**Run applications in Azure Kubernetes Service (AKS)**

## Deployment Configuration using Yaml

### Benefits of using Yaml

Using YAML for K8s definitions gives you a number of advantages, including:

* **Convenience:** You’ll no longer have to add all of your parameters to the command line
* **Maintenance:** YAML files can be added to source control, so you can track changes
* **Flexibility:** You’ll be able to create much more complex structures using YAML than you can on the command line

## Creating a deployment

1. Create a Manifest file – services.yaml(you can use whatever name you want)
2. Have the following value (example)

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name: ibmservice

spec:

replicas: 1

template:

metadata:

labels:

app: ibmservice

spec:

containers:

- name: ibmservice

image: saketcontainerregistry.azurecr.io/ibmservice:v1

ports:

- containerPort: 8900

name: ibmservice

1. Apply this yaml file to the k8 cluster using the below command
   * Kubectl apply -f services.yaml
   * The above command will create a deployment and a pod under default namespace
   * To see the deployments – Kubectl get deployments
   * To see the namespaces – Kubectl get namespaces
   * To see the pods – Kubectl get pods
2. But the above is not exposed as a service. Which means even though this pod is running no one can access it from outside the cluster (gives you a security abstraction,as in a microservice environment you would not expose all your pods,instead expose just the gateway).
3. Before we create a service and expose the pod to the world let us understand the Yaml structure first and then the deployment configurations.

**5.1) Yaml Structure**

5.1.1) There are two basic concepts of a Yaml(Yet-Another-Markup-Language) -List and Map.

5.1.2) There can be list of maps or maps of list or just list or just map. If you have got this one down, you are good to write your own yaml.

5.1.3) Yaml maps as Java maps provide a key value mapping, just in this case you do not create a map specifically as you do in Java(I am a Java guy !!).

**apiVersion: apps/v1beta1** -- This is an example of key value pair mapping in Yaml file provided by Yaml Maps.

**5**.1.4) You can also have a more controlled structure where a key of the map maps to another map

Ex - spec:

replicas: 1

(REMEMBER THE INDENTION,IT IS VERY IMPORTANT)

Spec key refers to another map which has a key value pair. So this is a concept of **map of maps.**

The YAML processor knows how all of these pieces relate to each other because we’ve indented the lines

5.1.5) Example of Yaml List would be

containers:

- name: ibmservice

Containers is a list and it can have many values as depicted above.To add a new key value ,just move to the next line(maintaining indentation) and create another key value pair using “-“. The above example is an example of List of maps.

**5.2) Deployment configurations – Taking piece by piece**

**5.2.1) Apiversion -** Whenever a Kubernetes object is created an apiversion field is required. Kubernets objects can be a deployment or a service etc. Apiversion value tells what version of the Kubernets API is used for creating the Object. If you want to know about Kubernetes apis follow - <https://kubernetes.io/docs/reference/using-api/api-overview/>

**APIVERSION CHEAT SHEET**

|  |
| --- |
| Kind ApiVersion |
| |  |  | | --- | --- | | CertificateSigningRequest | certificates.k8s.io/v1beta1 | | ClusterRoleBinding | rbac.authorization.k8s.io/v1 | | ClusterRole | rbac.authorization.k8s.io/v1 | | ComponentStatus | v1 | | ConfigMap | v1 | | ControllerRevision | apps/v1 | | CronJob | batch/v1beta1 | | DaemonSet | extensions/v1beta1 | | Deployment | extensions/v1beta1 | | Endpoints | v1 | | Event | v1 | | HorizontalPodAutoscaler | autoscaling/v1 | | Ingress | extensions/v1beta1 | | Job | batch/v1 | | LimitRange | v1 | | Namespace | v1 | | NetworkPolicy | extensions/v1beta1 | | Node | v1 | | PersistentVolumeClaim | v1 | | PersistentVolume | v1 | | PodDisruptionBudget | policy/v1beta1 | | Pod | v1 | | PodSecurityPolicy | extensions/v1beta1 | | PodTemplate | v1 | | ReplicaSet | extensions/v1beta1 | | ReplicationController | v1 | | ResourceQuota | v1 | | RoleBinding | rbac.authorization.k8s.io/v1 | | Role | rbac.authorization.k8s.io/v1 | | Secret | v1 | | ServiceAccount | v1 | | Service | v1 | | StatefulSet | apps/v1 | |

5.2.2) Kind – what Kind of object you want to create -Ex- Deployment

5.2.3) metadata.name – By what name should the deployment happen- which means if you say kubectl get deployment – the value of metadata.name is the name of the deployment.

