

Kubernetes configuration and deployment

A practical look at using Kubernetes

Recapping yesterday

- We looked at some considerations that need to be made when scaling an application
- There are many options to scale a monolith, but at a certain point, you need to redesign the system
- A shift to microservices allows an application to scale
 - It comes with the challenge of redefining the entire application to be distributed
- Kubernetes is a tool that helps manage distributed systems
 - It manages containers for you
- Kubernetes uses a client-server model to communicate and manage itself
- Containers are deployed in pods, which run on nodes and are managed by controllers. With the controller plane being the master.

Overview

- K8s basics
 - Creating a cluster
 - Deployment
 - Peeking inside
 - Services
 - Dns
 - Ingress
- Azure Kubernetes Service
- Securing k8s

K8s basics

Going through the important aspects

K8s basics



kubectl - the most important command

- Kubernetes provides a command line tool for communicating with a Kubernetes cluster's control plane, using the Kubernetes API.
 - This is the *kubectl* command
 - Often pronounced as "kube cuttle"
- For configuration, kubectl looks for a file named config in the *\$HOME/.kube* directory.
 - You can specify other *kubeconfig* files by setting the KUBECONFIG environment variable
 - or by setting the *--kubeconfig* flag.

K8s basics



kubectl template

- `kubectl [command] [TYPE] [NAME] [flags]`
 - **command**: Specifies the operation that you want to perform on one or more resources, for example *create*, *get*, *describe*, *delete*.
 - **TYPE**: Specifies the resource type, for example *pod*
 - **NAME**: Specifies the name of the resource. Can be omitted to show all resources.
 - **flags**: Specifies optional flags.
- There are many commands, we will look at the most important ones as we are going
 - Official [reference doc](#)
 - Official [cheat sheet](#)
 - “The Ultimate Kubectl Commands Cheat Sheet” [article](#)

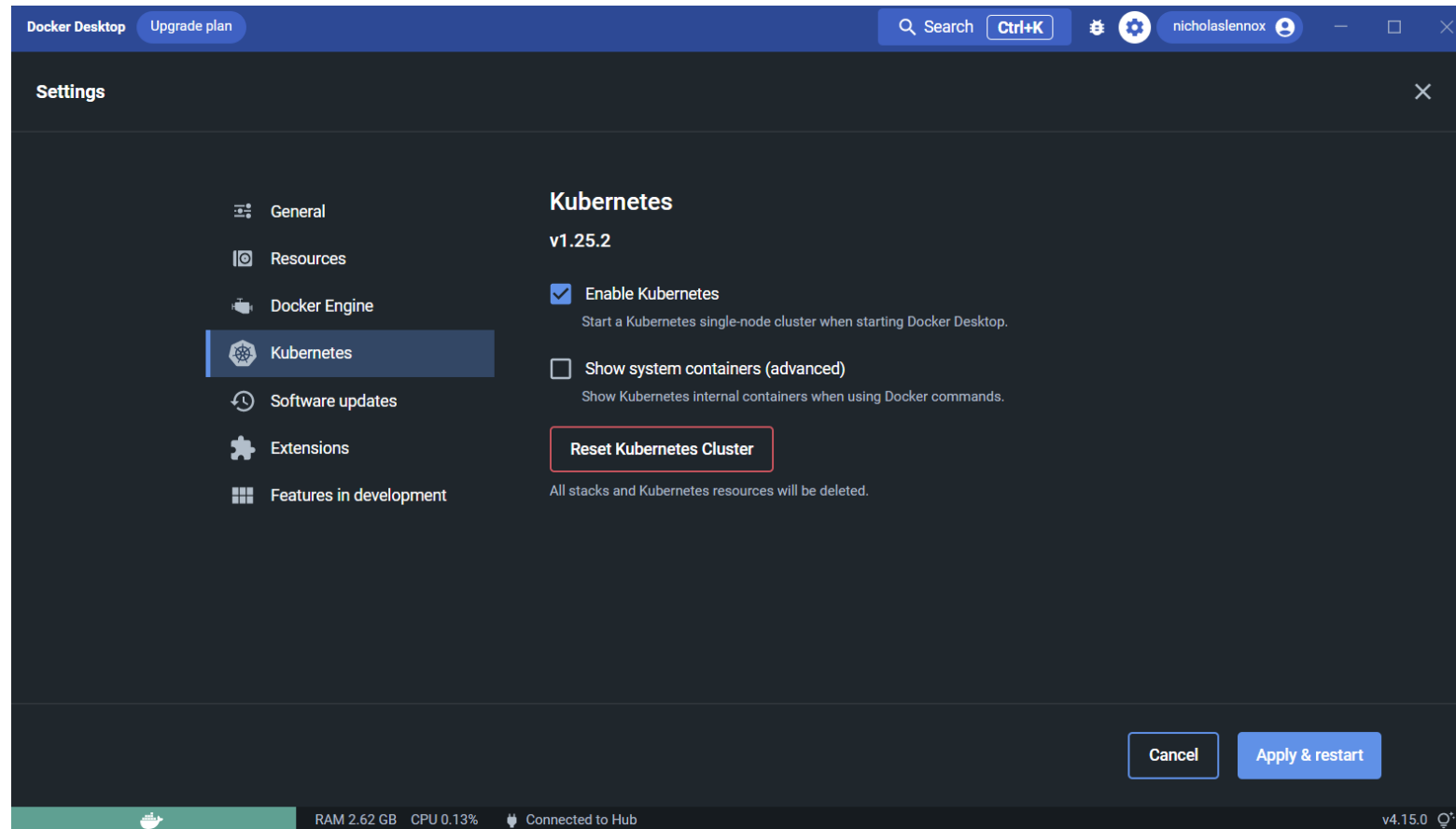
K8s basics – creating a cluster



How to create a cluster?

- Recall, a cluster is a set of computers working together as a single system.
- There are [multiple ways](#) of creating and deploying clusters
 - *kind* requires Docker and lets you run clusters on your machine (Linux)
 - *minikube* is a standalone tool to run a cluster (Linux VM)
 - *kubeadm* is a tool for self-hosted production deployments (Linux VM)
 - *Docker Desktop* has a built-in cluster that uses your host machine as a node.
 - We will do this, its simpler for windows ([link](#) to doc)

K8s basics – creating a cluster



Enabling k8s in Docker Desktop

K8s basics – creating a cluster

Images [Give feedback](#)

An image is a read-only template with instructions for creating a Docker container. [Learn more](#)

LOCAL REMOTE REPOSITORIES

Refresh to see disk usage 23 images Last refresh: Never

Search

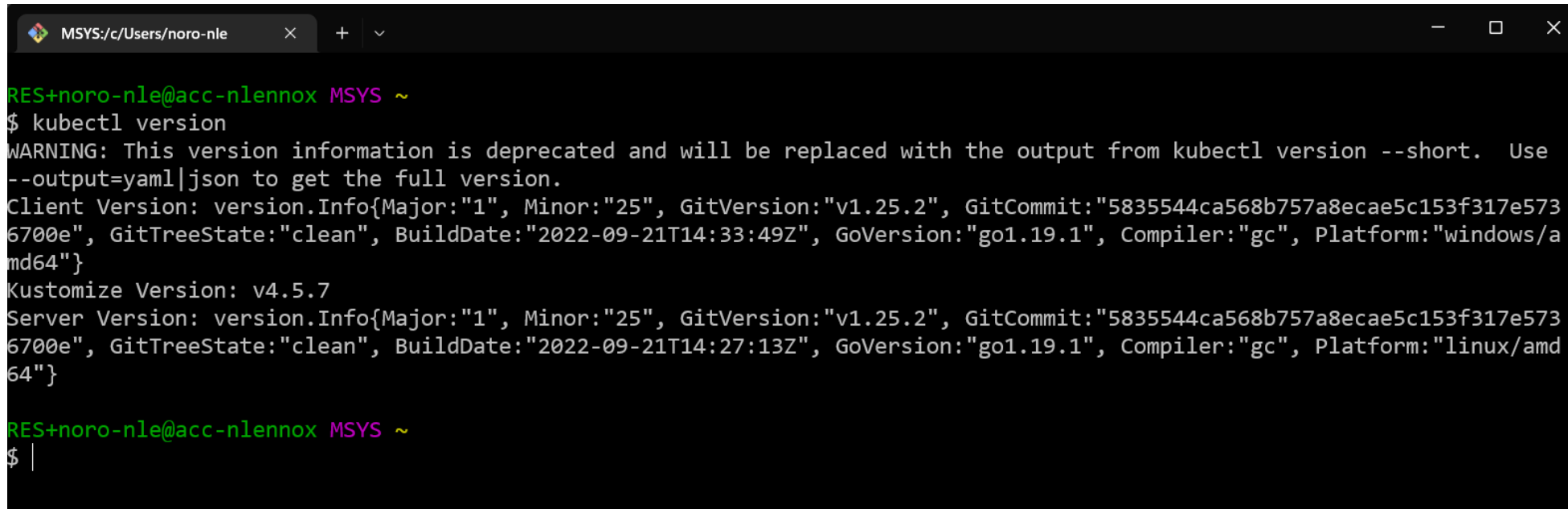
| <input type="checkbox"/> | NAME | TAG | STATUS | CREATED | SIZE | ACTIONS |
|--------------------------|---|----------------|--------|--------------|-----------|--|
| <input type="checkbox"/> | hubproxy.docker.internal:5000/docker/desktop-kubernetes 09d7e1dbc2c4 | kubernetes-v1. | Unused | 2 months ago | 363.32 MB | ▶ ⋮ 🗑️ |
| <input type="checkbox"/> | k8s.gcr.io/kube-apiserver 97801f839490 | v1.25.2 | Unused | 2 months ago | 127.73 MB | ▶ ⋮ 🗑️ |
| <input type="checkbox"/> | k8s.gcr.io/kube-controller-manager dbfceb93c69b | v1.25.2 | Unused | 2 months ago | 117.1 MB | ▶ ⋮ 🗑️ |
| <input type="checkbox"/> | k8s.gcr.io/kube-scheduler ca0ea1ee3cfd | v1.25.2 | Unused | 2 months ago | 50.58 MB | ▶ ⋮ 🗑️ |
| <input type="checkbox"/> | k8s.gcr.io/kube-proxy 1c7d8c51823b | v1.25.2 | Unused | 2 months ago | 61.69 MB | ▶ ⋮ 🗑️ |
| <input type="checkbox"/> | k8s.gcr.io/pause 4873874c08ef | 3.8 | Unused | 6 months ago | 711.18 KB | ▶ ⋮ 🗑️ |
| <input type="checkbox"/> | k8s.gcr.io/etcd a8a176a5d5d6 | 3.5.4-0 | Unused | 6 months ago | 299.52 MB | ▶ ⋮ 🗑️ |
| <input type="checkbox"/> | k8s.gcr.io/coredns | v1.0.2 | Unused | 6 months ago | 10.9 MB | ▶ ⋮ 🗑️ |

Showing 23 items



Looking at the images

K8s basics – creating a cluster

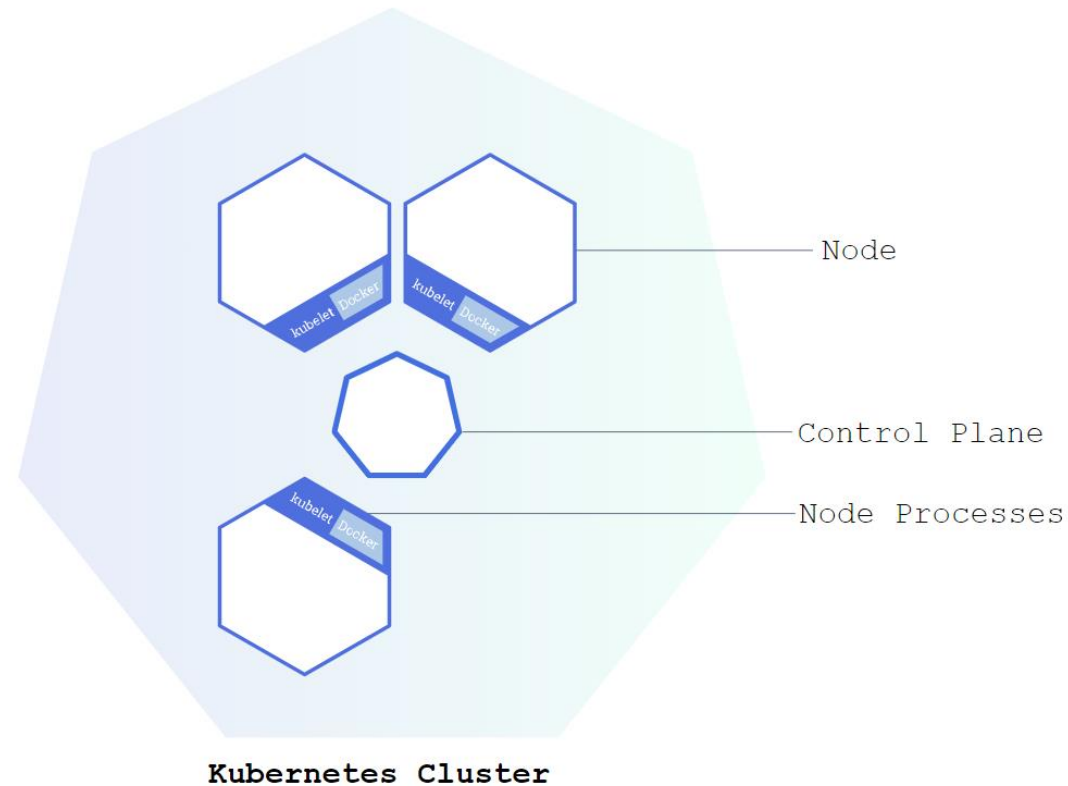


```
MSYS:/c/Users/noro-nle  x + v
RES+no-ro-nle@acc-nlennox MSYS ~
$ kubectl version
WARNING: This version information is deprecated and will be replaced with the output from kubectl version --short. Use
--output=yaml|json to get the full version.
Client Version: version.Info{Major:"1", Minor:"25", GitVersion:"v1.25.2", GitCommit:"5835544ca568b757a8ecae5c153f317e573
6700e", GitTreeState:"clean", BuildDate:"2022-09-21T14:33:49Z", GoVersion:"go1.19.1", Compiler:"gc", Platform:"windows/a
md64"}
Kustomize Version: v4.5.7
Server Version: version.Info{Major:"1", Minor:"25", GitVersion:"v1.25.2", GitCommit:"5835544ca568b757a8ecae5c153f317e573
6700e", GitTreeState:"clean", BuildDate:"2022-09-21T14:27:13Z", GoVersion:"go1.19.1", Compiler:"gc", Platform:"linux/amd
64"}
RES+no-ro-nle@acc-nlennox MSYS ~
$ |
```



