Kubernetes
the Deltatre way

3 JUN, 5:30 PM

ogr officine grandi riparazioni

newesis e Professional | Have Fun!

tag Talent Garden

Kubernetes basics

Deltatre Innovation Lab

ENRICO SABBADIN (DELTATRE)
SUPPORTED BY
DIEGO D'AGOSTINI (DELTATRE)

Agenda

- Basic Concepts
  - From docker to K8s: orchestrators
  - Cluster components
- Kubernetes Resources
  - Pods, Deployments, Services, Ingress, Namespaces, Configmaps, Secrets etc ..
- Interacting with the cluster
  - API Server, kubectl and the dashboard
- Deploy to cluster
  - Yaml (Kind)
  - Helm (AKS)
- Storage
- Security (RBAC)
- Next sessions

Basic Concepts

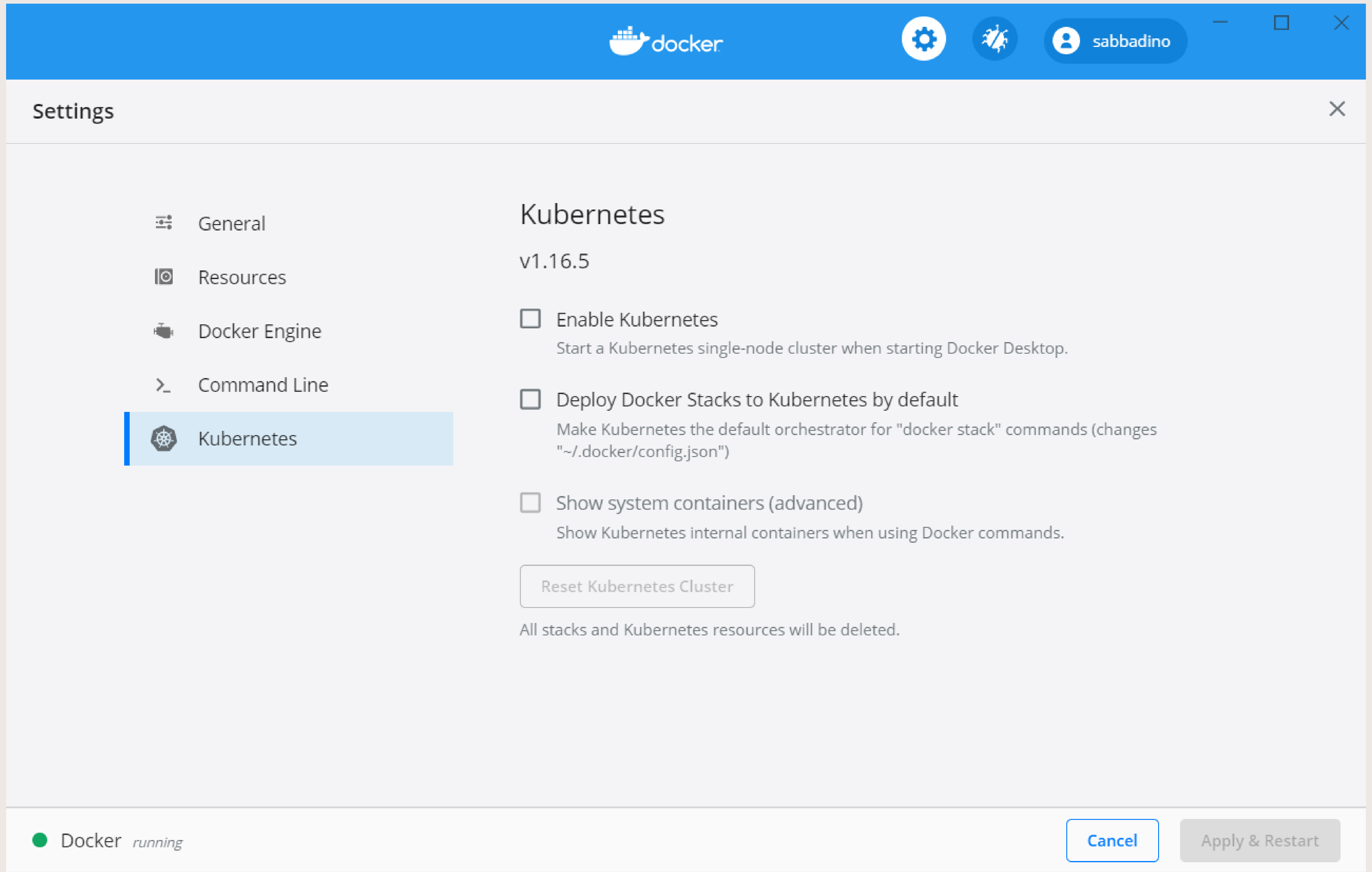
From Docker to k8s

- **K8s is an orchestrator for Docker containers**
  - Not the only one but the de-facto standard
  - Manages containers deployment and keeps them in the desired state
  - Manages networking among the containers
  - manages networking between a container and the «outside world»
  - Provides more then one can expect from an orchestrator, somehow you can see it as a solution for a private cloud
    - *No cloud vendor lock-in. One of the main reason why deltatre has chosen k8s as its current reference platform*