5.2.5)Spec List contains all the information that go into creating the pod

Spec.replicas = How many replicas to be created

* + So we know we need replicas (1,2) whatever value you set to it, but replicas of what?
  + The template within Spec defines the definition of the Object to be replicated

spec.template.metadata.lables.app = Labels are intended to be used to specify identifying attributes of objects that are meaningful and relevant to users, but do not directly imply semantics to the core system. Labels can be attached to objects at creation time and subsequently added and modified at any time. Each object can have a set of key/value labels defined. Labels allow for efficient queries and watches and are ideal for use in UIs and CLIs

spec.template.spec.containers = name of the pod

spec.template.spec.image = image to be pulled for the pod

spec.template.spec.ports.containerPort = The container port

Understanding the container port

**What is the problem in hosting on Docker?**

* + Docker containers can talk among themselves as long as they are on the same machine or node. In order for a Docker container to speak to another container hosted on a different machine , there must be ports allocated to the containers on their specific machine, so when a docker container a wants to speak to docker container b which is on a different machine, a port ex-80 must be allocated on the machine b is hosted and then any request at that port on that machine is forwarded/proxied to container b.
  + This is really struggling, because every container then needs to have a specific port attached to it and developers must be extremely careful in choosing the ports

**How Kubernets Solves it?**

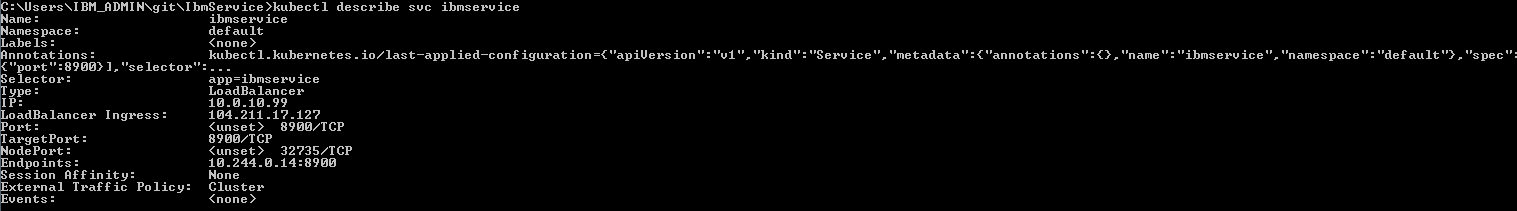
* + Kubernetes assumes that Pods communicate to other pods regardless of the host the pods land on.
  + Every pod is provided with its own cluster-private-ip address, so that pods can communicate to one another without mapping container ports to host ports.
  + All the containers within a pod can reach each other’s port on localhost.
  + All pods in a cluster(a cluster can have multiple nodes, each node can have multiple pods) can see each other without NAT.
  + spec.template.spec.ports.containerPort value makes a pod accessible from any node in the cluster.



See the Ip value, this makes it accessible across the cluster.

**Why should I create a service along with deployment?**

* + What if the deployment dies, the pod within it dies immediately,kubernetes does bring it up but the IP value that you see above changes, This is the problem service solves.
  + Kubernetes Service is just an abstraction for logical set of pods which have the same functionality.
  + When a service is created, each service is assigned a unique ip address called CLUSTER-IP.
  + This Ip will remain constant irrespective of the deployment going down and pods going down and coming up resulting in change of the ip of pods, until the service is deleted.
  + Pods can be configured to talk to service instead of doing inter pod communication.
  + When a service is created,a cluster-ip is assigned to it and the spec.ports.port is the port of the clusterip
  + No need to specify the target port, because target port gets allocated when the container comes up and by default is set
  + Pods are exposed through endpoints , the service loadbalances the request to one of the endpoints (if there are multiple pods running with the same spec.selector.app value.
  + The endpoints value is evaluated continuously, which means if a pod dies its endpoint(or the ip of the pod) is removed from the endpoints object. If a new pod comes up with the same selector app value then the IP/endpoint of the pod is updated to endpoints object of the service
  + Service is able to loadbalance the requests to pods
  + The same service can then be exposed to the external world by just setting the spec.type as LoadBalancer. And the Loadbalance ingress value plus the specs.port.port value together can be used to reach the service from external world
  + url would be LoadBalancerIngress:spect.port.port



Thinking of Discovery with Kubernetes and spring cloud together?

* + Try https://dzone.com/articles/quick-guide-to-microservices-with-kubernetes-sprin