Checking if kubectl is installed

K8s basics – creating a cluster



What we have done ([src](#))

K8s basics – creating a cluster

```
RES+no-ro-nle@acc-nlennox MSYS ~  
$ kubectl cluster-info  
Kubernetes control plane is running at https://kubernetes.docker.internal:6443  
CoreDNS is running at https://kubernetes.docker.internal:6443/api/v1/namespaces/kube-system/services/kube-dns:dn  
s:dn  
s:dn/proxy  
  
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

```
RES+no-ro-nle@acc-nlennox MSYS ~  
$ kubectl get nodes  
NAME             STATUS    ROLES    AGE   VERSION  
docker-desktop   Ready    control-plane   13m   v1.25.2
```



Inspecting our cluster

Any questions?

We have just started, but I want to make sure we are on the same page

K8s basics – deployment



Running a pod

- The first logical step is to run a container
- Recall, we do this by asking Kubernetes to run a pod for us with a specific image
 - This can be done with or without a file (YAML)
- We will first do it without a file
- Then with a file
- We want to run a simple app that greets the user
 - `nicholaslennox/howzit-greeter`

K8s basics – deployment

```
RES+noronle@acc-nlennox MSYS ~  
$ kubectl run howzit-greeter --image=nicholaslennox/howzit-greeter  
pod/howzit-greeter created  
  
RES+noronle@acc-nlennox MSYS ~  
$ kubectl get pods  
NAME          READY   STATUS    RESTARTS   AGE  
howzit-greeter 1/1     Running   0           8s  
  
RES+noronle@acc-nlennox MSYS ~  
$ kubectl get pods -o wide  
NAME          READY   STATUS    RESTARTS   AGE   IP          NODE          NOMINATED NODE   READINESS GATES  
howzit-greeter 1/1     Running   0           2m1s  10.1.0.7    docker-desktop <none>           <none>
```



Asking k8s to run a container

K8s basics – deployment



The most direct method

- This is known as creating **imperatively**
- Everything is typed out and done in a single line
 - It needs to be retyped to replicate
- It also won't watch for failed pods
- You also can't make replicas (need a separate ReplicaSet)
- *Do you think this is a good method?*

K8s basics – deployment



Using deployments

- Deployment is a specialized term in the context of Kubernetes
 - A file that defines a pod's desired behaviour or characteristics
 - It can be created imperatively like we did before
- Deployments help manage the pods, and keep them in the desired state with controllers.
 - It helps manage the replicas as well, something you cant do directly
- You shouldn't make pods directly unless there is a specific reason
 - Kubernetes recommends production to be with deployments and **declarative** (through a file)

K8s basics – deployment

```
RES+noronle@acc-nlennox MSYS ~  
$ kubectl create deployment howzit-greeter --image=nicholaslennox/howzit-greeter  
deployment.apps/howzit-greeter created  
  
RES+noronle@acc-nlennox MSYS ~  
$ kubectl get deployments  
NAME                READY    UP-TO-DATE    AVAILABLE    AGE  
howzit-greeter      1/1      1             1            7s  
  
RES+noronle@acc-nlennox MSYS ~  
$ kubectl get pods  
NAME                                READY    STATUS    RESTARTS    AGE  
howzit-greeter-8468555d69-b9vg2    1/1     Running   0           12s
```



Imperatively making a deployment

K8s basics – deployment

```

RES+noroff-nle@acc-nlennox MSYS ~
$ kubectl describe deployments howzit-greeter
Name:                howzit-greeter
Namespace:           default
CreationTimestamp:
Labels:               app=howzit-greeter
Annotations:          deployment.kubernetes.io/revision: 1
Selector:             app=howzit-greeter
Replicas:            1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:        RollingUpdate
MinReadySeconds:     0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=howzit-greeter
  Containers:
    howzit-greeter:
      Image:      nicholaslennox/howzit-greeter
      Port:       <none>
      Host Port:  <none>
      Environment: <none>
      Mounts:      <none>
      Volumes:     <none>
  Conditions:
    Type           Status  Reason
    ----           -
    Available      True    MinimumReplicasAvailable
    Progressing    True    NewReplicaSetAvailable
  OldReplicaSets:  <none>
  NewReplicaSet:   howzit-greeter-8468555d69 (1/1 replicas created)
  Events:
    Type           Reason             Age           From              Message
    ----           -
    Normal         ScalingReplicaSet   4m46s        deployment-controller Scaled up replica set howzit-greeter-8468555d69 to 1

```



Describing our deployment

K8s basics – deployment



Moving to declarative

- When looking deeper with describe, we saw a template.
- A Kubernetes **manifest** is a YAML file that describes each component or resource of your deployment and the state you want once applied
- For deployments, we describe our desired state and the controller helps reach that state
- All manifests have the same template
 - We differentiate with the *kind* flag

K8s basics – deployment



Describing our deployment

- We need to make a YAML file and populate it
- We can take the [deployments](#) page in Kubernetes docs as reference
- We won't supply all the arguments yet, but we will start with:
 - Names, replicas, images, ports
 - Just to redo what we did imperatively

K8s basics – deployment

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ ls -a
./  ../  deployments/

RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ cd deployments

RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop/deployments
$ ls -a
./  ../  howzit-greeter-deployment.yaml

RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop/deployments
$ |
```



Our folder structure

K8s basics – deployment

Yes the name
is repeated 4
times

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop/deployments
$ cat howzit-greeter-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: howzit-greeter-deployment
  labels:
    app: howzit-greeter
spec:
  replicas: 3
  selector:
    matchLabels:
      app: howzit-greeter
  template:
    metadata:
      labels:
        app: howzit-greeter
    spec:
      containers:
        - name: howzit-greeter
          image: nicholaslennox/howzit-greeter
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
```



Our description

K8s basics – deployment



Creating our deployment

- *kubectl apply -f howzit-greeter-deployment.yaml*
- We use the **apply** action to declaratively create our deployment
 - Apply is the preferred method as you tell k8s your desired state
 - It will update any existing resources to meet the needs
 - **Create** is you doing it directly, it will throw an error if it already exists
- We use the **-f** flag to say where the file is

K8s basics – deployment

```

RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop/deployments
$ kubectl describe deployments howzit-greeter-deployment
Name:                howzit-greeter-deployment
Namespace:           default
CreationTimestamp:    2022-01-10T15:10:10Z
Labels:               app=howzit-greeter
Annotations:          deployment.kubernetes.io/revision: 1
Selector:             app=howzit-greeter
Replicas:             3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:         RollingUpdate
MinReadySeconds:      0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=howzit-greeter
  Containers:
    howzit-greeter:
      Image:      nicholaslennox/howzit-greeter
      Port:       80/TCP
      Host Port:  0/TCP
      Environment: <none>
      Mounts:      <none>
      Volumes:     <none>
  Conditions:
    Type           Status  Reason
    ----           -
    Available       True    MinimumReplicasAvailable
    Progressing     True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   howzit-greeter-deployment-66c9bb7d84 (3/3 replicas created)
Events:
  Type           Reason             Age   From                  Message
  ----           -
  Normal        ScalingReplicaSet   5m15s deployment-controller  Scaled up replica set howzit-greeter-deployment-66c9bb7d84 to 3

```



Describing our deployment again

K8s basics – deployment

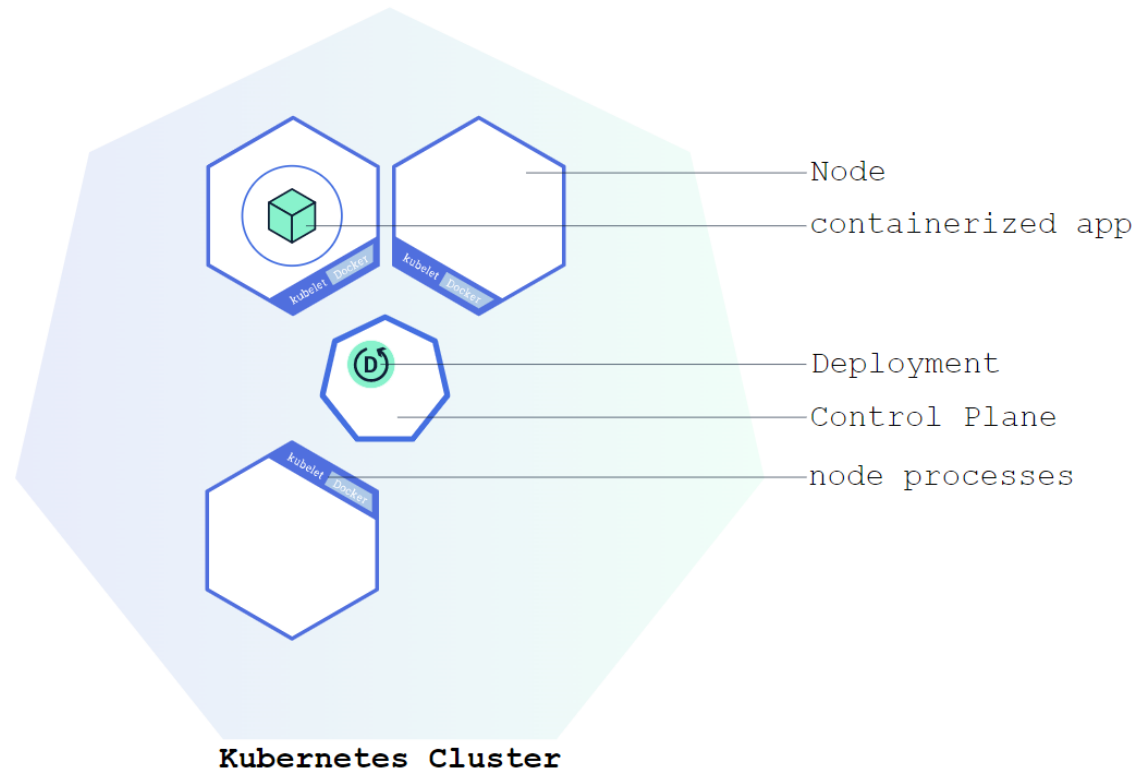
```
RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop/deployments
$ kubectl get deployments
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
howzit-greeter-deployment          3/3      3             3            7m53s

RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop/deployments
$ kubectl get pods
NAME                                READY    STATUS    RESTARTS    AGE
howzit-greeter-deployment-66c9bb7d84-2xjc7    1/1      Running    0           7m58s
howzit-greeter-deployment-66c9bb7d84-59547    1/1      Running    0           7m58s
howzit-greeter-deployment-66c9bb7d84-6v4mb    1/1      Running    0           7m58s
```



Inspecting our deployment and pods

K8s basics – deployment

































Where we currently are ([src](#))

K8s basics – deployment

Containers [Give feedback](#)

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. [Learn more](#)

☐ Only show running containers

| <input type="checkbox"/> | NAME | IMAGE | STATUS | PORT(S) | STARTED | ACTIONS |
|--------------------------|---|--|---------|---------|--|---|
| <input type="checkbox"/> |  k8s_POD_howzit-greeter-deployment- 4d47e7c2e903  | k8s.gcr.io/pause:3.8 | Running | | 43 minutes ago  |   |
| <input type="checkbox"/> |  k8s_POD_howzit-greeter-deployment- 87e48e3c1216  | k8s.gcr.io/pause:3.8 | Running | | 43 minutes ago  |   |
| <input type="checkbox"/> |  k8s_POD_howzit-greeter-deployment- 99650e63af9a  | k8s.gcr.io/pause:3.8 | Running | | 43 minutes ago  |   |
| <input type="checkbox"/> |  k8s_howzit-greeter_howzit-greeter-de a8724fdeb1a4  | nicholaslennox/howzit-gi | Running | | 43 minutes ago  |   |
| <input type="checkbox"/> |  k8s_howzit-greeter_howzit-greeter-de eefc873414e7  | nicholaslennox/howzit-gi | Running | | 43 minutes ago  |   |
| <input type="checkbox"/> |  k8s_howzit-greeter_howzit-greeter-de 5526cce65130  | nicholaslennox/howzit-gi | Running | | 43 minutes ago  |   |



Herding our cats

K8s basics – deployment



Summary of commands so far

- *kubectl get ...*
 - Basic information gathering
- *kubectl get ... -o wide*
 - More information
- *kubectl describe ...*
 - Verbose details
- *kubectl apply -f ...*
 - Creates or updates the cluster with the desired state
 - Can be anything (we did deployments)
 - If you need to update the deployment, change the file and run *apply* again

Any questions?

Before we do our first try it out



Try it out

1. Enable k8s through Docker Desktop
 - a) Check its installed by running *kubectl version*
2. Create a YAML file for a deployment (e.g., in VSCode)
 - a) You can deploy the image used in the slides if you want (nicholaslennox/howzit-greeter)
 - b) You can use the [deployments](#) page to get a template
3. Create the deployment with *kubectl apply -f*
4. Inspect your deployment with *describe*
 - a) You can also get deployments and pods

K8s basics – peaking inside



Invisible by default

- All our cluster activity happens isolated and we cannot access it without either directly exposing ports or using proxies
- Kubernetes comes with a proxy that allows you to interact with the pods inside
 - This is not the way you should expose your apps, that is later
 - Its just a way to peak in and look at what is going on
- Done via *kubectl proxy*

K8s basics – peaking inside



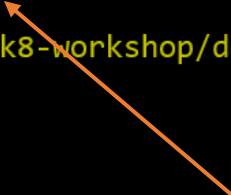
Inspecting our cluster

- Kubernetes uses services to expose applications
 - We will cover this later, we just need a single command now
- The kubectl command is interacting with an API, and we can expose all that
- We can use kubectl get svc to see all services
 - We will just have the one for the entire cluster right now
 - This is what kubectl interacts with

K8s basics – peaking inside

```
RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop/deployments
$ kubectl get svc
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
kubernetes    ClusterIP     10.96.0.1     <none>         443/TCP    160m

RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop/deployments
$ kubectl describe svc kubernetes
Name:          kubernetes
Namespace:     default
Labels:        component=apiserver
               provider=kubernetes
Annotations:    <none>
Selector:      <none>
Type:          ClusterIP
IP Family Policy: SingleStack
IP Families:   IPv4
IP:            10.96.0.1
IPs:           10.96.0.1
Port:          https 443/TCP
TargetPort:    6443/TCP
Endpoints:     192.168.65.4:6443
Session Affinity: None
Events:        <none>
```



Looking at our services

K8s basics – peaking inside

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop/deployments
$ curl http://localhost:8001/api/v1/namespaces/default
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "default",
    "uid": "0b4ab50d-8e9f-4add-b967-12ef24149a7f",
    "resourceVersion": "191",
    "creationTimestamp": "2022-12-03T11:34:17Z",
    "labels": {
      "kubernetes.io/metadata.name": "default"
    }
  },
  "managedFields": [
    {
      "manager": "kube-apiserver",
      "operation": "Update",
      "apiVersion": "v1",
      "time": "2022-12-03T11:34:17Z",
      "fieldsType": "FieldsV1",
      "fieldsV1": {
        "f:metadata": {
          "f:labels": {
            ".": {},
            "f:kubernetes.io/metadata.name": {}
          }
        }
      }
    }
  ]
}
```

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```



Enabling the proxy

K8s basics – peaking inside

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop/deployments
$ curl http://localhost:8001/api/v1/namespaces/default/pods
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "13382"
  },
  "items": [
    {
      "metadata": {
        "name": "howzit-greeter-deployment-66c9bb7d84-2xjc7",
        "generateName": "howzit-greeter-deployment-66c9bb7d84-",
        "namespace": "default",
        "uid": "80525f50-77a9-47a6-b648-c15683c8da0c",
        "resourceVersion": "8607",
        "creationTimestamp": "2022-12-03T13:22:40Z",
        "labels": {
          "app": "howzit-greeter",
          "pod-template-hash": "66c9bb7d84"
        },
        "ownerReferences": [
          {
            "apiVersion": "apps/v1",
            "kind": "ReplicaSet",
            "name": "howzit-greeter-deployment-66c9bb7d84",
            "uid": "c7c8aac8-a85a-4bc8-b2c1-f44ff4f75a38",
            "controller": true,
            "blockOwnerDeletion": true
          }
        ]
      }
    }
  ]
}
```

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```



Looking at our pods

K8s basics – services



What is a service in k8s?

- A logical abstraction to expose a set of pods as a network service
- Deployments make sure to have the desired state of the pods
 - E.g., replicas and resources
 - IP addresses change as pods are created and destroyed
- Services create a static IP (and policies) so your pods can be accessed by others via an internal DNS
 - This avoids the challenge of connecting two pods
 - E.g., front-end and back-end
- This is where we start creating the “microservice”

K8s basics – services



What makes a service

- To provide discovery and routing between pods, a service needs to know what pods (or deployments) should be grouped
 - This is done via selectors and labels
 - It can be done without selectors (this is for communication to another cluster or namespace)
- It also needs to control ports
 - Assigned port numbers and mapping to external ports
 - So services can be accessed from outside the cluster as well

K8s basics – services



Service types

- **ClusterIP** (default) - Exposes the Service on an internal IP in the cluster.
 - This type makes the Service only reachable from within the cluster.
- **NodePort** - Exposes the Service on the same port of each selected Node in the cluster using NAT.
 - Makes a Service accessible from outside the cluster using `<NodeIP>:<NodePort>`.
 - Superset of ClusterIP.

K8s basics – services



Service types cont.

- **LoadBalancer** - Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP to the Service.
 - Superset of NodePort.
- **ExternalName** - Maps the Service to the contents of the externalName field (e.g. foo.bar.example.com)
 - By returning a CNAME record with its value.
- Note: if you don't use selectors, you can manually map externalName or manually assign IP addresses

K8s basics – services

```
RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl get deployments
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
howzit-greeter-deployment          3/3      3              3             21h

RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl expose deployment howzit-greeter-deployment --type=NodePort --name=greeter-service
service/greeter-service exposed

RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl get services
NAME            TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
greeter-service NodePort     10.100.53.127 <none>         80:31740/TCP     7s
kubernetes      ClusterIP    10.96.0.1     <none>         443/TCP          23h
```



Starting simple with imperative

K8s basics – services

```
RES+no-ro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl describe service greeter-service
Name: greeter-service
Namespace: default
Labels: app=howzit-greeter
Annotations: <none>
Selector: app=howzit-greeter
Type: NodePort
IP Family Policy: SingleStack
IP Families: IPv4
IP: 10.100.53.127
IPs: 10.100.53.127
LoadBalancer Ingress: localhost
Port: <unset> 80/TCP
TargetPort: 80/TCP
NodePort: <unset> 31740/TCP
Endpoints: 10.1.0.15:80,10.1.0.16:80,10.1.0.17:80
Session Affinity: None
External Traffic Policy: Cluster
Events: <none>
```

What we labelled
our deployment

Recall, we can access
via
<NodeIP>:<NodePort>



Describing our service

K8s basics – services



Accessing our service

- Because we use Docker Desktop, which creates a single node from our machine
 - We can use localhost 😊
- NodePort services are not really designed for external internet traffic (load balancers are more appropriate)
 - But they are good for local testing
- Lets go see our application
 - `curl http://<node ip>:<nodeport>`
 - `curl http://localhost:31740`

K8s basics – services

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop  
$ curl http://localhost:31740  
{"message": "Howzit!"}
```



Accessing our service

K8s basics – services



Declarative services

- It is preferred to write services so they can be easily replicated
- This is done in the same way as the deployments
 - With a YAML file
- We still use the same syntax, but different variables
- We will rewrite our service from before
 - Simply expose our single deployment

K8s basics – services

```
RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop/services
$ cat howzit-greeter-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: greeter-service ←
spec:
  selector:
    app: howzit-greeter ←
  ports:
    - protocol: TCP
      port: 80 ←
      targetPort: 80
  type: NodePort ←
```



Looking at our declaration

K8s basics – services

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop/services
$ ls -a
./ ../ howzit-greeter-service.yaml

RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop/services
$ kubectl apply -f howzit-greeter-service.yaml
service/greeter-service created

RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop/services
$ kubectl get service/greeter-service
```

| NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|-----------------|----------|----------------|-------------|--------------|-----|
| greeter-service | NodePort | 10.102.145.103 | <none> | 80:32638/TCP | 22s |

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop/services
$ curl http://localhost:32638
{"message":"Howzit!"}
```



Applying our service

K8s basics – services

```
RES+no-ro-nle@acc-nlennox MSYS ~/Documents/k8-workshop/services
$ kubectl get all ←
```

| NAME | READY | STATUS | RESTARTS | AGE |
|--|-------|---------|-------------|-----|
| pod/howzit-greeter-deployment-66c9bb7d84-2xjc7 | 1/1 | Running | 1 (72m ago) | 22h |
| pod/howzit-greeter-deployment-66c9bb7d84-59547 | 1/1 | Running | 1 (72m ago) | 22h |
| pod/howzit-greeter-deployment-66c9bb7d84-6v4mb | 1/1 | Running | 1 (72m ago) | 22h |

| NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|-------------------------|-----------|----------------|-------------|--------------|-------|
| service/greeter-service | NodePort | 10.102.145.103 | <none> | 80:32638/TCP | 4m17s |
| service/kubernetes | ClusterIP | 10.96.0.1 | <none> | 443/TCP | 24h |

| NAME | READY | UP-TO-DATE | AVAILABLE | AGE |
|---|-------|------------|-----------|-----|
| deployment.apps/howzit-greeter-deployment | 3/3 | 3 | 3 | 22h |

| NAME | DESIRED | CURRENT | READY | AGE |
|--|---------|---------|-------|-----|
| replicaset.apps/howzit-greeter-deployment-66c9bb7d84 | 3 | 3 | 3 | 22h |



Looking at the bigger picture

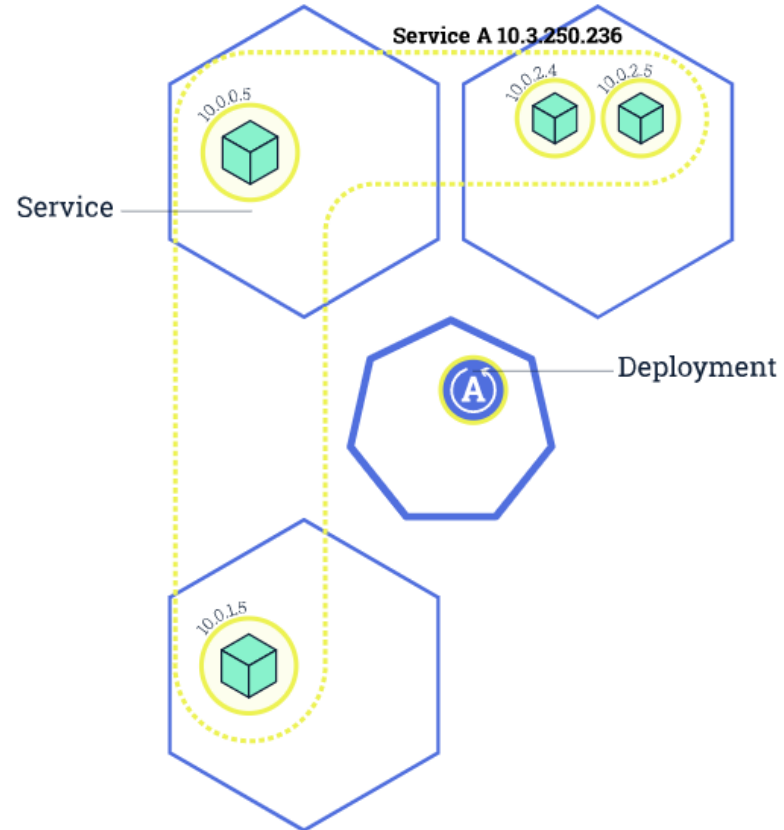
K8s basics – services



Rolling updates

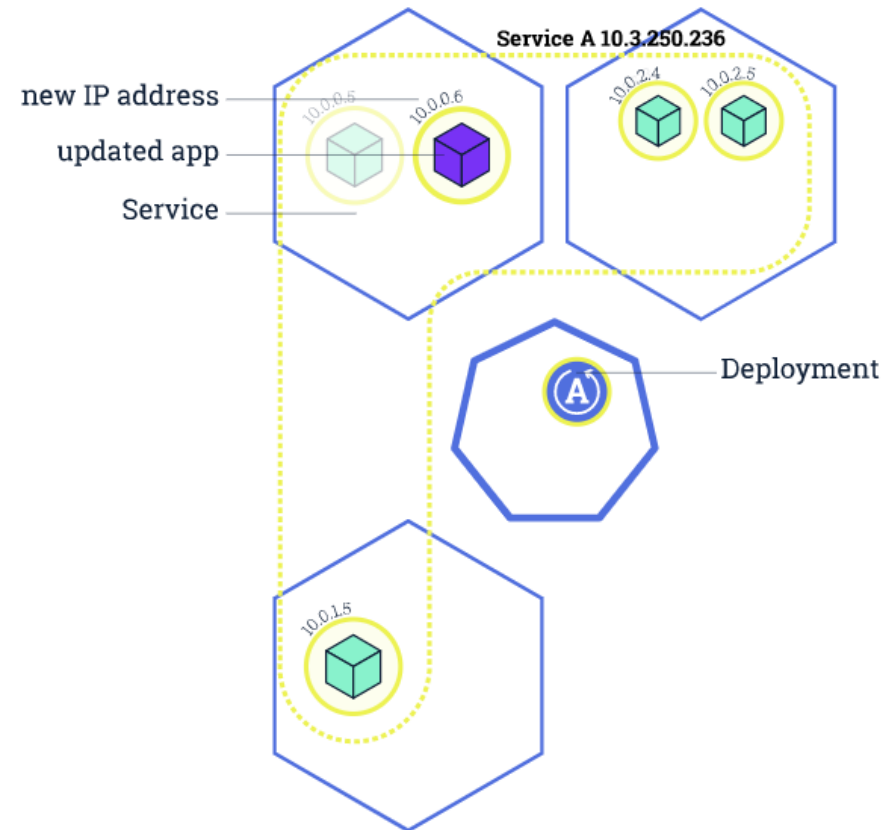
- No matter the resource (pod, deployment, or service), k8s will roll updates out to ensure zero downtime
- This can be for a new version (through CI/CD), moving to a new environment, or altering a resource
 - This also involves new IPs being allocated
- K8s updates individual containers in pods to ensure there is disruptions
 - It also checks to see if the new containers are healthy, if not, you can roll back
- Done by simply writing *kubectl apply -f*