- **Provides an abstraction on the docker «stuff»**
  - Lets you concentrate on higher level concerns
  - *docker build and push* is the only 2 commands you need to know about docker (in basic scenarios)
    - Forget docker run, forget docker networking

deltatre

K8s install options

- Managed K8s on cloud providers (managed) AKS (Azure) , EKS (Aws), GKE (Google), Digital Ocean, etc
- Bare metal: set up an on-premise cluster (not for beginners)
- K8s for development, testing and experimenting
  - You can activate a k8s cluster in Docker desktop (Windows and mac)
  - Minikube
  - Kind: runs an image containing a k8scluster.
    You can simulate more than one node
  - Microk8s



Kubernetes

v1.16.5

☐ Enable Kubernetes
Start a Kubernetes single-node cluster when starting Docker Desktop.

☐ Deploy Docker Stacks to Kubernetes by default
Make Kubernetes the default orchestrator for "docker stack" commands (changes "~/.docker/config.json")

☐ Show system containers (advanced)
Show Kubernetes internal containers when using Docker commands.

All stacks and Kubernetes resources will be deleted.

What we use in this session: Kind and AKS clusters

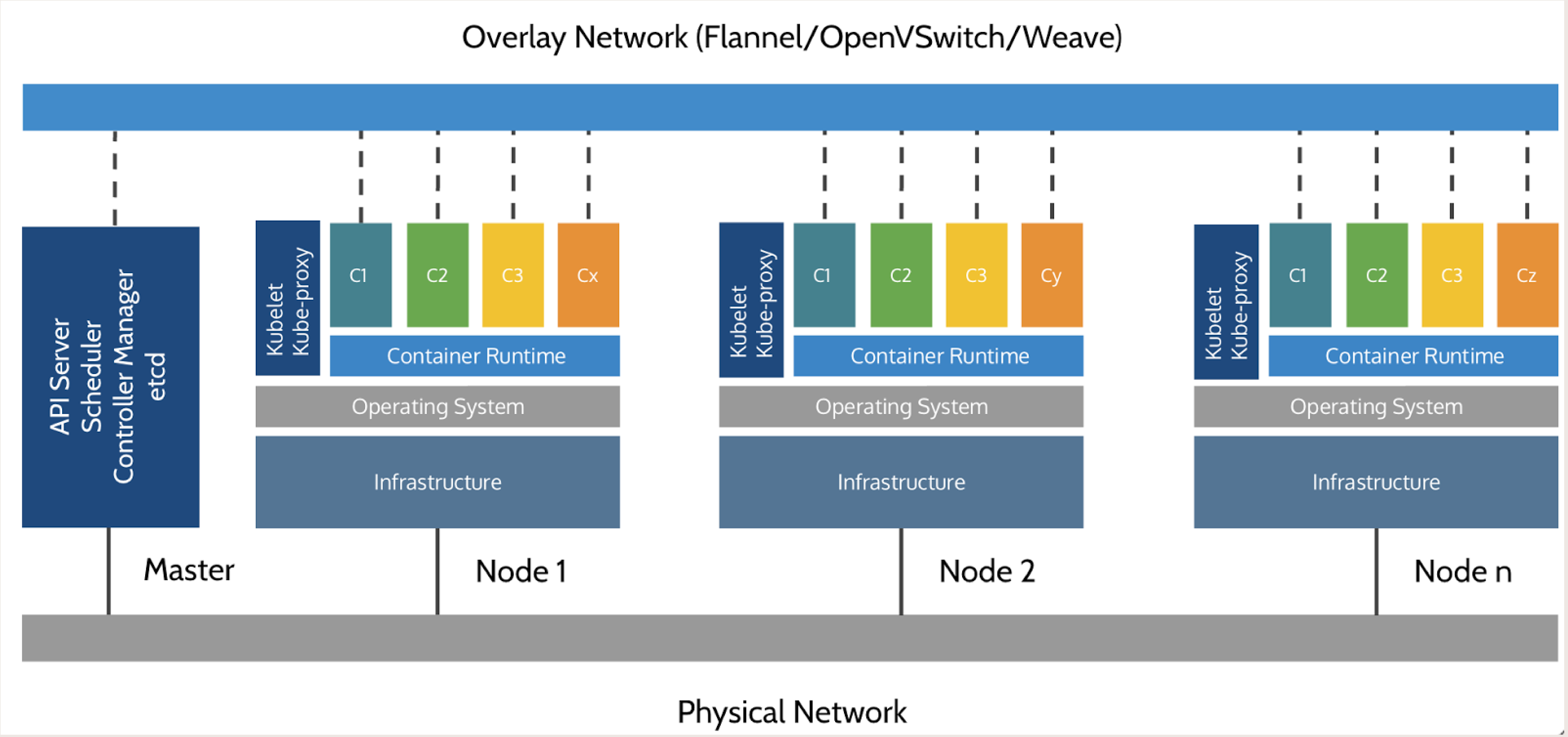
One time cluster setup done ahead:

