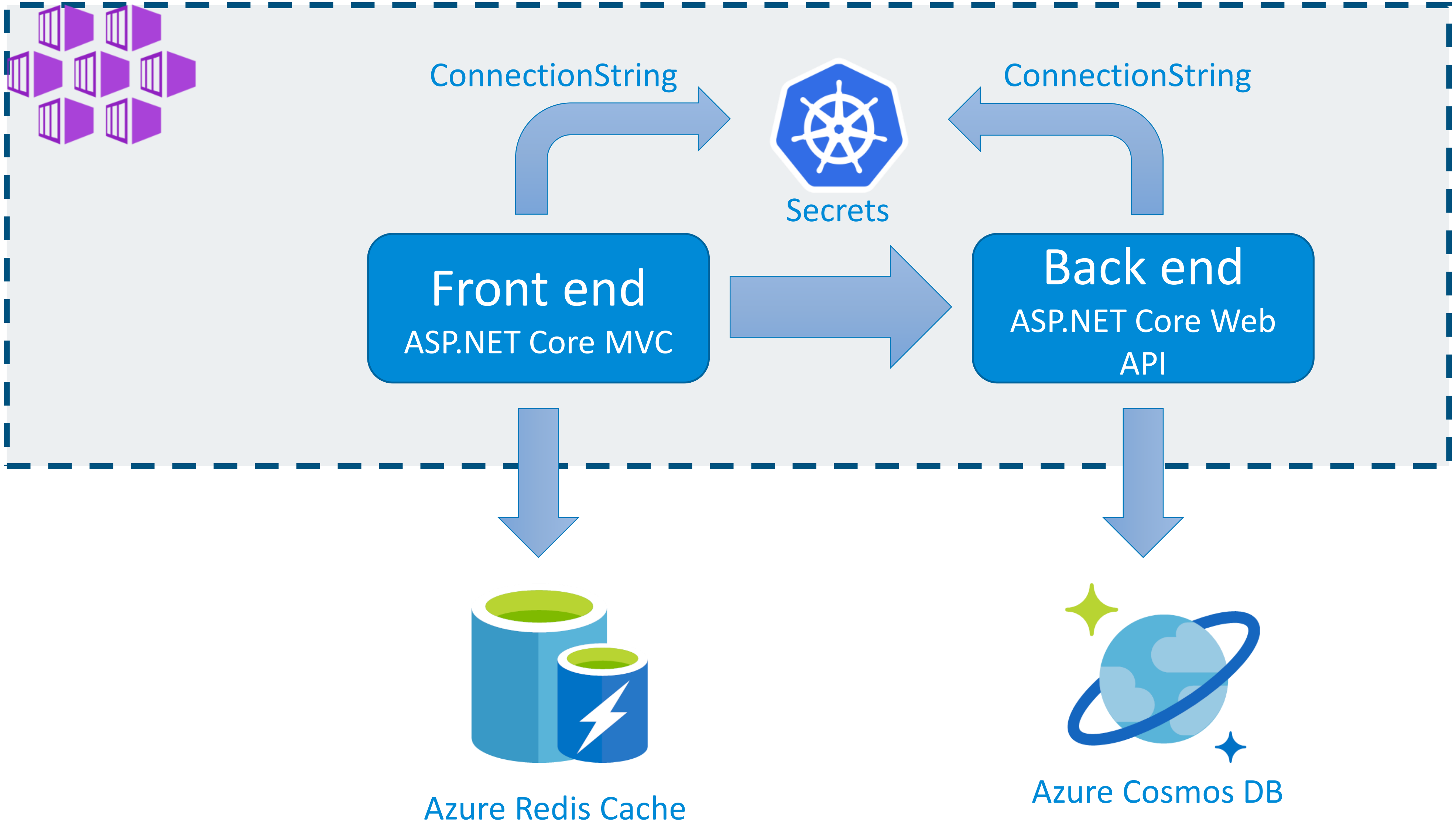
344

340

Ready to deploy to Azure Kubernetes Service...



…but let’s also leverage Azure PaaS offering



ConnectionString

ConnectionString

Secrets

Front end

ASP.NET Core MVC

Back end

ASP.NET Core Web API

Azure Redis Cache

Azure Cosmos DB

What we want from our Release pipeline

Select the images we want to deploy

Automatically provision Azure PaaS Resources

Deploy the entire system and wire it up in AKS

Spin up different environments in our AKS cluster

How do we execute unit tests?