K8s basics – services



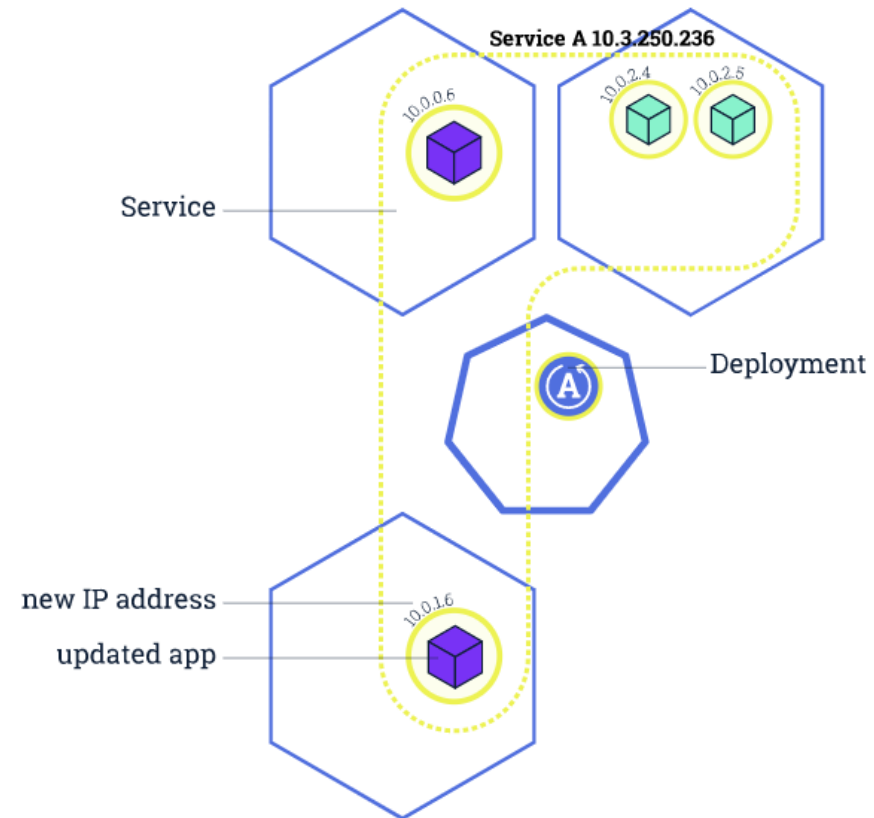
Rolling update 1/4

K8s basics – services



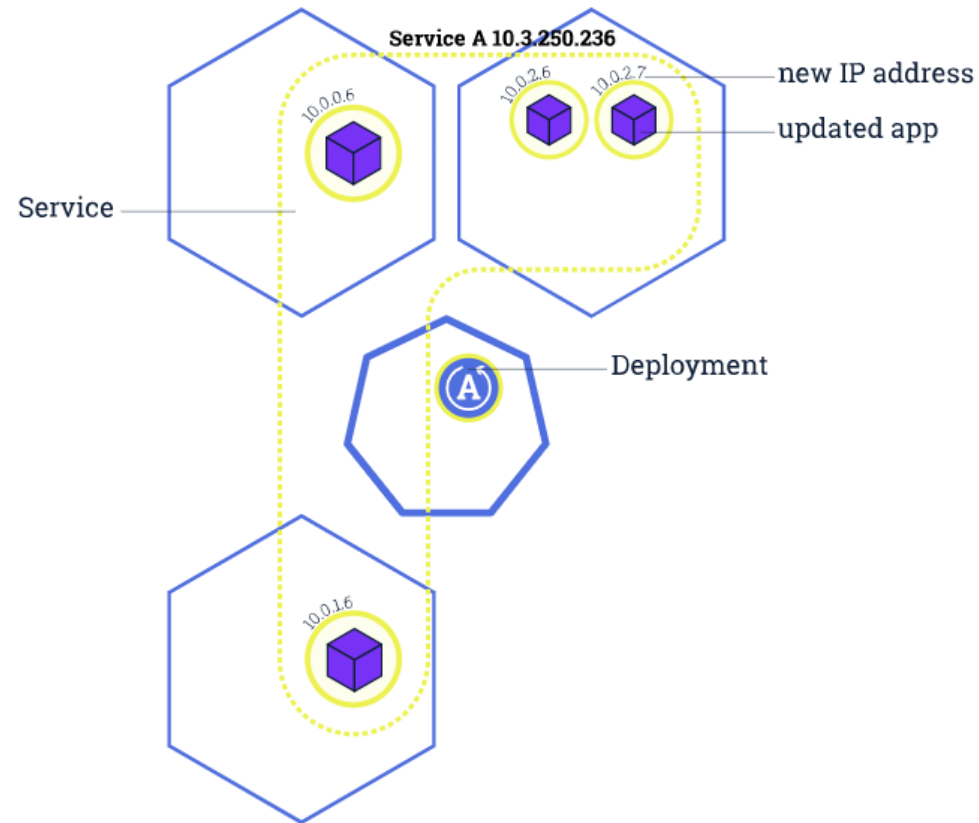
Rolling update 2/4

K8s basics – services



Rolling update 3/4

K8s basics – services



Rolling update 4/4 ([src](#))

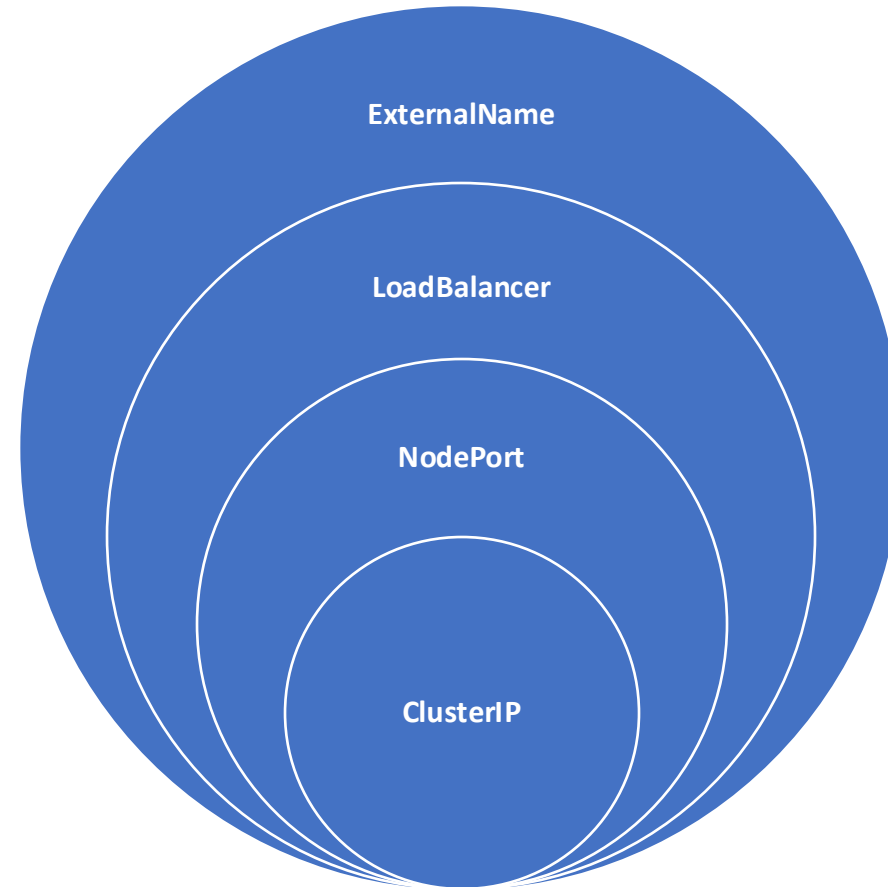
K8s basics – services



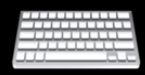
Visibility levels of service types

- When a service is NodePort it can be seen inside the node through statically exposing a port
- By default, ClusterIP doesn't let this happen
 - Unless kubectl proxy – we saw this to access our cluster
- Load balancers control outside traffic into your cluster
 - They can route internet traffic
 - Normally link up to cloud load balancers (Azure)
- DNS helps resolve names both internally and externally
 - Via ExternalName, or kube-dns internally

K8s basics – services



Different types of services



Try it out

1. Write a service manifest (yaml) to expose your previous deployment as a NodePort service
 - a) Can use this [link](#) as a starting template
2. Find its exposed node port with *kubectl get service*
3. Visit localhost:<nodeport> to view your running application

K8s basics – services



Taking a step back

- Now that we have exposed a service and accessed it, lets put it into perspective
- From a functional point of view (user interaction), what do you think is different than if we just did this all with docker (built and run)?
 - What about docker-compose?
 - How would replicas be handled?

K8s basics – services



Kubernetes learning curve (perspective)

- It is an insanely complicated tool, we just did the base basics to get a service running
- The strength it has is managing complex systems
 - Not our single stateless container
- We will look at some more complex features in this workshop
 - But cannot cover everything (even in a month course)
- K8s has a really high skill floor and ceiling
 - Making it a highly desired skill for developers

K8s basics – services



Service types resources

- The following are links to the k8s documentation in relation to service types
 - [ClusterIP](#) (default)
 - [NodePort](#)
 - [LoadBalancer](#)
 - [ExternalName](#)
- Note: most articles you will read (including most of these slides :D) use the official docs, so may as well go straight to the source.

K8s basics – dns



A method of discovery

- Simply put, a DNS turns IP addresses into searchable names
- There is a popular addon called **kube-dns** (Kubernetes enables it by default) that performs the role of a cluster aware DNS
- It watches the Kubernetes API for new Services and creates a set of DNS records for each one.
 - This allows you to refer to a service by its label and not IP
 - It solves the issues that arise from deployment controllers
 - Yes, it's the same as *docker-compose*
- Internally, we can say <http://greeting-service> to refer to our created service and not worry about its actual IP
 - Similar to how we used localhost through the exposed docker node to access it

K8s basics – dns



Inspecting kube-dns

- It is important to remember: *k8s is just a bunch of containers*
 - It is not a single conductor, its many conductors all playing their part
- Why don't we see this?
 - When we look at docker, we see our pods.
 - When we say *kubectl get all* we just see our resources
- *Its actually being hidden from you*
 - For a very good reason, it's a lot of bloat
 - We live in the default namespace, there are others

K8s basics – dns

```
RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop
```

```
$ kubectl get all --all-namespaces
```

| NAMESPACE | NAME | READY | STATUS | RESTARTS | AGE |
|-------------|--|-------|---------|---------------|-----|
| default | pod/howzit-greeter-deployment-66c9bb7d84-2xjc7 | 1/1 | Running | 1 (3h38m ago) | 24h |
| default | pod/howzit-greeter-deployment-66c9bb7d84-59547 | 1/1 | Running | 1 (3h38m ago) | 24h |
| default | pod/howzit-greeter-deployment-66c9bb7d84-6v4mb | 1/1 | Running | 1 (3h38m ago) | 24h |
| kube-system | pod/coredns-95db45d46-mc7wf | 1/1 | Running | 1 (3h38m ago) | 26h |
| kube-system | pod/coredns-95db45d46-s7cn1 | 1/1 | Running | 1 (3h38m ago) | 26h |
| kube-system | pod/etcd-docker-desktop | 1/1 | Running | 1 (3h38m ago) | 26h |
| kube-system | pod/kube-apiserver-docker-desktop | 1/1 | Running | 1 (3h38m ago) | 26h |
| kube-system | pod/kube-controller-manager-docker-desktop | 1/1 | Running | 1 (3h38m ago) | 26h |
| kube-system | pod/kube-proxy-j5wcl | 1/1 | Running | 1 (3h38m ago) | 26h |
| kube-system | pod/kube-scheduler-docker-desktop | 1/1 | Running | 1 (3h38m ago) | 26h |
| kube-system | pod/storage-provisioner | 1/1 | Running | 2 (3h37m ago) | 26h |
| kube-system | pod/vpnkit-controller | 1/1 | Running | 30 (12m ago) | 26h |

| NAMESPACE | NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|-------------|-------------------------|-----------|----------------|-------------|------------------------|------|
| default | service/greeter-service | NodePort | 10.102.145.103 | <none> | 80:32638/TCP | 149m |
| default | service/kubernetes | ClusterIP | 10.96.0.1 | <none> | 443/TCP | 26h |
| kube-system | service/kube-dns | ClusterIP | 10.96.0.10 | <none> | 53/UDP,53/TCP,9153/TCP | 26h |



A peak behind the curtain

K8s basics – dns

```
RES+no-ro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl describe service kube-dns --namespace=kube-system
Name: kube-dns
Namespace: kube-system
Labels: k8s-app=kube-dns
        kubernetes.io/cluster-service=true
        kubernetes.io/name=CoreDNS
Annotations: prometheus.io/port: 9153
              prometheus.io/scrape: true
Selector: k8s-app=kube-dns
Type: ClusterIP
IP Family Policy: SingleStack
IP Families: IPv4
IP: 10.96.0.10
IPs: 10.96.0.10
Port: dns 53/UDP
TargetPort: 53/UDP
Endpoints: 10.1.0.18:53,10.1.0.20:53
Port: dns-tcp 53/TCP
TargetPort: 53/TCP
Endpoints: 10.1.0.18:53,10.1.0.20:53
Port: metrics 9153/TCP
TargetPort: 9153/TCP
Endpoints: 10.1.0.18:9153,10.1.0.20:9153
Session Affinity: None
Events: <none>
```



A detailed look at kube-dns

K8s basics – dns



Checking visibility

- To see if our service is discoverable, we can drill into a pod and execute a command
 - Specifically, **nslookup**.
- *kubectl exec* lets you start a shell session to containers running in your Kubernetes cluster.
 - Use a -- to separate kubectl commands from arguments
 - [Link](#) to docs
- So, we want to start a shell on a container in a pod, then see if we can see our service through the internal network

K8s basics – dns

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl get pods
NAME                                         READY   STATUS    RESTARTS   AGE
howzit-greeter-deployment-66c9bb7d84-2xjc7 1/1     Running   1 (3h50m ago) 25h
howzit-greeter-deployment-66c9bb7d84-59547 1/1     Running   1 (3h50m ago) 25h
howzit-greeter-deployment-66c9bb7d84-6v4mb 1/1     Running   1 (3h50m ago) 25h

RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl exec howzit-greeter-deployment-66c9bb7d84-2xjc7 -- ls -a
.
..
Microsoft.OpenApi.dll
Swashbuckle.AspNetCore.Swagger.dll
Swashbuckle.AspNetCore.SwaggerGen.dll
Swashbuckle.AspNetCore.SwaggerUI.dll
appsettings.Development.json
appsettings.json
howzit-api.deps.json
howzit-api.dll
howzit-api.pdb
howzit-api.runtimeconfig.json
web.config
```



Inspecting the folder structure

K8s basics – dns



Checking on our services

- Our pod wont have **nslookup** installed
 - We could install it, but that's a bit silly as the pod will be destroyed
- Its easier to make a utility pod, specifically for DNS
 - You can do the same for *logging* or any other utility/addon
- We will follow the [Debugging DNS Resolution](#) docs to do this
- It's a simple pod that has nothing but some utility commands installed
 - It has no processes either

K8s basics – dns

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl apply -f https://k8s.io/examples/admin/dns/dnsutils.yaml
pod/dnsutils created

RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl get pod dnsutils
NAME          READY   STATUS    RESTARTS   AGE
dnsutils      1/1     Running   0           14s

RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl exec -i -t dnsutils -- nslookup kubernetes.default
Server:         10.96.0.10
Address:        10.96.0.10#53

Name:   kubernetes.default.svc.cluster.local
Address: 10.96.0.1
```



Running our dns util pod

K8s basics – dns

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl get svc
NAME                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
greeter-service     NodePort      10.102.145.103 <none>         80:32638/TCP     3h11m
kubernetes           ClusterIP     10.96.0.1     <none>         443/TCP          27h

RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl exec -i -t dnsutils -- nslookup greeter-service.default ←
Server:             10.96.0.10
Address:            10.96.0.10#53

Name:   greeter-service.default.svc.cluster.local ←
Address: 10.102.145.103

RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl exec -i -t dnsutils -- nslookup greeter-service ←
Server:             10.96.0.10
Address:            10.96.0.10#53