1. Cluster creation
2. k8s dashboard installation (skipped if available as built-in feature, as in aks)
    1. Dashboard permissions adjusted
3. Ingress controller installation (ngnix in the demo)
4. CertManager for https (AKS only) installation
5. Dns name for the provided LoadBalancer Ip (AKS only) registration
6. Persistent storage set up

Basic Concepts

Cluster components

- In K8S a cluster is an abstraction
  - It lets you interact with virtual machines (nodes), containers, all the networking stuff, etc.., as if you were interacting with a single entity
- A k8s cluster is made by
  - Master components
    - *api server*
    - *etcd (store)*
    - *Scheduler*
    - *kube controller*
    - ***cloud controller***
  - Nodes (VM or real HW)
    - *container runtime*
    - *kubelet (node agent)*
    - *kubeproxy (guarantee cluster network communication)*

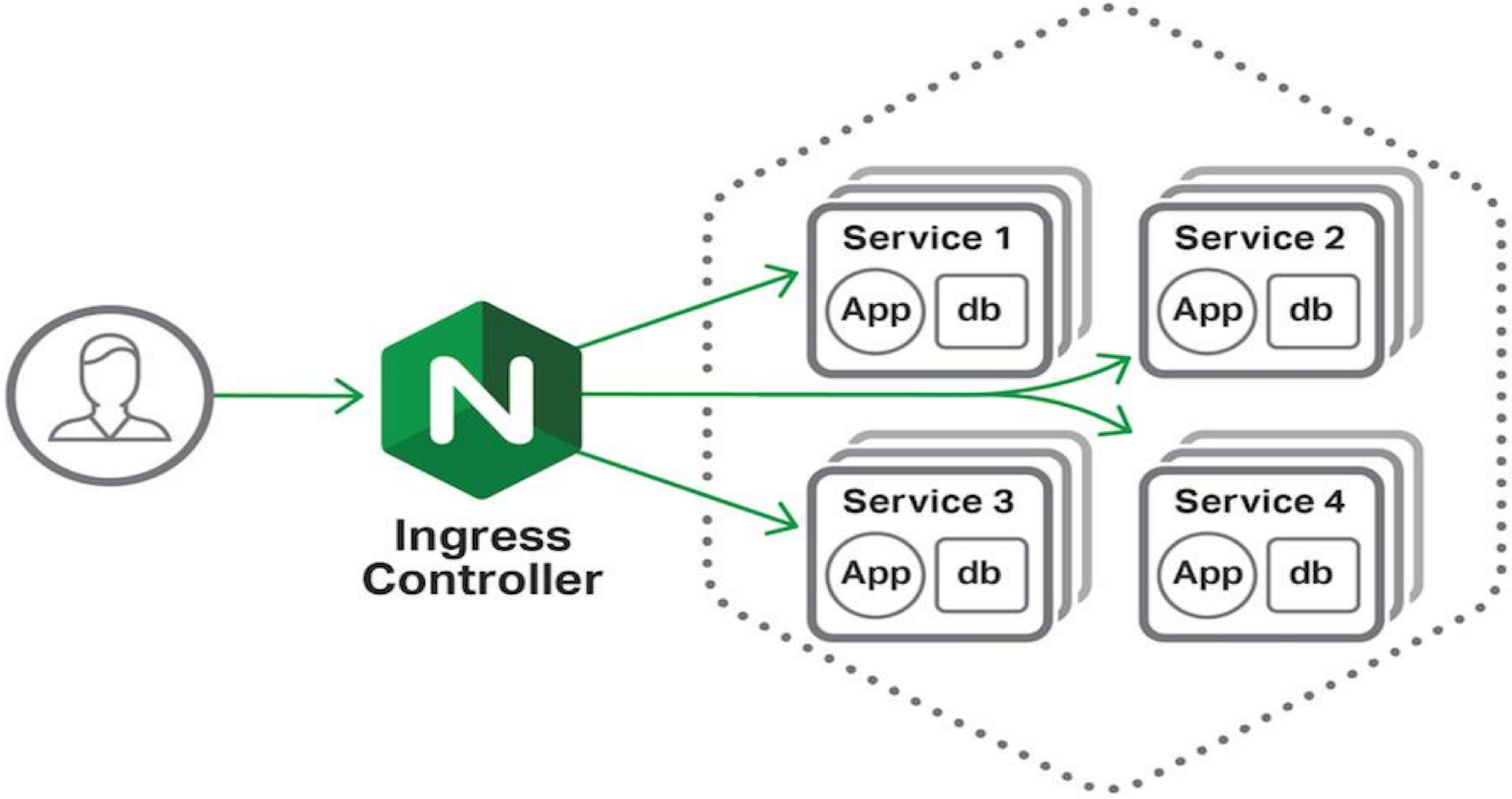

Overlay Network (Flannel/OpenVSwitch/Weave)
API Server
Scheduler
Controller Manager
etcd
Kubelet
Kube-proxy
C1
C2
C3
Cx
Container Runtime
Operating System
Infrastructure
Kubelet
Kube-proxy
C1
C2
C3
Cy
Container Runtime
Operating System
Infrastructure
Kubelet
Kube-proxy
C1
C2
C3
Cz
Container Runtime
Operating System
Infrastructure
Master
Node 1
Node 2
Node n
Physical Network

Kubernetes Resources

- To understand k8s *you must learn quite a lot of new terms and concepts*
- **Namespace**: most of the k8s resources lives inside a namespace
  - there is a default namespace that has no name
- **Pod**: equivalent to a container in basic scenarios
  - There can be more than one container in a pod, but this is for advanced scenarios (service mesh, sidecar containers, init containers)
- **Service**: to enable connectivity among pods and the outside world**: ClusterIp, NodePort, LoadBalancer**
- **Ingress rule**: to enable connectivity to the outside world in a more "efficient manner" (requires an ingress controller (a reverse proxy) installed in the cluster)
- **Configmap**: to store pieces of information in the cluster (basically for configuration: setting up env variables or config files to be injected in the container at startup)
- **Secrets**: protected pieces of data
- .. And more, but these are the basic ones

Ingress controller

- Not a basic component of k8s but required in real world scenarios
- Acts as a reverse proxy
- Routes request to services according to *ingress rule*
- Requires a single public ip address
- http and https only
- Many available: Ngnix, Kong, Traefik, etc ..
- Integrates with CertManager component to automatically provide and bind certificates from “Let’s Encrypt” to ingress rules