We want to run tests as part of the image build process

No image should be built if unit tests don’t pass

We want to run them with Docker

We want to expose the results

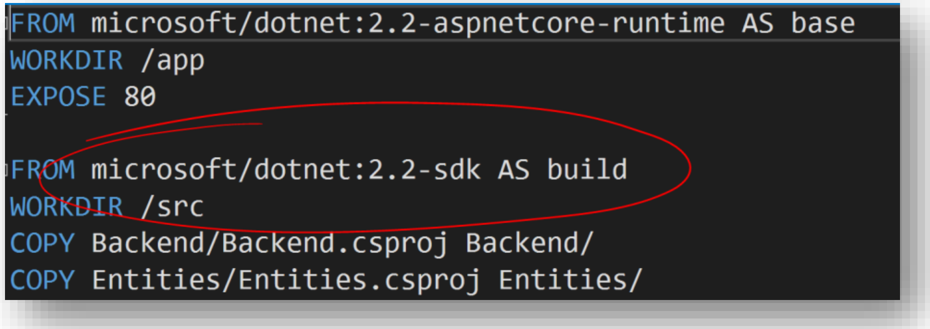
Testing in 3 steps

```
dotnet test    # executes a unit test project
  -l trx       # export log format
  -r /results # results folder
```

Requires the full SDK!

```
docker build . -f .\Backend\Dockerfile `
  --tag backend-ut `
  --target build
```

backend-ut



```
FROM microsoft/dotnet:2.2-aspnetcore-runtime AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/dotnet:2.2-sdk AS build
WORKDIR /src
COPY Backend/Backend.csproj Backend/
COPY Entities/Entities.csproj Entities/
```

```
docker run `
  --rm `                                # remove after execution
  -w /src/Backend.Tests `               # set working directory
  -v C:\...\testresults:/results # share /results with host machine
  backend-ut
  dotnet test –l trx –r /results # command to execute
```

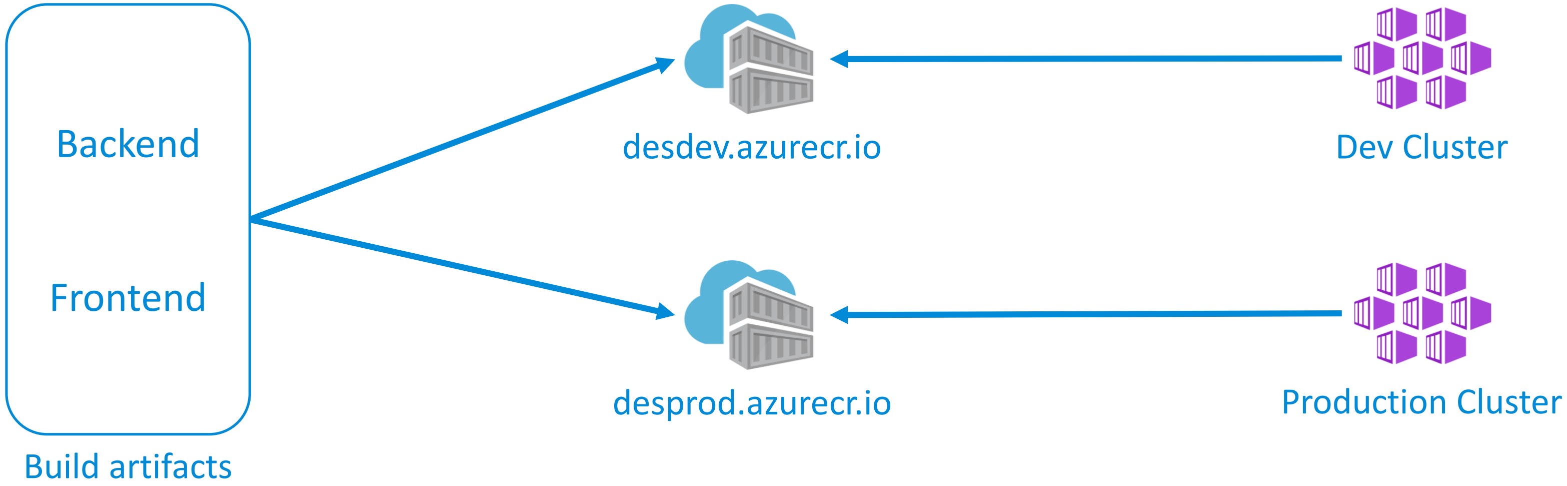
Going to production!

What else do we need?

Separate registries for dev & prod
Our custom domain under HTTPS
Ability to perform controlled roll updates

