AKP CKAD Notes  
  
<https://www.udemy.com/course/certified-kubernetes-application-developer/learn/lecture/12299352?start=45#overview>

|  |  |
| --- | --- |
| 12/17/19 | Multi container pod (IDesign) patterns   1. Ambassador 2. Adapter 3. Sidecar   Certs   * CKAD   + Cloud Native Computing Foundation (CNCF.io)   + $300 with 1 free re-take w/in 12 mos   + On-line Exam with proctor   + Performance based, not multiple choice   + **Code: DEVOPS15** – for registering for the CKA or CKAD exams at Linux Foundation to get a 15% discount.   + <https://www.cncf.io/certification/ckad/>   + Candidate Handbook: <https://www.cncf.io/certification/candidate-handbook>   + Exam Tips: <https://www2.thelinuxfoundation.org/ckad-tips> * CK Administrator   + A little more difficult   Dev to learn   * Python * NodeJS   Architecture   * etcd   + Key-value store * kubelet   + installed on each node   + monitors containers * Node   + Where the pods are launched   + Master Node     - Orchestrates containters     - Monitors the worker nodes     - kube-apiserver     - etcd (kvs)     - controller     - scheduler   + Worker Node (minion)     - kubelet     - Container runtime       * Docker * Cluster is a group of Nodes   + Single node   + Multi node   kubectl   * Run (to launch containers in pods)   + –image to point to image   + K get pods to see pods * cluster-info * get nodes (to see nodes)   pod   * Can have multiple containers * Application containers   + Helper containers * Containers in a pod share   + Storage   + Network namespace   + Fate (created, destroyed together)   yaml   * lead dash indicates first item in a list * editor: Pycharm (JetBrains family )   pod definition   * specification   + apiVersion (v1 for pods)   + kind (type of object - pod)   + metadata (specific properties only)     - name     - labels (any kv pair)   + spec     - per kube object specs     - list (-) of containers   edit an existing pod (create the yaml)  ***kubectl get pod <pod-name> -o yaml > pod-definition.yam***  Replication Controller (being replaced by ReplicaSet)   * run multiple pods of the same type for redundancy * also spins new pod (or only) if last pod fails * for load balancing (across the cluster)   + pods may not be on the same node   apiVersion: v1  kind: ReplicationController  metadata:  name: myapp-rc  labels:  app: myapp  type front-end  set:  template: (pod template)  metadata: (pod)  name:  labels:  app:  type:…  replicas: 3  kubectl get replicationcontroller  Replica Set (requires selector)   * a process that monitors pods (filtered by label)   apiVersion: apps/v1  kind: ReplicaSet  metadata: (name, labels)  spec:  template:  (pod def)  replicas: 3  selector:  matchLabels:  type: front-end  apiVersion: apps/v1  kind: ReplicaSet  metadata:  name: frontend  labels:  app: guestbook  tier: frontend  spec:  # modify replicas according to your case  replicas: 3  selector:  matchLabels:  tier: frontend  template:  metadata:  labels:  tier: frontend  spec:  containers:  - name: php-redis  image: gcr.io/google\_samples/gb-frontend:v3  kubectl create -f (file)  kubectl get replicaset  can monitor existing, but if not will create them (if none are running that match the matchLabels)  To Scale up number of replicas:   * edit yaml, then kubectl replace -f (file.yml) * kubectl scale –replicas=6 -f (file.yml)   + does not change the value in file.yaml * kubectl scale –replicas=6 replicaset [rs name] * kubectl scale –replicas=3 deployment [deployment name] * can be scaled automatically based on load (later topic)   Deployment   * wraps ReplicaSet * same yaml as ReplicaSet * deploys a Replicaset   kubectl get all to see all objects  Imperative commands  **Create an NGINX Pod** kubectl run --generator=run-pod/v1 nginx --image=nginx  **Generate POD Manifest YAML file (-o yaml). Don't create it(--dry-run)** kubectl run --generator=run-pod/v1 nginx --image=nginx --dry-run -o yaml  **Create a deployment** kubectl run --generator=deployment/v1beta1 nginx --image=nginx  Or the newer recommended way:  kubectl create deployment --image=nginx nginx  ***deployment/v1beta1 has been deprecated***  kubectl create deployment --image=nginx nginx --dry-run -o yaml  kubectl create deployment does not have a --replicas option. You could first create it and then scale it using the kubectl scale command.  **Create a Service named nginx of type NodePort and expose it on port 30080 on the nodes:** kubectl create service nodeport nginx --tcp=80:80 --node-port=30080 --dry-run -o yaml  kubectl expose deployment webapp --type=NodePort --port=8080 --name=webapp-service --dry-run -o yaml > webapp-service.yaml  **Expose redis pod on port 6379 via clusterip service:** kubectl expose pod redis --port=6379 --name redis-service  **NodePort Example:**   * The *port* is 8080 which represents that ***webapp-service*** can be accessed ***by other services in the cluster*** at port 8080. * The *targetPort*is 8170 which represents the ***webapp-service*** is ***actually running on port 8170 on pods*** * The *nodePort* is 32222 which represents that ***webapp-service*** can be ***accessed by external users via kube-proxy*** on port 32222   It is the **ipTables**in Kubernetes which does the magic. It maintains the mapping of nodePort vs targetPort. Kube-Proxy uses the ipTables to resolve the requests coming on a specific nodePort and redirect them to appropriate pod(s). |
| 12/23 | Namespaces   * Inherent   + kube-system   + Default   + kube-public * Parameters   + Policies   + Quotas * To connect to a service in the “dev” namespace   + [service-name].dev.svc.cluster.local     - cluster.local is the domain name of the local cluster     - svc is the service subdomain      * kubectl get pods –namespace=[ns-name] * kubectl apply –f pdef.yaml --namespace=[ns-name] * put in def file:   Create a new namespace    OR  kubectl create namespace [dev]  Set the ref namespace in the current context    show pods in all namespaces  kubectl get pods --all-namespaces  set up a resource quota: |
| 1/3/20 | Docker   * docker ps (running containers) * docker ps -a (all containters including stopped) * docker run Ubuntu * docker run Ubunto [sleep 5] overwrites the CMD parameter from the docker file * FROM ubuntu CMD sleep 5 * ***OR*** FROM Ubuntu CMD [“sleep”,”5”] * docker build ubuntu-sleeper . * docker run ubuntu-sleeper * FROM ubuntu ***ENTRYPOINT*** [“sleep”] * docker run ubuntu-sleeper [10] seconds to sleep [gets ***appended*** to the ENTRYPOINT command] * FROM ubuntu ENTRYPOINT[“sleep”] CMD [“5”] -- default to use if not provided in the “docker run” * docker run ubuntu-sleeper 10 -- overrides the default value of 5 * docker run --entrypoint sleep 2.0 ubuntu-sleeper 10   + result command: sleep 2.0 10 |
| 1/13/20 | Start image with run command      Docker run with command “python app.py”    Docker run with command “python app.py –colorred”    Pod config: command (“—color green”) [overrides the entrypoint, CMD from Docker]    {Kube pod definition overrides the (concatenated) Docker entry point command}  Override the args only:    Environment Variables:    ConfigMaps: config data in key value pairs   1. create    1. imperitive (command line)    2. declarative (object file) 2. inject into pod   Config Maps:    Data is stored under the name of the file  Create configmap and inject it into a Pod:    3 ways to inject config map: (all values from configmap [envFrom], single value from configmap [env, valueFrom], mount configmap as a volume [volumes]) |
|  |  |
|  |  |
|  |  |
|  |  |