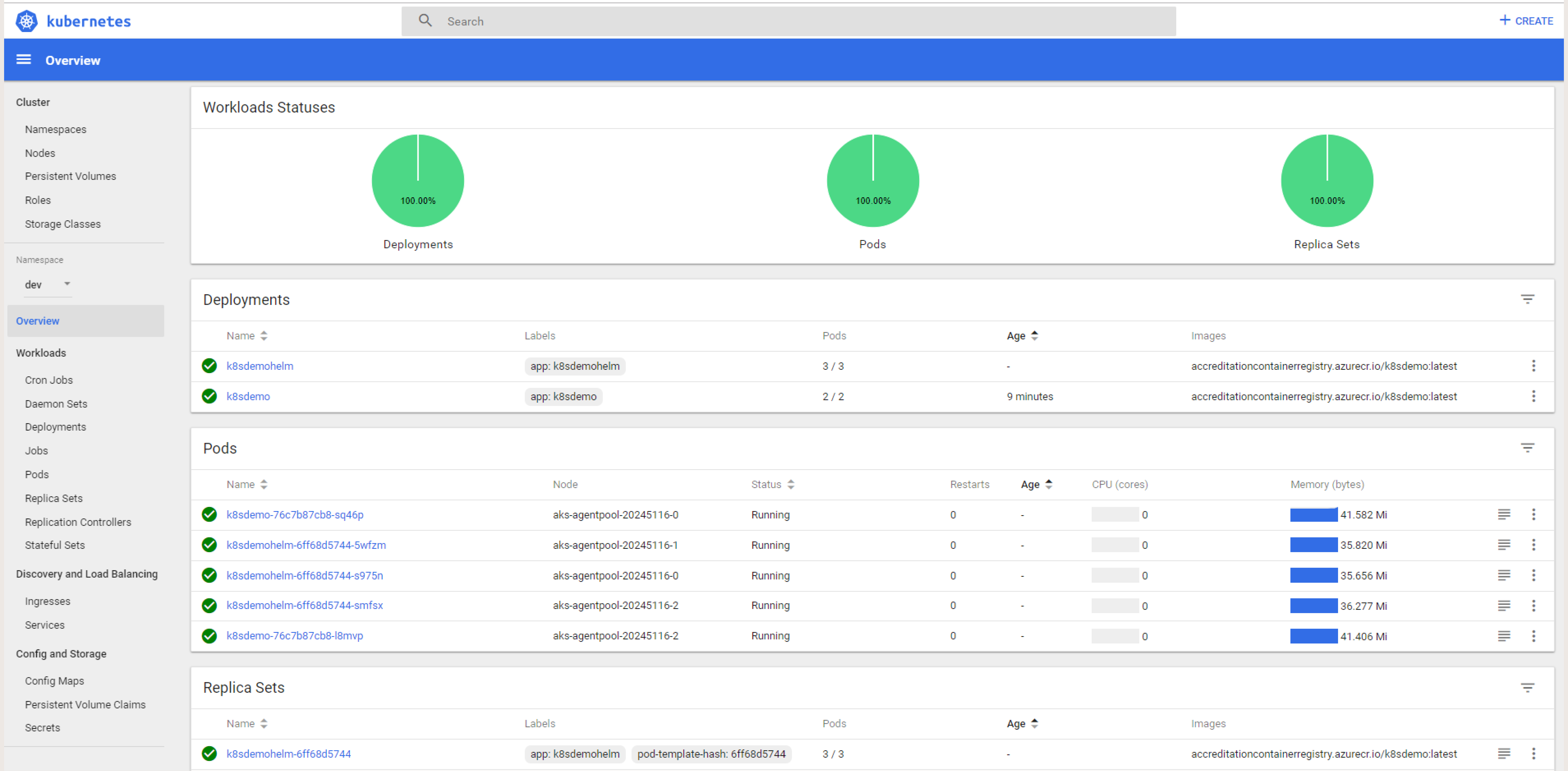


In-cluster communication

- **Cross pod communication does not need to pass through an ingress:, better to go through services**
- **Scenario : WEB Ap1 1 needs to call Web API 2**
  - Use the *service name* :
    - Same namespace: http://<targetservicename>/.. (doesn't change from dev to prod ☺)
    - Different namesapace: http://<targetservicename>.<target namespace where service resides>
  - Ingress is only for http/https (uses an http reverse proxy)
  - through services any traffic on any port is fine

  "mongodb://user:pwd@mongodbsrv.mongodbnamespace:27017 .."