Name:   greeter-service.default.svc.cluster.local
Address: 10.102.145.103
```



Seeing if our service is discoverable

Any questions?

Maybe a break?

K8s basics – ingress



Providing access from outside

- Exposing individual services at random ports is not a good idea
 - Ports can change
 - We are breaking the “cohesive whole”
 - Each service needs its own certificate for TLS
- It's better to have a dedicated load balancer that routes traffic from outside to services
 - This way our cluster is hidden, and exposed through a single service

K8s basics – ingress



What is ingress?

- Dictionary definition – “the action or fact of going in or entering”
- Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster.
- Traffic routing is controlled by rules defined on the Ingress resource.
- These ingress rules (just called ingresses) are fulfilled with an ingress controller which acts as a load balancer

K8s basics – ingress



Ingress controllers

- Controllers abstract away complexities of k8s traffic
- For ingress resources to work, a controller needs to be running
- By default, k8s does not run one. However, it's a very common addon and there are many templated options
- The most commonly used is **nginx**
 - There is one maintained and supported by k8s
- [Link](#) to quick start ([link](#) to general nginx ingress docs)

K8s basics – ingress

```
RES+noro-nle@acc-nlennox MSYS ~  
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.5.1/deploy/static/provider/cloud/deploy.yaml  
namespace/ingress-nginx created  
serviceaccount/ingress-nginx created  
serviceaccount/ingress-nginx-admission created  
role.rbac.authorization.k8s.io/ingress-nginx created  
role.rbac.authorization.k8s.io/ingress-nginx-admission created  
clusterrole.rbac.authorization.k8s.io/ingress-nginx created  
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created  
rolebinding.rbac.authorization.k8s.io/ingress-nginx created  
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created  
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created  
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created  
configmap/ingress-nginx-controller created  
service/ingress-nginx-controller created  
service/ingress-nginx-controller-admission created  
deployment.apps/ingress-nginx-controller created  
job.batch/ingress-nginx-admission-create created  
job.batch/ingress-nginx-admission-patch created  
ingressclass.networking.k8s.io/nginx created  
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created
```



After following quick start 1/3

K8s basics – ingress

| NAMESPACE | NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|---------------|--|--------------|----------------|-------------|----------------------------|------|
| default | service/kubernetes | ClusterIP | 10.96.0.1 | <none> | 443/TCP | 2d5h |
| ingress-nginx | service/ingress-nginx-controller | LoadBalancer | 10.103.199.218 | localhost | 80:32688/TCP,443:30433/TCP | 79s |
| ingress-nginx | service/ingress-nginx-controller-admission | ClusterIP | 10.107.127.44 | <none> | 443/TCP | 79s |
| kube-system | service/kube-dns | ClusterIP | 10.96.0.10 | <none> | 53/UDP,53/TCP,9153/TCP | 2d5h |

| NAMESPACE | NAME | DESIRED | CURRENT | READY | UP-TO-DATE | AVAILABLE | NODE SELECTOR | AGE |
|-------------|---------------------------|---------|---------|-------|------------|-----------|------------------------|------|
| kube-system | daemonset.apps/kube-proxy | 1 | 1 | 1 | 1 | 1 | kubernetes.io/os=linux | 2d5h |

| NAMESPACE | NAME | READY | UP-TO-DATE | AVAILABLE | AGE |
|---------------|--|-------|------------|-----------|------|
| ingress-nginx | deployment.apps/ingress-nginx-controller | 1/1 | 1 | 1 | 79s |
| kube-system | deployment.apps/coredns | 2/2 | 2 | 2 | 2d5h |

| NAMESPACE | NAME | DESIRED | CURRENT | READY | AGE |
|---------------|---|---------|---------|-------|------|
| ingress-nginx | replicaset.apps/ingress-nginx-controller-8574b6d7c9 | 1 | 1 | 1 | 79s |
| kube-system | replicaset.apps/coredns-95db45d46 | 2 | 2 | 2 | 2d5h |

| NAMESPACE | NAME | COMPLETIONS | DURATION | AGE |
|---------------|--|-------------|----------|-----|
| ingress-nginx | job.batch/ingress-nginx-admission-create | 1/1 | 9s | 79s |
| ingress-nginx | job.batch/ingress-nginx-admission-patch | 1/1 | 10s | 79s |



After following quick start 2/3

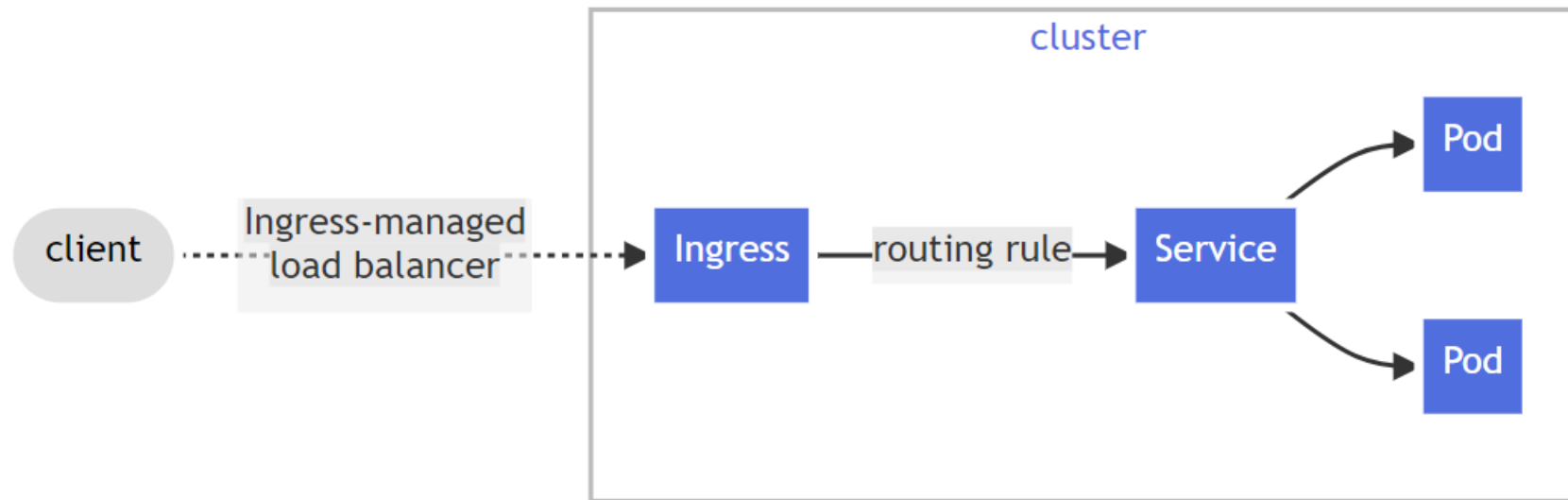
K8s basics – ingress

```
RES+noronle@acc-nlennox MSYS ~  
$ curl localhost  
<html>  
<head><title>404 Not Found</title></head>  
<body>  
<center><h1>404 Not Found</h1></center>  
<hr><center>nginx</center>  
</body>  
</html>
```



After following quick start 3/3

K8s basics – ingress



Basic illustration ([src](#))

K8s basics – ingress



Now what?

- We added the production ready ingress controller
 - If you look at the manifest its quite scary
 - Also all the extra roles and configuration it added
- We can start adding ingress resources to create maps
- Lets start by mapping */greeting* to our greeting service
- First lets change our current service and deployments to be a little clearer and completely hidden from outside traffic
 - We can do both deployment and service in one file (---)

K8s basics – ingress

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ cat howzit-greeter.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: greeter
  labels:
    app: greeter
spec:
  replicas: 2
  selector:
    matchLabels:
      app: greeter
  template:
    metadata:
      labels:
        app: greeter
    spec:
      containers:
        - name: greeter
          image: nicholaslennox/howzit-greeter
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
---
```



Our cleaned up resource 1/2

K8s basics – ingress

```
---
apiVersion: v1
kind: Service
metadata:
  name: greeter
spec:
  selector:
    app: greeter
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      name: http
  type: ClusterIP
```

Will select deployment.app/greeter

We gave the port a name, so we can refer to it as http

Our service is fully internal



Our cleaned up resource 2/2

K8s basics – ingress

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl apply -f howzit-greeter.yaml
deployment.apps/greeter created
service/greeter created

RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl get all
```

| NAME | READY | STATUS | RESTARTS | AGE |
|-----------------------------|-------|---------|----------|-----|
| pod/greeter-d4644f8c8-4qx77 | 1/1 | Running | 0 | 5s |
| pod/greeter-d4644f8c8-f4xp7 | 1/1 | Running | 0 | 5s |

| NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|--------------------|-----------|---------------|-------------|---------|------|
| service/greeter | ClusterIP | 10.110.155.37 | <none> | 80/TCP | 5s |
| service/kubernetes | ClusterIP | 10.96.0.1 | <none> | 443/TCP | 2d5h |

| NAME | READY | UP-TO-DATE | AVAILABLE | AGE |
|-------------------------|-------|------------|-----------|-----|
| deployment.apps/greeter | 2/2 | 2 | 2 | 5s |

| NAME | DESIRED | CURRENT | READY | AGE |
|-----------------------------------|---------|---------|-------|-----|
| replicaset.apps/greeter-d4644f8c8 | 2 | 2 | 2 | 5s |



Applying our resource

K8s basics – ingress



Adding an ingress resource

- Once again, it's a template. So we can just follow the [k8s docs](#)

K8s basics – ingress

```
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: greeting
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /greeting
        pathType: Prefix
        backend:
          service:
            name: greeter
            port:
              name: http
```

Lets you create a different
URL representation if you
want

Ingress class?

Referring to our
service port by name



Our ingress resource

K8s basics – ingress

```
RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop  
$ kubectl apply -f howzit-greeter.yaml  
deployment.apps/greeter unchanged  
service/greeter unchanged  
ingress.networking.k8s.io/greeting created
```

```
$ kubectl get ingress  
NAME          CLASS    HOSTS    ADDRESS    PORTS    AGE  
greeting      nginx    *        localhost  80       66m
```

```
$ curl localhost/greeting  
{"message": "Howzit!"}
```



Applying our resource

K8s basics – ingress



ingressClass?

- Clusters can have multiple ingress controllers
 - Yours, and one for the cloud provider (Azure, AWS, and so on)
- You need to say which controller is going to implement your ingress resource
 - We just want our default ingress-nginx (comes with it)
 - [Creating](#) your own default ingress-nginx class

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl get ingressClass
NAME      CONTROLLER      PARAMETERS      AGE
nginx     k8s.io/ingress-nginx <none>          91m
```

K8s basics – ingress

```
RES+noro-nle@acc-nlennox MSYS ~/Documents/k8-workshop
$ kubectl describe ingressClass/nginx
Name:          nginx
Labels:        app.kubernetes.io/component=controller
               app.kubernetes.io/instance=ingress-nginx
               app.kubernetes.io/name=ingress-nginx
               app.kubernetes.io/part-of=ingress-nginx
               app.kubernetes.io/version=1.5.1
Annotations:   <none>
Controller:    k8s.io/ingress-nginx
Events:        <none>
```



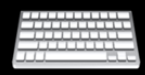
Describing our ingressClass

K8s basics – ingress



What else?

- It can help manage egress (outgoing) traffic
- It also is commonly used as a TLS terminator
 - Meaning, HTTPS to the ingress controller, HTTP from it to the services
 - This is fine because all the services are not exposed and any validation can happen in the ingress controller
 - We just need to get a certification for a hostname and then we can route TLS traffic and quite efficiently secure our cluster



Try it out

1. Make sure you have your service running as ClusterIP (NodePort wont register with ingress)
 - a) You can just alter the yaml and re-apply
2. Install the ingress controller from nginx (link in slides)
 - a) Make sure its running by checking `kubectl get all --namespace=ingress-nginx`
3. Create an ingress resource to map a url to your service
 - a) Don't forget `ingressClassName`
4. Check it out by visiting `localhost/your-url` 😊

Azure Kubernetes Service

Using Azure to help deploy to production

Azure Kubernetes Service



What is it?

- A managed Kubernetes cluster on Azure
 - It handles critical tasks, like health monitoring and maintenance.
 - You don't pay for the cluster itself, only the VMs running nodes
- When you deploy an AKS cluster, you specify the number and size of the nodes
 - AKS deploys and configures the Kubernetes control plane and nodes
- It provides Authentication and RBAC with Azure AD
- It provides logging and monitoring as well

Azure Kubernetes Service

[Home](#) > [Create a resource](#) > [Marketplace](#) > [Azure Kubernetes Service \(AKS\)](#) >

Create Kubernetes cluster ...

Basics Node pools Access Networking Integrations Advanced Tags Review + create

Azure Kubernetes Service (AKS) manages your hosted Kubernetes environment, making it quick and easy to deploy and manage containerized applications without container orchestration expertise. It also eliminates the burden of ongoing operations and maintenance by provisioning, upgrading, and scaling resources on demand, without taking your applications offline.

[Learn more about Azure Kubernetes Service](#)

Project details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Pay-As-You-Go



Resource group * ⓘ

noroff-k8s-workshop

[Create new](#)

Cluster details

Cluster preset configuration

Dev/Test (\$)

To quickly customize your Kubernetes cluster, choose one of the preset configurations above. You can modify these configurations at any time.

[Learn more and compare presets](#)

Kubernetes cluster name * ⓘ

noroff-accelerate

Region * ⓘ

(Europe) West Europe

Makes master node publicly accessible



Creating our cluster

Azure Kubernetes Service

API server availability ⓘ

☐ 99.99%
Optimize for availability. 99.95% is available when at least one availability zone is selected.

☒ 99.5%
Optimize for cost.

99.5% API server availability is recommended for dev/test configuration.

Automatic upgrade ⓘ

Enabled with patch (recommended) ▼

Primary node pool

The number and size of nodes in the primary node pool in your cluster. For production workloads, at least 3 nodes are recommended for resiliency. For development or test workloads, only one node is required. If you would like to add additional node pools or to see additional configuration options for this node pool, go to the 'Node pools' tab above. You will be able to add additional node pools after creating your cluster. [Learn more about node pools in Azure Kubernetes Service](#)

Node size * ⓘ

Standard B2s
2 vcpus, 4 GiB memory

Standard B4ms is recommended for dev/test configuration.

[Change size](#)

Scale method * ⓘ

☐ Manual

☒ Autoscale

Autoscaling is recommended for dev/test configuration.

Node count range * ⓘ

1 5

[Review + create](#) [< Previous](#) [Next : Node pools >](#)

Make sure it's
the cheapest.



