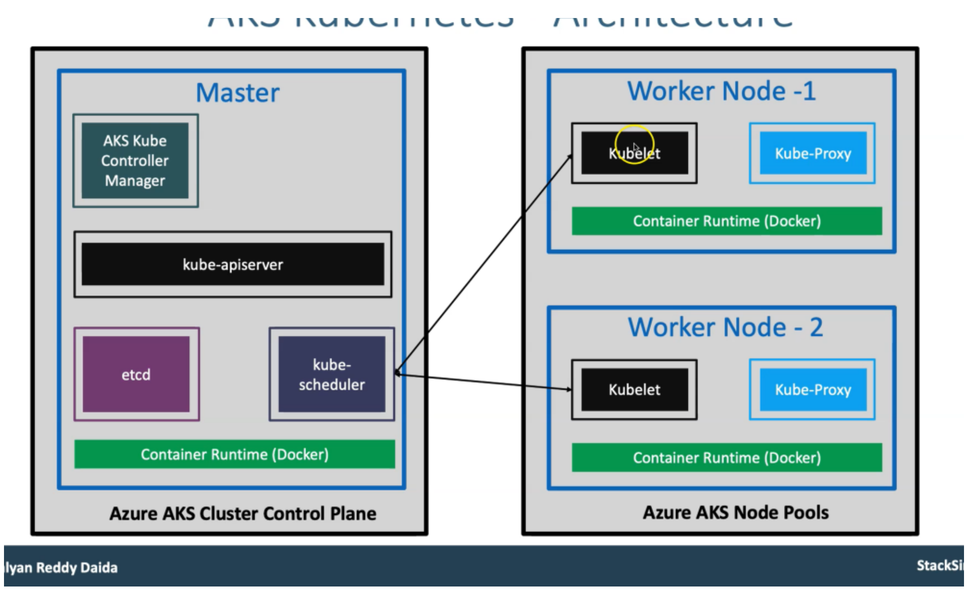
Kubernetes Demo

A Kubelet runs on every node



- Create a k8s cluster (not arc) on portal
    - Make sure you are on Project-3-Training Subscription
    - 06012021Batch_Resource group (or name one)
    - K8s cluster name - k8s06012021Demo
    - South central region
    - Zones 1,2,3
    - Version 1.20.7 (or default?)
    - Size – DS2_v2 (not smallest, but recommended)
    - Manual scale method
    - Node count = 1
- Next – Node Pools
    - Virtual machine scale sets are required (default).
- Authentication
    - System assigned managed identity
    - Role-based access control – enabled
    - Encryption Type - (Default) Encryption at-rest with a platform-managed key.
    - Use all other defaults.
- Networking
    - Network configuration - Azure CNI –
    - Use defaults for:
        - Virtual network, cluster subnet, k8s service address range, k8s DNS service IP address, Docker Bridge address, DNS name prefix.
    - load balancer - standard

    - NO - Enable HTTP application routing
    - NO - private cluster
    - NO - set authorized IP ranges
    - Network policy – Azure
- Integrations
    - Don't create container registry. - NONE
    - Container monitoring – Enabled
    - Log Analytics Workspace – default
    - Azure policy- Disabled (this is a for now thing... the Udemy demo doesn't have this option)
- Tags
    - None
- Review and create
    - Wait a bit..... (take a break bc this may take >10 mins)
- Create
    - These last 2 may take a bit of time......
- When created, 'Go To Resource'
    - You will need the **resource group** and **name** to log in and begin
- After taking these steps, you are automatically given a loadbalancer service. DO NOT ERASE THIS SERVICE.. Or you will have to recreate it to expose any pods.

Using Azure CLI

- PREP = Go online OR use 'az aks install-cli' to Install azure cli
- PREP = https://docs.microsoft.com/en-us/cli/azure/?view=azure-cli-latest
- PREP = To log in, you need all your credentials. Get these in the JSON view of portal or after using the below 'az login' command which logs you in through the browser.
- PREP = 'az login' in gitbash to log in. Inspect the return after logging in.
- 'az aks get-credentials --resource-group 06012021Batch_ResourceGroup --name k8s06012021Demo' to merge your resource group with the current context.
    - You may need to change the subscription you are on.
    - Use 'az account list --output table'
    - Then 'az account set --subscription <subscriptionGUIDId>'
- 'kubectl get nodes' to see your nodes
- 'Kubectl get nodes –o wide' to see more info
- 'kubectl get namespaces' to see your namespaces in the control plane
- 'kubectl get pods –all-namespaces' to see all your workloads that are running from kubeproxy, etc
- 'kubectl get all --all-namespaces' to see a table of all resources
- Show them the 'kubectl get -h' flag to see help and inspect it.
- Show them to click on the namespaces online to see the YAML or JSON formatted info.
- Go online to see all the same information in GUI.
    - Click – Namespaces (kubectl get namespaces or 'kubectl get ns')
    - Workloads
    - Services and Ingresses