K8s dashboard



Workloads Statuses

Deployments

	Name	Labels	Pods	Age	Images
✓	k8sdemohelm	app: k8sdemohelm	3 / 3	-	accreditationcontainerregistry.azurecr.io/k8sdemo:latest
✓	k8sdemo	app: k8sdemo	2 / 2	9 minutes	accreditationcontainerregistry.azurecr.io/k8sdemo:latest

Pods

	Name	Node	Status	Restarts	Age	CPU (cores)	Memory (bytes)
✓	k8sdemo-76c7b87cb8-sq46p	aks-agentpool-20245116-0	Running	0	-	0	41.582 Mi
✓	k8sdemohelm-6ff68d5744-5wfzm	aks-agentpool-20245116-1	Running	0	-	0	35.820 Mi
✓	k8sdemohelm-6ff68d5744-s975n	aks-agentpool-20245116-0	Running	0	-	0	35.656 Mi
✓	k8sdemohelm-6ff68d5744-smfsx	aks-agentpool-20245116-2	Running	0	-	0	36.277 Mi
✓	k8sdemo-76c7b87cb8-l8mvp	aks-agentpool-20245116-2	Running	0	-	0	41.406 Mi

Replica Sets

	Name	Labels	Pods	Age	Images
✓	k8sdemohelm-6ff68d5744	app: k8sdemohelm pod-template-hash: 6ff68d5744	3 / 3	-	accreditationcontainerregistry.azurecr.io/k8sdemo:latest

Interacting with the cluster

- **The API server is the only entry point to the cluster**
    - Likely you will never interact with the API server directly
- **Kubectl is the mainstream tool (command line)**
    - Inspect the cluster
    - Modify
    - Deploy (mainly via yaml files, more on it later)
- **The K8s dashboard**
    - Can accomplish many kubectl tasks using a web UI

Interacting with the cluster

kubectl

- kubectl: a command line tool, *the "current context" is the cluster it is connected to*
  - Use –context <clustrname> on commands to avoid switching active context
  - **Check the active context before sending commands**
  - Contexts info are stored in $HOME/.kube/config file (C:\Users\<username>\.kube on Windows)
  - Kubectl config get-contexts -> active one is marked with an *
  - Kubectl config use-context <context name> : see active contexts
  - All commands are in the scope of the default namespace if the –n or --all-namespaces is not specified
    - --all-namespaces , -n <namespace name>
    - kubectl get pods --all-namespaces
    - kubectl get pod <pod name> -o wide -n <namespace>
    - kubectl describe pod <pod name> -n <namespace>

Interacting with the cluster

kubectl

- Port forward
    - e.g. to connect to a mongo instance in the cluster from your local machine
      kubectl port-forward -n mongodbnamespace svc/mongodb <localport>:<internalport>
- Run command in container kubectl exec <podname> -n <namespace> -- <command>
- Log into a pod: kubectl exec -n <namespace> -it <podname> -- /bin/bash (if bash installed in the image)
- Copy files from pod to local (and vice versa)
    - kubectl cp <namespace>/<podname>:/app ./app
- Watch pod logs in streaming mode: kubectl logs -f <podname> (-f for follow)
- Kubernets dashboard

Deploy to cluster– walkthrough

YAML

- Some basic commands can be sent straight through command line (e.g. create namespace)
- Recommended way is to use a declarative approach. **Kubectl apply –f <filename>**
  - you tell the cluster what you want, not how accomplish what you want
  - desired state expressed through yaml files
  - Infrastructure as code (as azure arm templates)
- Want to edit on the fly ?
  kubectl edit deployment/<deployment name> -n <namespace> (download definition, open editor, push on save)
  or do the same using the dashboard

deltatre

Deploy app to cluster

Create Namespace

```
kubectl create namespace dev
```

Deploy app to cluster

- To deploy an app running on the cluster to the outside world, you need at minimum 3 k8s resources defined in a yaml file
  - **Deployment** (what image to deploy, how many replicas, etc..)
  - **Service** (ClusterIp)
  - **Ingress Rule** (requires an ***ingress controller*** available (installed up front) in the cluster
    - Not strictly required, services can be of type LoadBalancer, but this implies a separate public IP for each web app, which is not an optimal solution, cloud providers put limits on public ip provisioning
  - **configMaps** : define "per environment" settings for you app (env variables or files injected into the container)
    - For sure you don't want to make an image for each environment
- WATCH OUT: kubectl apply will do nothing if yaml file has not changed ("problematic" for deployment yaml: use of latest tag, changes in config map)