Keep a separate production registry

We want separate Azure Container Registries for Dev and Prod

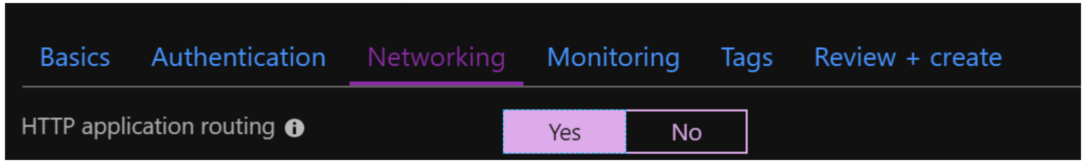


Isolate images signed off for prod

Production cluster cannot access Dev Registry

Bring your custom domain under HTTPS

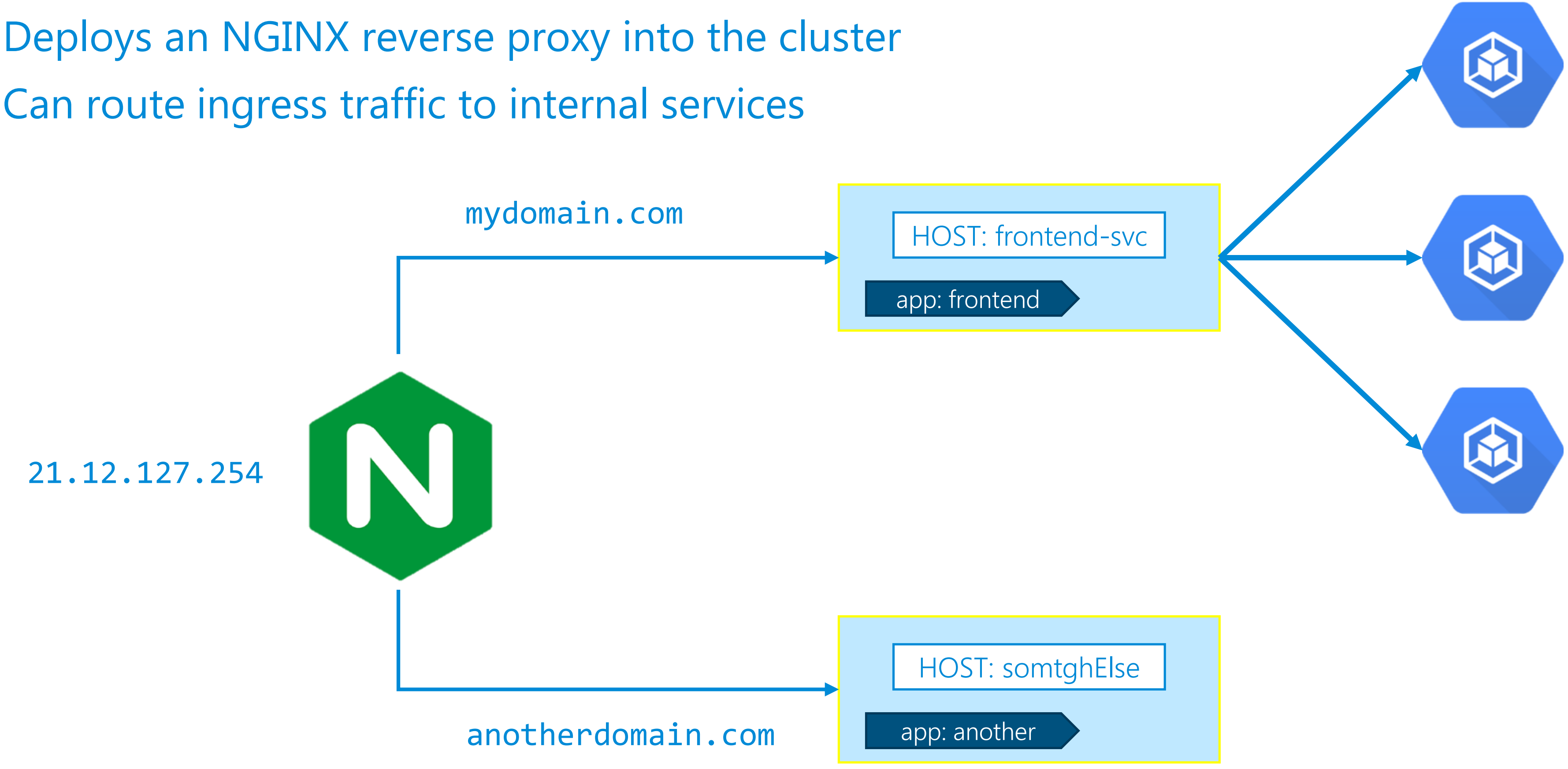
- Installed and configured by HTTP-Application-Routing



Basics | Authentication | Networking | Monitoring | Tags | Review + create

HTTP application routing [✓ Yes] No

- Deploys an NGINX reverse proxy into the cluster
- Can route ingress traffic to internal services



Configure TLS in 3 steps

1) Install **CertManager** on the Cluster to add the capability of requesting certificates

```
helm install stable/cert-manager \
    --namespace kube-system \
    --set ingressShim.defaultIssuerName=letsencrypt-prod \
    --set ingressShim.defaultIssuerKind=ClusterIssuer
```

2) Create a **ClusterIssuer** object

```
apiVersion: certmanager.k8s.io/v1alpha1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  ...
```

3) Create a **Certificate** object

```
apiVersion: certmanager.k8s.io/v1alpha1
kind: Certificate
metadata:
  name: tls-people-secret
spec:
  issuerRef:
    name: letsencrypt-prod
...
```

https://docs.microsoft.com/en-us/azure/aks/ingress-tls

Recap – What we've achieved in 60 minutes

Run a **containerised ASP.NET Core** solution locally

Created a simple **build pipeline**

Added **unit tests**

Used **docker-compose** for **integration** and **UI tests**

Created a multi-environment **release pipeline**

Managed to keep **separated Docker registries** for Dev and Prod

Released to production with **custom domain under HTTPS**

Thank you! ☺

Questions?

info@marcodesanctis.it

Get the code at

https://github.com/cradle77/AksDevOps

https://github.com/cradle77

@crad77

https://uk.linkedin.com/in/desanctismarco