Trying to keep costs down



Azure Kubernetes Service

Basics **Node pools** Access Networking Integrations Advanced Tags Review + create

Node pools

In addition to the required primary node pool configured on the Basics tab, you can also add optional node pools to handle a variety of workloads [Learn more about node pools](#)

+

 Add node pool

🗑

 Delete

| Name | Mode | OS type | Node count | Node size |
|------------------------------------|--------|---------|------------|--------------|
| <input type="checkbox"/> agentpool | System | Linux | 1-5 | Standard_B2s |

<

>

Enable virtual nodesVirtual nodes allow burstable scaling backed by serverless Azure Container Instances. [Learn more about virtual nodes](#)Enable virtual nodes

📘

☐**Node pool OS disk encryption**By default, all disks in AKS are encrypted at rest with Microsoft-managed keys. For additional control over encryption, you can supply your own keys using a disk encryption set backed by an Azure Key Vault. The disk encryption set will be used to encrypt the OS disks for all node pools in the cluster. [Learn more](#)Encryption type

(Default) Encryption at-rest with a platform-managed key

Review + create

< Previous

Next : Access >



Node pools group nodes

Azure Kubernetes Service

Basics Node pools **Access** Networking Integrations Advanced Tags Review + create

Resource identity ⓘ
System-assigned managed identity
By default, Azure uses a managed identity. To use a service principal, use the CLI.
[Learn more](#) ↗

Choose between local accounts or Azure AD for authentication and Azure RBAC or Kubernetes RBAC for your authorization needs.

Authentication and Authorization ⓘ
Local accounts with Kubernetes RBAC

Local accounts with Kubernetes RBAC
Use built-in Kubernetes role-based access control for authorization checks on the cluster.

Azure AD authentication with Kubernetes RBAC
Use Azure AD for authentication and Kubernetes native RBAC for authorization.

Azure AD authentication with Azure RBAC
Use Azure role assignments for authorization checks on the cluster.

[Review + create](#) < Previous Next : Networking >



We will just use local accounts for now

Azure Kubernetes Service

Basics Node pools Access **Networking** Integrations Advanced Tags Review + create

You can change networking settings for your cluster, including enabling HTTP application routing and configuring your network using either the 'Kubenet' or 'Azure CNI' options:

- The **kubenet** networking plug-in creates a new VNet for your cluster using default values.
- The **Azure CNI** networking plug-in allows clusters to use a new or existing VNet with customizable addresses. Application pods are connected directly to the VNet, which allows for native integration with VNet features.

[Learn more about networking in Azure Kubernetes Service](#)

Network configuration ⓘ ☒ Kubenet ☐ Azure CNI

DNS name prefix * ⓘ ✓

Traffic routing

Load balancer ⓘ Standard

Enable HTTP application routing ⓘ ☒ ← Sets up an ingress controller for us (only for dev)

Security

Enable private cluster ⓘ ☐

Set authorized IP ranges ⓘ ☐

[Review + create](#) [< Previous](#) [Next : Integrations >](#)



We will just enable HTTP routing

Azure Kubernetes Service

Azure Monitor

In addition to the CPU and memory metrics included in AKS by default, you can enable Container Insights for more comprehensive data on the overall performance and health of your cluster. Billing is based on data ingestion and retention settings.

[Learn more about container performance and health monitoring](#)

[Learn more about pricing](#)

Container monitoring

☒ Enabled ☐ Disabled

Log Analytics workspace ⓘ

(New) DefaultWorkspace-7cd83741-a350-4064-beee-0b2cc0e389b4-WEU

[Create new](#)

Use managed identity (preview) ⓘ

☐

Azure Policy

Apply at-scale enforcements and safeguards for AKS clusters in a centralized, consistent manner through Azure Policy.

[Learn more about Azure Policy for AKS](#)

Azure Policy

☐ Enabled ☒ Disabled

[Review + create](#)

[< Previous](#)

[Next : Advanced >](#)



We want to include monitoring

Azure Kubernetes Service

Basics Node pools Access Networking Integrations Advanced Tags Review + create

Enable secret store CSI driver ⓘ ☐

Infrastructure resource group ⓘ ✓
[Edit](#)

[Review + create](#) [< Previous](#) [Next : Tags >](#)



We wont include the Key Vault



Azure Kubernetes Service

noroff-accelerate

Kubernetes service

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Microsoft Defender for Cloud

Kubernetes resources

Namespaces

Workloads

Services and ingresses

Storage

Configuration

Settings

Node pools

Cluster configuration

Networking

Open Service Mesh

Essentials JSON View

Resource group : [noroff-k8s-workshop](#)

Status : Succeeded (Running)

Location : West Europe

Subscription : [Pay-As-You-Go](#)

Subscription ID : 7cd83741-a350-4064-beee-0b2cc0e389b4

Tags ([edit](#)) : [Click here to add tags](#)

Get started

Properties

Monitoring

Capabilities (3)

Recommendations

Tutorials

Kubernetes services

Encryption type

Virtual node pools

Encryption at-rest with a platform-managed key

Not enabled

Node pools

Node pools

Kubernetes versions

Node sizes

1 node pool

1.23.12

Standard_B2s

Configuration

Kubernetes version

Auto Upgrade Type

1.23.12

Patch

Networking

API server address

Network type (plugin)

Pod CIDR

Service CIDR

DNS service IP

Docker bridge CIDR

Network Policy

Load balancer

HTTP application routing

Private cluster

noroff-accelerate-dns-35956f46.hcp.westeurope.azmk8s.io

Kubenet

10.244.0.0/16

10.0.0.0/16

10.0.0.10

172.17.0.1/16

None

Standard

Enabled

Not enabled





Finally deployed.




Azure Kubernetes Service


Kubernetes resources

 Namespaces


 Workloads


 Services and ingresses


 Storage

 Configuration

Settings


 Node pools

 Cluster configuration

 Networking

Filter by namespace name

Enter the full namespace name

 Add label filter

| <input type="checkbox"/> | Name | Status | Age |
|--------------------------|---------------------------------|----------|-----------|
| <input type="checkbox"/> | kube-node-lease | ✓ Active | 5 minutes |
| <input type="checkbox"/> | kube-public | ✓ Active | 5 minutes |
| <input type="checkbox"/> | kube-system | ✓ Active | 5 minutes |
| <input type="checkbox"/> | default | ✓ Active | 5 minutes |



Our namespaces



Azure Kubernetes Service

Kubernetes resources

Namespaces

Workloads

Services and ingresses

Storage

Configuration

Settings

Node pools

Cluster configuration

Networking

Open Service Mesh

ServicesIngresses

Filter by service name

Filter by namespace

Enter the full service name

All namespaces

| <input type="checkbox"/> | Name | Namespace | Status | Type | Cluster IP | External IP | Ports | Age ↓ |
|--------------------------|--|-------------|--------|--------------|-------------|------------------------------|-------------------|-----------|
| <input type="checkbox"/> | kubernetes | default | ✓ Ok | ClusterIP | 10.0.0.1 | | 443/TCP | 6 minutes |
| <input type="checkbox"/> | kube-dns | kube-system | ✓ Ok | ClusterIP | 10.0.0.10 | | 53/UDP,53/TCP | 6 minutes |
| <input type="checkbox"/> | addon-http-application-routing-nginx-ingress | kube-system | ✓ Ok | LoadBalancer | 10.0.28.74 | 20.73.192.39 | 80:31506/TCP,4... | 6 minutes |
| <input type="checkbox"/> | metrics-server | kube-system | ✓ Ok | ClusterIP | 10.0.12.148 | | 443/TCP | 6 minutes |



Our services

Azure Kubernetes Service

Home > microsoft.aks-20221204163300 | Overview > noroff-accelerate

noroff-accelerate | Alerts

Kubernetes service

Policies

Properties

Locks

Monitoring

Insights

Alerts

Metrics

Diagnostic settings

Advisor recommendations

Logs

Workbooks

Automation

Tasks (preview)

Export template

Support + troubleshooting

Resource health

New Support Request

Create Alert rules Action groups Alert policies

Set up alert rules

Get notified on important alert rules or creating your own

Enable recommended

Enable recommended alert rules

Alert me if

☒ CPU Usage Percentage is greater than %

☒ Memory Working Set Percentage is greater than %

[More alerting options](#)

Notify me by

☒ Email

☐ Email Azure Resource Manager Role

☐ Azure mobile app notification

☐ Use an existing action group

Estimated monthly total: \$0.20

Enable

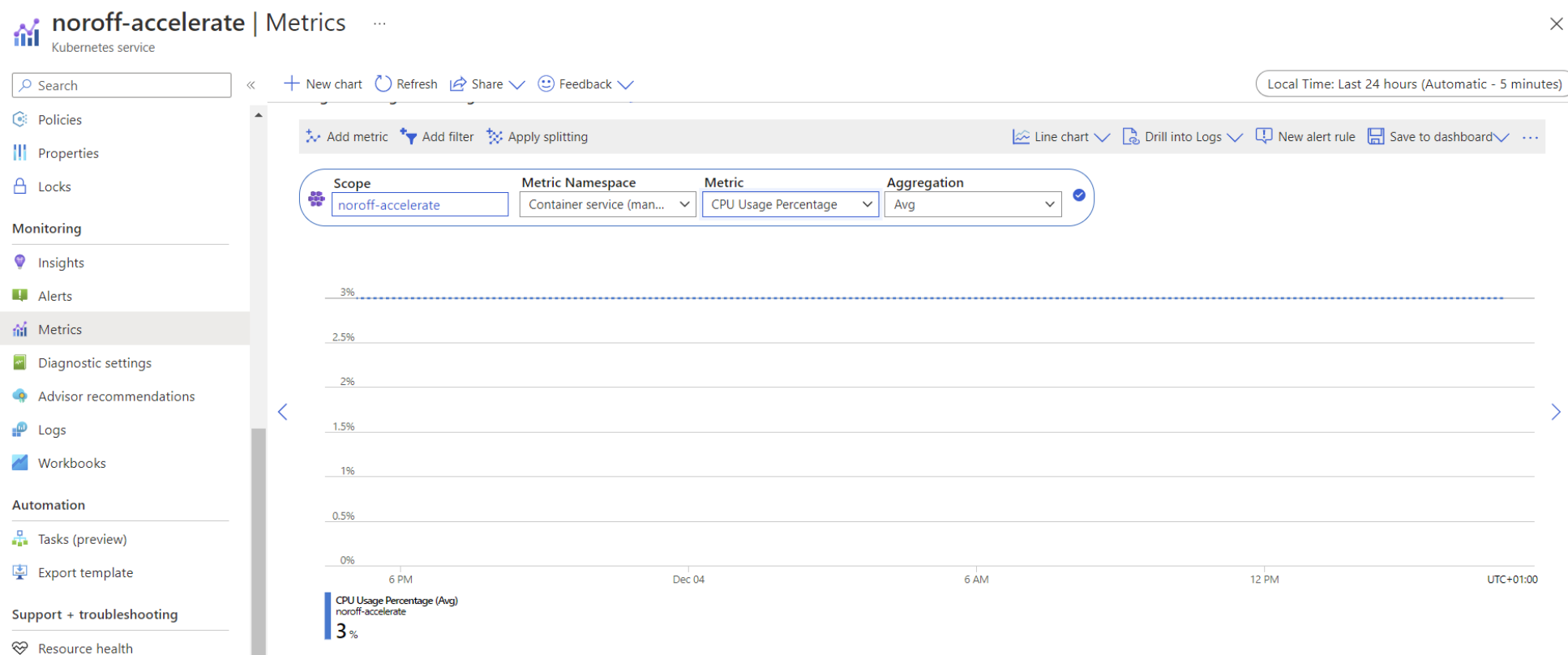
Cancel



Looking at alerts



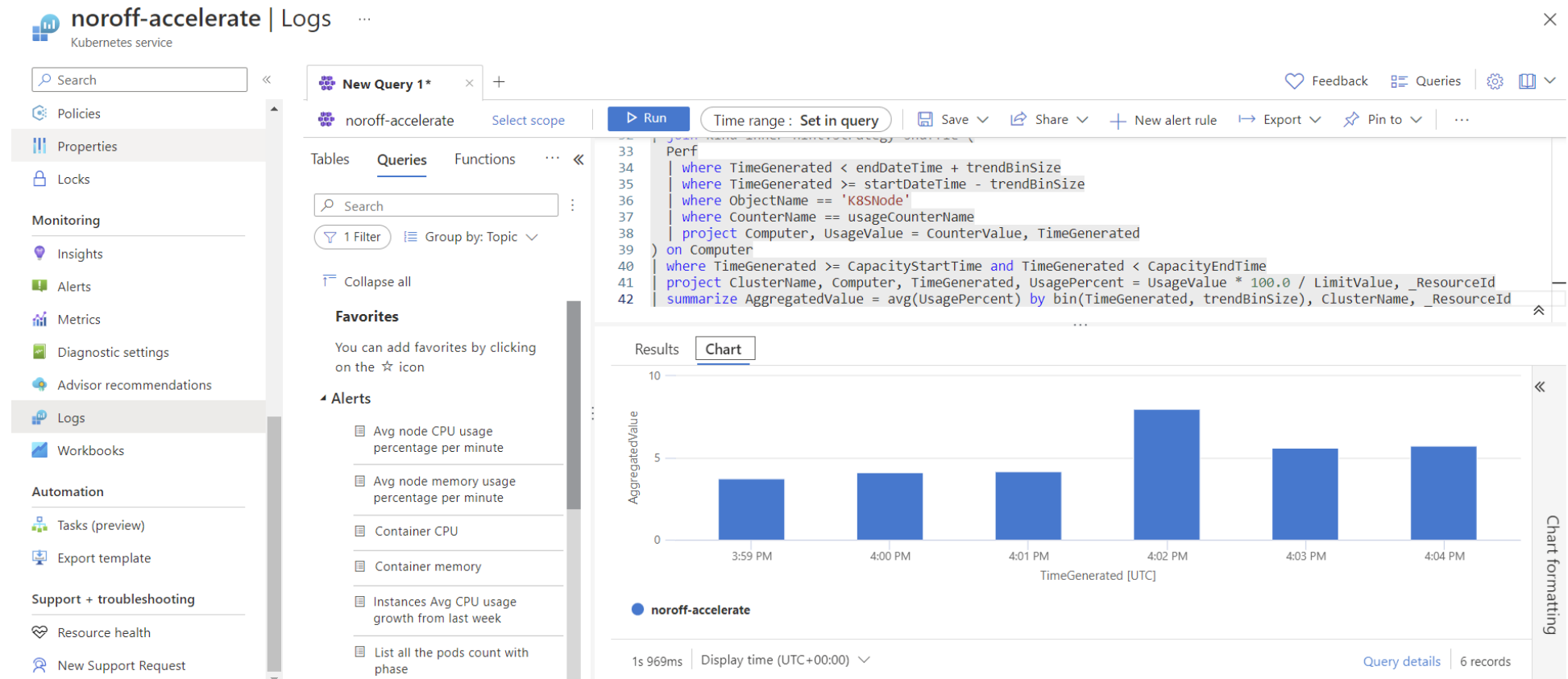
Azure Kubernetes Service



Looking at metrics



Azure Kubernetes Service



Looking at logs of CPU usage

Azure Kubernetes Service



Okay now what?