Deploy to cluster

Config Map

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: k8sdemo-config-map
  namespace: dev
data:
  ASPNETCOREENVIRONMENT: "kind-env“
  appsettings.kind-env.json: |-
    {
      "MySetting": "kind-setting"
    }
```

Deploy to cluster

Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: k8sdemo
 namespace: dev
 labels:
  app: k8sdemo
spec:
 replicas: 1
 selector:
  matchLabels:
   app: k8sdemo
 template:
  metadata:
   labels:
    app: k8sdemo
  spec:
   containers:
   - name: k8sdemo
    image: sabbadino/k8sdemo:1.2
    imagePullPolicy: Always
    resources: null
    ports:
     - name: http
      containerPort: 80
    env:
    - name: ASPNETCORE_ENVIRONMENT
     valueFrom:
       configMapKeyRef:
        name: k8sdemo-config-map
        key: ASPNETCOREENVIRONMENT
    volumeMounts:
    - name: settings-volume
     mountPath: /app/appsettings.kind-env.json
     subPath: appsettings.kind-env.json
   volumes:
   - name: settings-volume
    configMap:
     name: k8sdemo-config-map
```

Deploy to cluster

Service

```
apiVersion: v1
kind: Service
metadata:
  name: k8sdemo
  namespace: dev
spec:
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 80
  selector:
    app: k8sdemo //selector for pods, can be a list
```