    - Node Pools (show that you can change the scaling to 2 or more nodes.)
    -

- ………………………..…………………………..………………….
- **SECTION 1**
- PLAY WITH A CLUSTER
- SOURCE (sections 2 and 4)- https://www.udemy.com/course/azure-kubernetes-service-with-azure-devops-and-terraform/learn/lecture/23660842#overview
- Go to the repo for 'azure-aks-kubernetes-masterclass/01-Create-AKS-Cluster/kube-manifests' dir in azure-aks-kubernetes-masterclass repo clone
    - Copy the 'kube-manifests' directory into the batch repo beforehand to make this quicker
- In Terminal in Kubernetes folder that holds the kube-manifests dir:
    - 'kubectl apply -f kube-manifests/'
    - to deploy the directory with the pods to your kluster created above
- 'kubectl get pods' to see the pods you've just created
- MOVE THIS TO AN APPROPRIATE PLACE - 'kubectl logs -f myapp1-deployment-58ccb86d9-hhjw4' to see the logs. There will be nothing to see here yet.
- 'kubectl describe pod <NameOfPod>' to get the looong description of the running pod.
    - Each pod is different but the readout is the same.
- Inspect the return. Especially the EVENTS at the bottom. Note that an image was pulled from docker hub.
- 'kubectl get deployments' to see the current deployments names
- 'kubectl get services' to see the deployed services
    - Here look at the external-ip and go to it.
- 'kubectl delete -f kube-manifests' to delete the deployment and pods from the cluster
- NOW you can run 'kubectl get pods' or 'kubectl get services' and nothing will be there but the original cluster.

….……………………..…………..……………………………..…………………..……………………..……….

- K8S ARCHITECTURE
- There are master and worker nodes. Every node has the Container(Docker) runtime.
- The master node has...
    - **Etcd** – key:value store for master and worker node info
    - **Kube-scheduler** - watches for new nodes and assigns them to a pod
    - **Kube-apiserver** - exposes K8s API for the CLI commands
    - **Kube controller Manager** – notices when nodes, containers, or endpoints go down and run new containers when needed.
        - **Node-controller** – notices and responds when a node goes down
        - **Replication controller** – maintains correct number of pods on replication controller object.
        - **Endpoints controller** – joins a service to a pod by populating the endpoints.
        - **Service account & token controller** - creates default accounts and API access for new namespaces.

    - **Cloud Controller Manager** – This component is only created if the cluster is in the cloud. It communicates with cloud services like checking if nodes have been deleted, runs controllers from your cloud provider (Azure Kubernetes Service).
        - **Node Controller** – checks with the cloud provider to see if a node has been deleted when it stops responding.
        - **Route Controller** – sets up routes int eh cloud infrastructure
        - **Service controller** – creates, updates, and deletes cloud providers load balancer.
- Worker nodes -
    - Container Runtime (Docker)
    - Kubelet – agent that runs on every node in the cluster. Makes sure that the correct container is running on the node
    - Kube-Proxy – a network proxy that runs on each node. Follows network rules (like https, CORS, etc).
- Concepts
    - POD – smallest instance of an app
    - REPLICASET – Makes sure that the correct number of identical pods is active
    - Deployment – runs the replicas and replaces instances that fail for become unresponsive. It rolls back and rolls out changes to the app, too.
    - Service – an abstraction for pods that provides a stable Virtual IP address. Balances the load on the pod... redirects traffic to replicas in round robin fashion.
- 1 container, 1 pod. A **container** is in a 1-to-1 relationship with a **pod**.
- The Worker node should never have multiple identical pods. You will almost always have just one container on a single pod. There is the concept of a 'sidecar container' that se rves to get data and give it to the other container on the same pod but that is rare.
- ……………………………………………………………………………………
- **SECTION 2**
- PLAYING WITH PODS
- 'az aks get-credentials --resource-group 06012021Batch_ResourceGroup --name demo1' to merge again the context.
- NOTE: if you deleted everything form the other cluster, you probably deleted the loadbalancer
- 'kubectl get nodes' to see the node running
- 'kubectl run apod --image stacksimplify/kubenginx:1.0.0' to run a pod directly from an image on Docker -- -- you can push the **rpsgameangularimage** image to try to see it running!
- 'kubectl get pods' to see the pod running
- 'kubectl describe pod rpsgameangularpod' to see all the details of the pod running the image
- 'kubectl get pods –o wide' to see more detail about all the pods and their IP addresses!
- You need to create a load balancer service to access the node itself.
    - When you create a **service** in your cluster, and target the internal (:80) port of the container (running on the pod), automatically, an external (public) port is created to allow people to access the app.
    - The Standard Load Balancer handles external requests and routes them to the loadbalancer which routes the request to the containers port (inside the pod).
    - Define a port and target port.
- Search 'public ip' in the search bar in portal.azure.

- NOTE: Make sure you are on the correct subscription
- Click on 06012021Batch resource group and note the IP address(20.88.226.240)
- NOW Search LoadBalancers and click on the 06012021batch resource group
- Click on Frontend IP configuration and note the public IP address (20.94.138.247)
- Go back to command line and create the pod (again) if needed.
- 'kubectl expose pod rpsgameangularpod --type=LoadBalancer --port=80 --name=rpsgamepod' to expose the pod externally using the LoadBalancer service.
- 'kubectl describe service rpsgamepod' to see the details of the service
- Get the **LoadBalancer Ingress:** ip address and see the app running
- Go back to Public IP, click the correct resource group, click on kubernetes load balancer or the type:public ip address to see the public IP address listed.
- Go back to LoadBalancers, click on the correct resource group, the Public IP Addresses, and see the public IP address there.
- Go back to your **Load balancer**, click on **load balancing rules** and see that a new load balancing rule has been created.
- All the above is automatically created.
- …..............................…..........….........................…….…..
- Go back to the **demo1 kubernetes** service and look under **Kubernetes resources**.
- See that everything you see in the command line in here.
- Go to services and ingresses and click on the pod you just created
- Click on the YAML representation of the pod.
-
-