- We can interact with *kubect!* through the azure cli
 - Or directly copy paste yaml contents
- Lets use *kubect!* to replicate some of the things we did locally
- Then use the “add” tool to copy paste deployments and services



Azure Kubernetes Service

The screenshot displays the Azure portal interface for the 'noroff-accelerate' Kubernetes service. The top navigation bar shows the breadcrumb 'Home > microsoft.aks-20221204163300 | Overview >'. The service name 'noroff-accelerate' is prominently displayed with a star icon and the text 'Kubernetes service'. Below this, a search bar and a row of action buttons (Create, Connect, Start, Stop, Delete, Refresh, Give feedback) are visible. A blue arrow points to the 'Connect' button. The left sidebar contains navigation links for Policies, Properties, Locks, Monitoring, Insights, Alerts, Metrics, and Diagnostic settings. The main content area shows the 'Essentials' tab with details for the 'noroff-k8s-workshop' resource group, including its status (Succeeded (Running)), location (West Europe), subscription (Pay-As-You-Go), and subscription ID. Below this, there are tabs for Get started, Properties (selected), Monitoring, Capabilities (3), Recommendations (1), and Tutorials. On the right, a 'Connect to noroff-accelerate' panel provides instructions on how to connect to the cluster using command-line tools. It lists two steps: '1. Open Cloud Shell or the Azure CLI' and '2. Run the following commands'. A blue arrow points to the first step. Below the instructions, there are two code blocks: one for 'az account set' and another for 'az aks get-credentials'. Below the main content area, a terminal window titled 'Bash' shows the output of the 'az' commands. The terminal text includes 'Requesting a Cloud Shell. Succeeded.', 'Connecting terminal...', 'Welcome to Azure Cloud Shell', and the execution of the 'az account set' and 'az aks get-credentials' commands. A blue arrow points to the terminal output.

Home > microsoft.aks-20221204163300 | Overview >

noroff-accelerate

Kubernetes service

Search

Create Connect Start Stop Delete Refresh Give feedback

Policies

Properties

Locks

Monitoring

Insights

Alerts

Metrics

Diagnostic settings

Essentials

Resource group : [noroff-k8s-workshop](#)

Status : Succeeded (Running)

Location : West Europe

Subscription : [Pay-As-You-Go](#)

Subscription ID : 7cd83741-a350-4064-beee-0b2cc0e389b4

Tags ([edit](#)) : [Click here to add tags](#)

Get started **Properties** Monitoring Capabilities (3) Recommendations (1) Tutorials

Connect to noroff-accelerate

Connect to your cluster using command line tooling to interact directly with cluster using kubectl, the command line tool for Kubernetes. Kubectl is available within the Azure Cloud Shell by default and can also be installed locally. [Learn more](#)

1. Open Cloud Shell or the Azure CLI

2. Run the following commands

```
az account set --subscription 7cd83741-a350-4064-beee-0b2cc0e389b4
```

```
az aks get-credentials --resource-group noroff-k8s-workshop --name noroff-accelerate
```

Sample commands

Once you have run the command above to connect to the cluster, you can run any kubectl commands. Here are a few examples of useful commands you can try.

```
# List all deployments in all namespaces
kubectl get deployments --all-namespaces=true
```

Bash

```
Requesting a Cloud Shell. Succeeded.
Connecting terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI
Type "help" to learn about Cloud Shell

nicholas [ ~ ]$ az account set --subscription 7cd83741-a350-4064-beee-0b2cc0e389b4
nicholas [ ~ ]$ az aks get-credentials --resource-group noroff-k8s-workshop --name noroff-accelerate
Merged "noroff-accelerate" as current context in /home/nicholas/.kube/config
```



Opening cloud shell

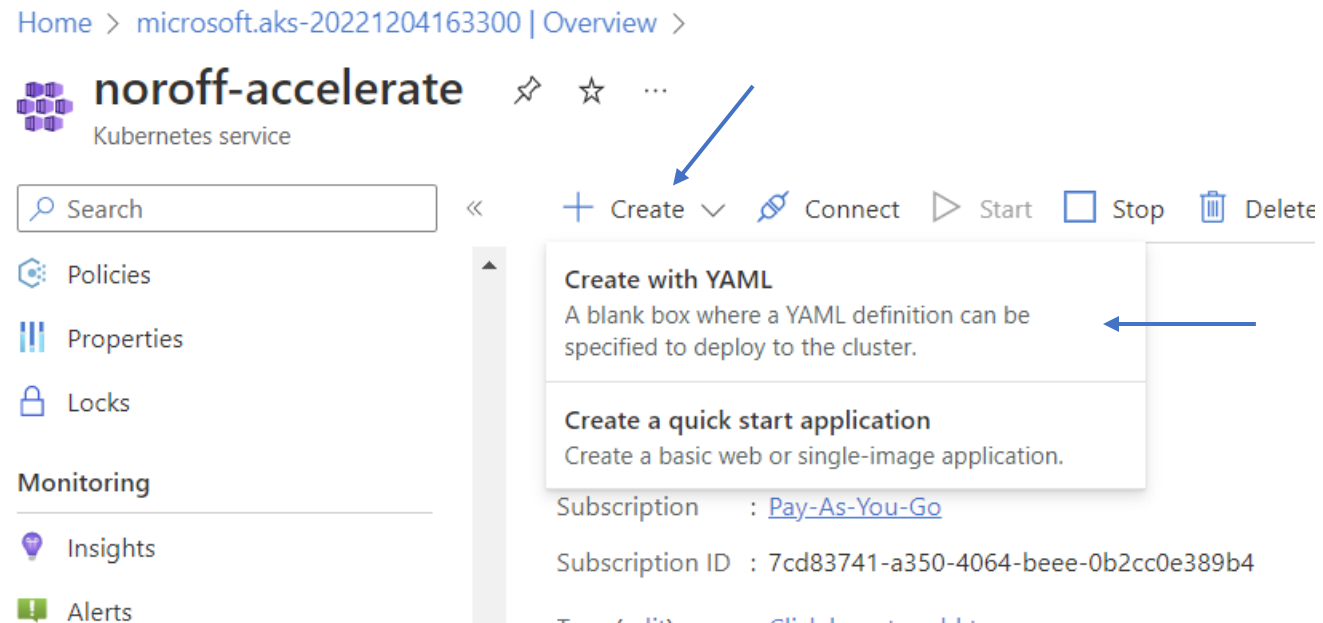
Azure Kubernetes Service

```
Bash  ▾ | 🔌 ? ⚙️ 📄 📄 {} 🔍  
nicholas [ ~ ]$ kubectl get all  
NAME                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE  
service/kubernetes  ClusterIP     10.0.0.1      <none>         443/TCP    4h27m  
nicholas [ ~ ]$
```



Our current cluster status

Azure Kubernetes Service



Opening the create screen

Azure Kubernetes Service

[Home](#) > [microsoft.aks-20221204163300 | Overview](#) > [noroff-accelerate](#) >

Add with YAML ...

Not sure where to start? [Deploy a quickstart application to get up and running.](#)

YAML JSON

1



Copy paste the manifests in

Azure Kubernetes Service

```
Bash
nicholas [ ~ ]$ kubectl get all
NAME                                     READY   STATUS    RESTARTS   AGE
pod/howzit-greeter-deployment-6b8dddd497-2n6d4   1/1     Running   0          112s
pod/howzit-greeter-deployment-6b8dddd497-9xbh2   1/1     Running   0          112s
pod/howzit-greeter-deployment-6b8dddd497-l4sm5   1/1     Running   0          112s

NAME                                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/greeter-service            NodePort      10.0.60.133  <none>        80:30263/TCP     112s
service/kubernetes                  ClusterIP     10.0.0.1     <none>        443/TCP          4h30m

NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/howzit-greeter-deployment  3/3     3            3          112s

NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/howzit-greeter-deployment-6b8dddd497  3         3         3       112s
nicholas [ ~ ]$
```



Our state after adding manifests

Azure Kubernetes Service



Accessing our service

- Right now, we cant access anything
 - Previously we could with a NodePort service, because we were on the host machine (node)
- Azure locks all that off, and the only way to access the services is through ingress
- The “enable http application routing” gives us an easy ingress controller to use
- We just need to write an ingress rule for it

Azure Kubernetes Service



Using HTTP application routing

- NB: This is only for dev/testing. Production should have a configured ingress controller.
 - [Link](#) to relevant doc
- To follow what we are doing in the next few slides with more detail, see this article on [http application routing](#)

Azure Kubernetes Service



What does HTTP application routing do

- Makes it easy to access applications that are deployed to your AKS cluster.
- When the solution's enabled, it configures an Ingress controller in your AKS cluster.
- As applications are deployed, the solution also creates publicly accessible DNS names for application endpoints.
- When the add-on is enabled, it creates a DNS Zone in your subscription.
- Note: Doing this yourself can get really complicated and expensive (links in a slide further down)

Azure Kubernetes Service



What it adds

- The add-on deploys two components: a **Kubernetes Ingress controller** and an **External-DNS controller**.
- Ingress controller: The Ingress controller is exposed to the internet by using a Kubernetes service of type LoadBalancer.
- External-DNS controller: Watches for Kubernetes Ingress resources and creates DNS A records in the cluster-specific DNS zone.

Azure Kubernetes Service






Getting the host name

- We didn't have to consider hostname locally (because it was localhost)
 - However, we now need to say which hosts our ingress resource is for
- We can get our assigned host name through the Azure Cloud shell or Azure cli
- `az aks show \`
 - `-g $RESOURCE_GROUP \`
 - `-n $CLUSTER_NAME \`
 - `-o tsv \`
 - `--query`
`addonProfiles.httpApplicationRouting.config.HTTPApplicationRoutingZoneName`

Azure Kubernetes Service

Home >

 **noroff-accelerate**   ...

Kubernetes service

```
Bash | [Icons: Power, Help, Settings, Copy, Paste, Find, Close]

Requesting a Cloud Shell. Succeeded.
Connecting terminal...

nicholas [ ~ ]$ az account set --subscription 7cd83741-a350-4064-beee-0b2cc0e389b4
nicholas [ ~ ]$ az aks get-credentials --resource-group noroff-k8s-workshop --name noroff-accelerate
Merged "noroff-accelerate" as current context in /home/nicholas/.kube/config
nicholas [ ~ ]$ az aks show \
  -g noroff-k8s-workshop \
  -n noroff-accelerate \
  -o tsv \
  --query addonProfiles.httpApplicationRouting.config.HTTPApplicationRoutingZoneName
fc72cc12b581448b8f8a.westeurope.aksapp.io
nicholas [ ~ ]$
```