Type: ClusterIp, NodePort, LoadBalancer

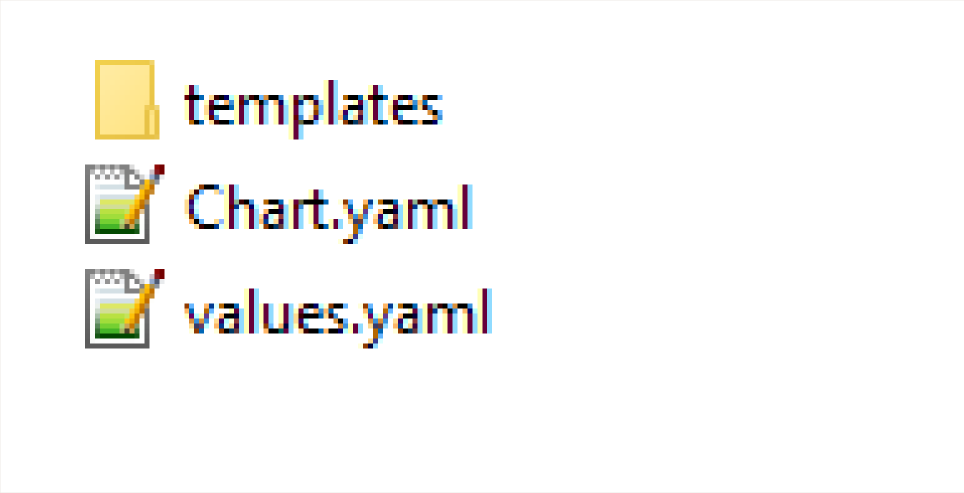
Deploy to cluster

Ingress

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: k8sdemo
  namespace: dev
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
    - host: localhost
      http:
        paths:
          - path: /demoapi(/|$)(.*)
            backend:
              serviceName: k8sdemo
              servicePort: 80
```

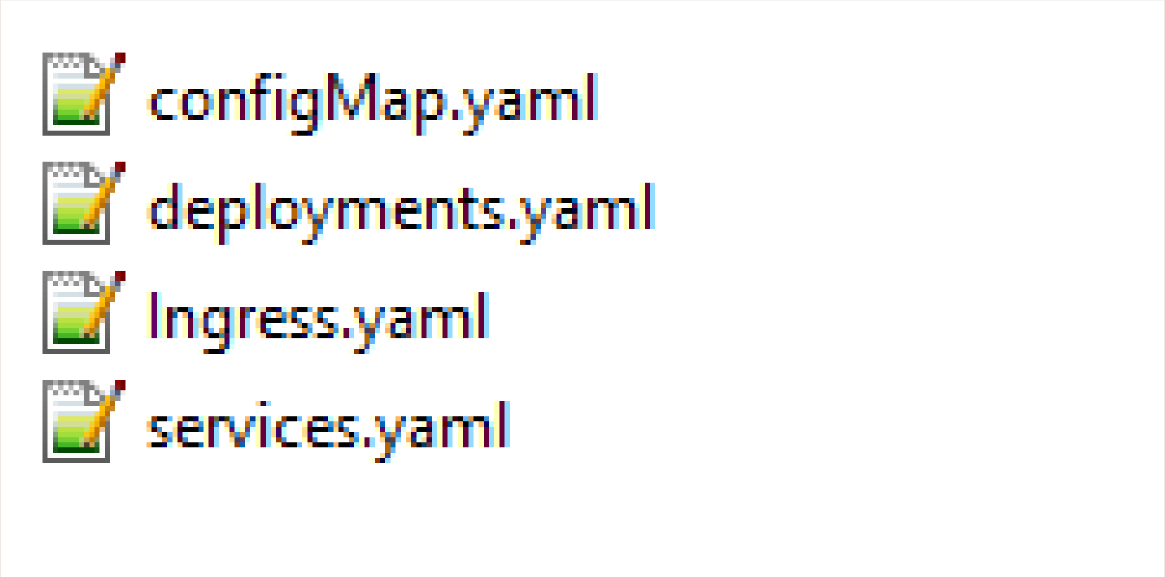
Deploy to cluster : AKS with HELM

- K8s package manager
- Groups yaml files into a *Chart*
- **Token replacement, if conditions, iterations on templated yaml**
  - Similar to azure arm templates
- Many public repo with helm deployment of different resources
  - https://hub.helm.sh/
  - https://github.com/helm/charts
- Helm is a command line tool like kubectl
  - In version 3 it does not requires a server counter part (tiller)



templates
Chart.yaml
values.yaml

Template folder



configMap.yaml
deployments.yaml
Ingress.yaml
services.yaml

Ingress.yaml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: k8sdemohelm
  namespace: {{ .Values.app.namespace }}
  annotations:
```

Value file

```
app:
  namespace: dev
```

```
helm install <deployment-name> <chart-folder> -f myvalues.yaml –f anothervaluefile.yaml
```

values.yaml file in root folder of the chart provides defaults

Values can also be provided via command line, e.g. : --set app.namespace=dev2

Values evaluation (override) from left to right

Tips: use upgrade –install for idempotency

```
helm upgrade --recreate-pods --install k8sdemowithhelm demoapp -f myvalues.yaml
```

Delete a deployed chart: helm delete <deployment-name>

(persistent) Storage

- Pods are ephemeral, when restarted what they wrote on disk is lost
- K8s offers an abstraction to hide as much as possible specific details of external (cloud) storage provider
  - Think twice before going to PRD using K8s storage

(persistent) Storage

- **Storage class**