Getting our domain

Azure Kubernetes Service

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: greeter
  annotations:
    kubernetes.io/ingress.class: addon-http-application-routing
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: greeter.fc72cc12b581448b8f8a.westeurope.aksapp.io
    http:
      paths:
      - path: /greeting
        pathType: Prefix
        backend:
          service:
            name: greeter
            port:
              number: 80
```

Cant use
ingressClassName
(addons fault)

Format of host:
<ingress-
name>.<domain-name>



Writing our ingress rule

Azure Kubernetes Service

| <input type="checkbox"/> | Name | Namespace | Class | Hosts | Address | Ports |
|--------------------------|---------------------------------|-----------|----------------------------|--|--|-------|
| <input type="checkbox"/> | howzit-greeting | default | addon-http-application-... | howzit-greeting.fc72cc12b5 | 20.73.192.39 🔗 | 80 |
| <input type="checkbox"/> | greeter | default | addon-http-application-... | greeter.fc72cc12b581448b8 | 20.73.192.39 🔗 | 80 |

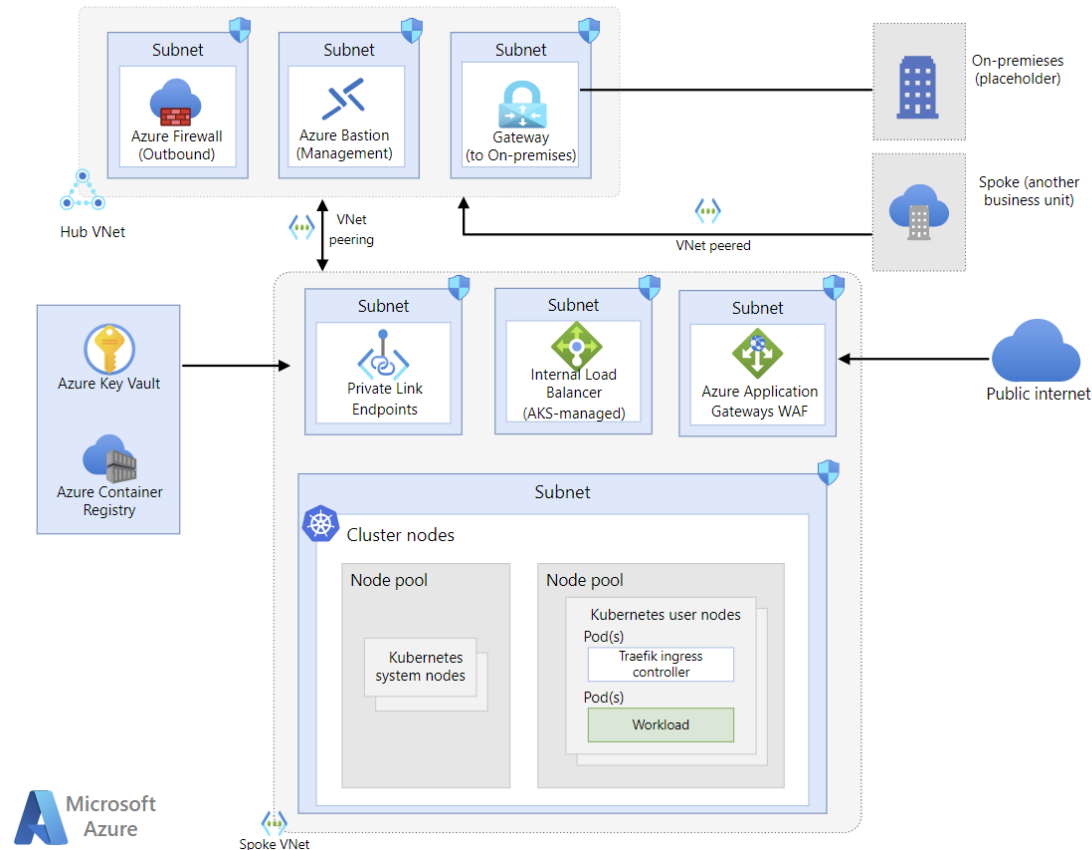
```
RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop
$ curl http://greeter.fc72cc12b581448b8f8a.westeurope.aksapp.io/
{"message": "Howzit!"}
```



Our ingress (it may take ~10 mins)

Any questions?

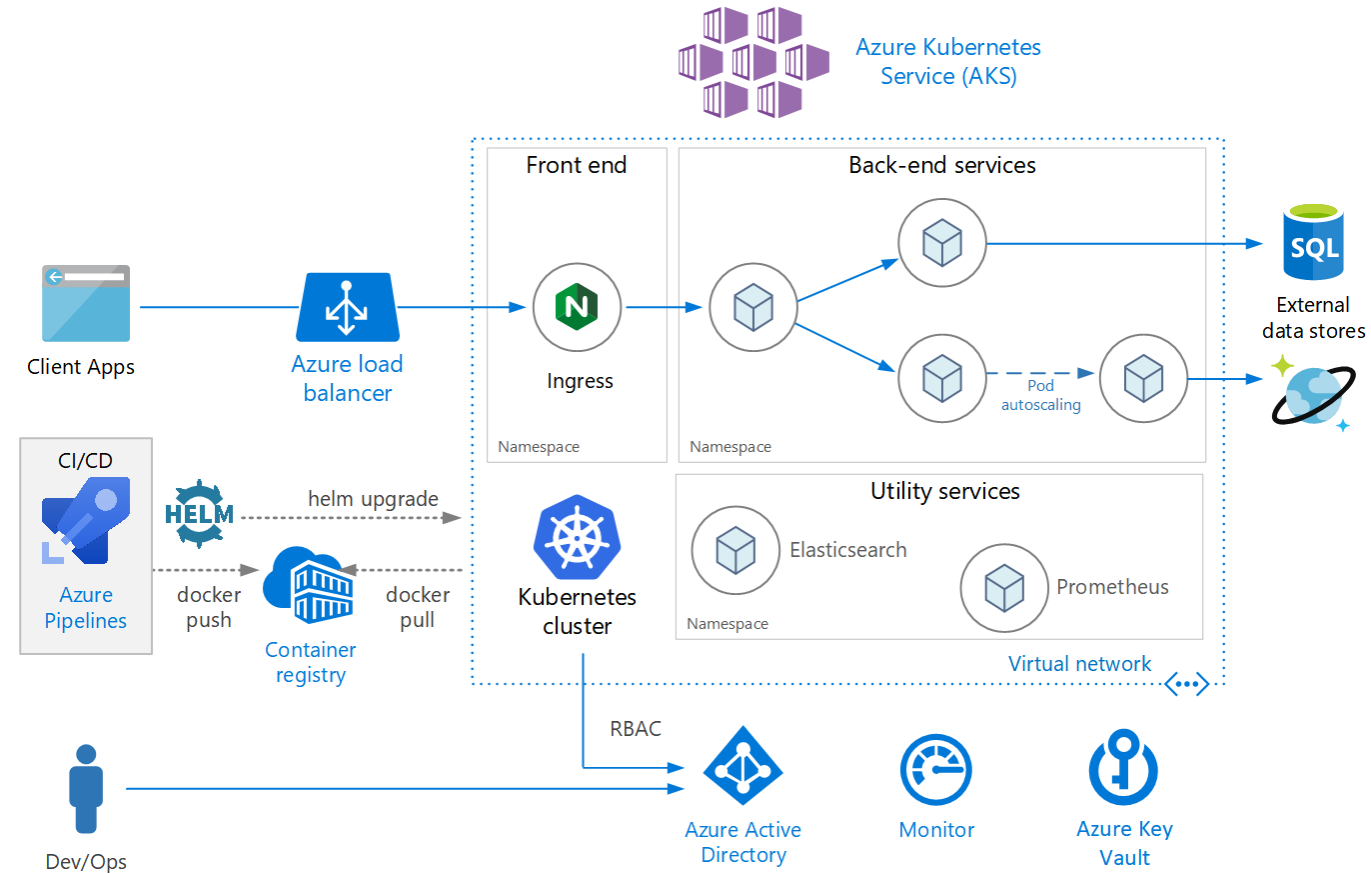
Azure Kubernetes Service



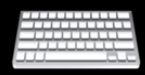
Looking at baseline AKS architecture



Azure Kubernetes Service



Looking at AKS + microservices



Try it out

1. Create a resource for AKS
 - a) Remember to enable HTTP application routing (if you forget, you can change it afterwards 😊)
2. Add your deployment and service from the previous try it out
 - a) You can check them in the Workloads and Services blades
3. Add an ingress resource
 - a) Follow the HTTP application routing guide linked, or the slides
 - b) You can see it in Kubernetes Resources>Services and ingresses>ingresses
 - c) Remember you cant use ingressClassName in spec
 - d) You also need to have / as a path (this requires some investigating)

Securing k8s

Securing traffic and access to a k8s cluster

Securing k8s



How can we protect k8s?

- There are a few ways in which security can be brought into k8s
- The most common is access control
 - We saw this with Azure AD in our AKS
 - It restricts which users can access your cluster and control panel
- There is also secret management (encrypted configuration)
 - Used to store tokens, certs, and other sensitive information
- Then there is secure traffic over TLS (https://...)

Securing k8s



What we will do

- We will just do a local self signed TLS certificate for our ingress controller
- Doing it on azure is very pricy (app gateways and so on) and quite complicated without *helm* and *azure cli*
- Will follow [this guide](#)
 - There is [this guide](#) from nginx
 - And supported with [this doc](#) in k8s
- We need to have a host name
 - Docker exposes localhost through k8s as: **kubernetes.docker.internal**
 - We can use this hostname for the cert

Securing k8s



Get a cert

- We will create our own one using openssl
- This required us to be a certificate authority and create a public private key pair
 - as well as a certification with our domain
- Part of the previously linked guide, I wont show the steps
 - Link to the [relevant steps](#)
- NOTE: This certificate will not be trusted by our browser
 - We aren't on the trusted CA list, there are links to solve this after the demonstration

Securing k8s



Adding to k8s secrets

- Recall, k8s stores its configuration internally.
 - This includes all manifests, env variables, secrets, and more.
- We can use the *kubectl secret* command to tap into the secrets
- We will do this to add our fake certificate to the secret store
- We follow the template:
 - `kubectl create secret tls ${CERT_NAME} --key ${KEY_FILE} --cert ${CERT_FILE}`

Securing k8s

```
RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop  
$ kubectl create secret tls howzit-tls --namespace default --key server.key --cert server.crt|
```

```
RES+noronle@acc-nlennox MSYS ~/Documents/k8-workshop  
$ kubectl get secret  
NAME          TYPE          DATA   AGE  
howzit-tls    kubernetes.io/tls  2       8m11s
```



Adding our fake certs

Securing k8s



Now what?

- We need to use the secret in our ingress resource
- We do this to specify the TLS requirements
 - We specify which hosts this applies to
 - And which secret to use

Securing k8s

New spec to define tls

We need to define host, otherwise it wont pass the cert

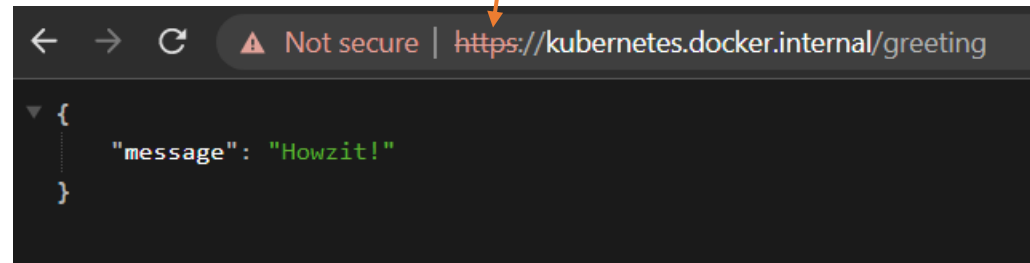
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: greeting
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - kubernetes.docker.internal
      secretName: howzit-tls
  rules:
    - host: kubernetes.docker.internal
      http:
        paths:
          - path: /greeting
            pathType: Prefix
            backend:
              service:
                name: greeter
                port:
                  name: http
```



Our updated ingress

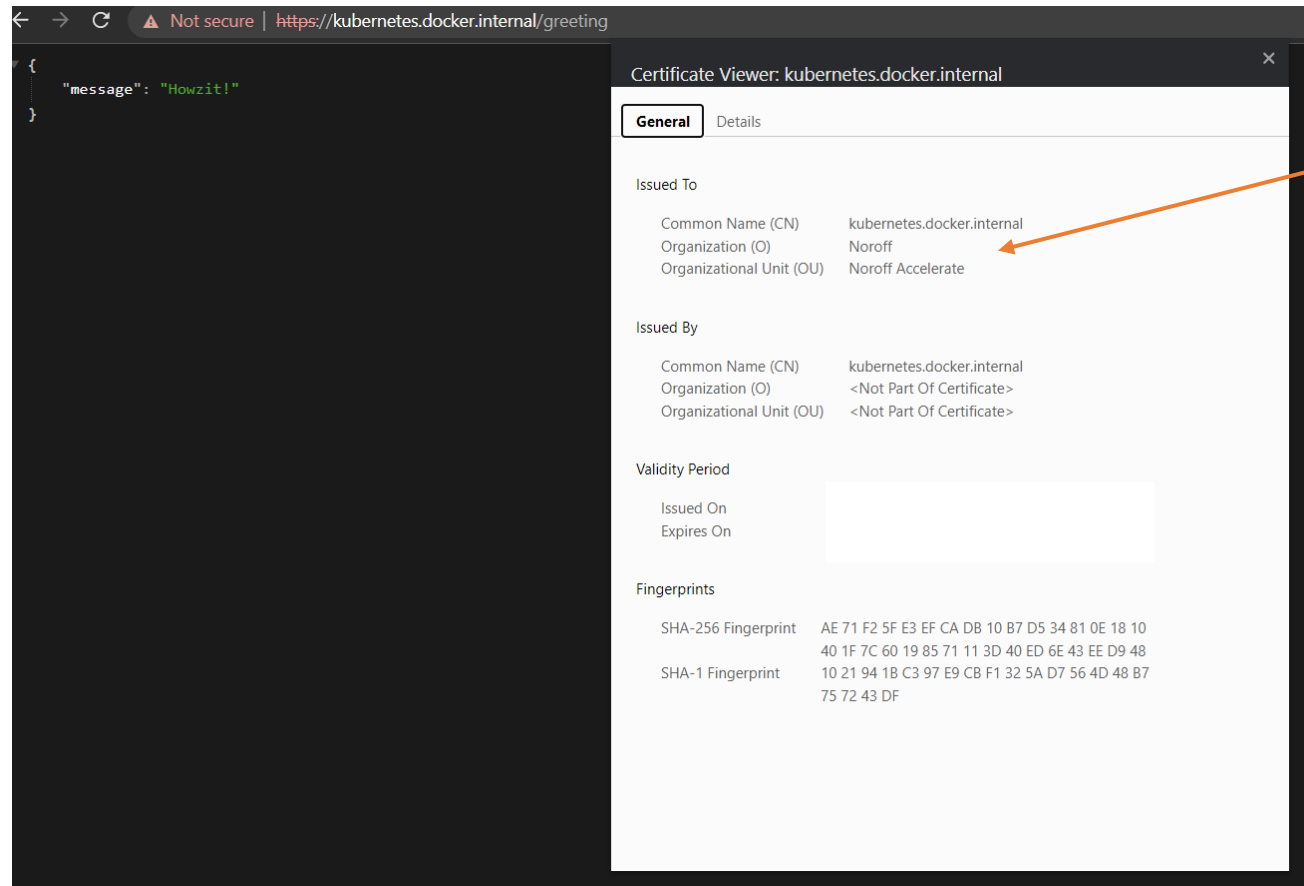
Securing k8s

Saying insecure, lets inspect



Going to https

Securing k8s



Inspecting the cert

Securing k8s



How to get a trusted cert?

- You need to get it from a proper cert provider (not self signed), normally <https://letsencrypt.org/>
- This is a bit too long for us to do in this workshop
- Nginx page on [cert managers](#)
 - [Link](#) to full example (requires helm)
- The process of adding secrets and using them as TLS spec is the same.

Any questions?

I know the securing part feels a bit meh, we would need another workshop to actually cover certs, authentication and TLS properly

Securing k8s



Relevant security documentation

- [Controlling access in k8s](#)
- [RBAC on k8s](#)
- [Secrets on k8s](#)
- [TLS in k8s](#)
- [HTTPS/TLS on ingress-nginx](#)
- [Expose an AKS service over HTTP or HTTPS using Application Gateway](#)
 - [Enabling ingress controller](#)
- [Use TLS with an ingress controller on Azure Kubernetes Service \(AKS\)](#)

In Conclusion



Key takeaways

- Kubernetes is a very complicated set of tools. It has a very high skill floor and the highest skill ceiling. You can work with it for years and still be confused most of the time.
- The fundamental resources are:
 - Deployments manage pods
 - Services abstract deployments and provide a way of interacting with dns
 - Ingress helps expose your services to the outside with rules
- AKS throws a lot at you, but its not that difficult to get some thing basic working – with some limitations.
- K8s is secured through access control, secrets, and TLS termination in the ingress controllers

In Conclusion



Looking Ahead

- We touched the surface of Kubernetes and distributed computing.
- It is an insanely complex field, but an interesting one at that.
 - It requires networking, and security knowledge (more, but these are the largest)
- If you want to continue with k8s, my biggest suggestion is to learn more about security (cryptography, certs, etc.) and networking. It will help when learning the more complex parts.
- Helm is a great resource to learn, it's basically a mandatory tool with k8s in production.

Additional Resources



Hungry for more?

- The next few slides provide links to useful resources we also use from time to time:
 - Reference guides
 - A must have for any developer
 - Video demonstrations
 - Quality videos explaining selected topics or demonstrating skills
 - Articles, sample code and hands-on tutorials
 - Interesting articles
 - Additional code examples
 - Step-by-step guides for extra practice

Additional Resources



Articles, Sample Code & Tutorials

- Kubernetes [docs](#)
- Ingress Nginx [docs](#)
- Azure AKS [docs](#)
- [Kubernetes Ingress 101](#) video by nginx
- Helm [docs](#)