  - Link to cloud or on premise specific storage options, e.g.
    - Azure disk single *ReadWriteOnce*, more performance
    - Azure files (Storage Account) – *ReadWriteMany*, less performance
    - Watch out for ReclaimPolicy
- **Persistent Volume (PV)**
  - Actual storage instance
- **Persistent Volume Claim**
  - Storage Requirement (refer to a storage class)
- Yaml Deployment refer to a storage class (never to a PV explicitly)
  - If no storage class is defined the default one is used (if defined)
- PV can be provisioned dynamically or created upfront by the administrator

RBAC Security

Role Based Access Security

- **Authorization model controlling access to the API server**
  - What endpoints and what verbs
- **Entered stable mode in 1.8**
- **Defines four top-level resources**
  - Role (namespace specific) and ClusterRole (cluster wide)
    - contains rules that represent a set of permissions
    - ClusterRoles can be aggregated
  - RoleBinding and ClusterRoleBinding
    - Bind Roles / ClusterRoles to subjects : **serviceAccounts**
    - Each pod runs under a serviceAaccount

Subjects: serviceAccount

- ServiceAccount: Identity of a Pod
  - Can be specified in the yaml
  - If not present "serviceAccountName": "default“
  - If a pod requires access to k8s api server, bind its seviceAccount to a proper (Cluster)Role with a (Cluster)RoleBinding: ***you cannot ignore RBAC (if enabled) for the dashboard or any other pod accessing the k8s API server***
- Default RBAC policies grant scoped permissions to control-plane components, nodes, and controllers, *but grant no permissions to service accounts outside the kube-system namespace*
  - Many headaches when RBAC was introduced, many things (samples, yaml, helm charts) available on the web stopped working

In next sessions

- Readyness & Liveness probes
- Pods resources requests and limits
- StatefullSets
- CertManager (acquire and renew automaticaaly certs from letsEnrypt for htpps)
- CronJobs
- Init contaners
- And lot more ☺