**פקודות**

הצגת כל ה Kubernetes Clusters: kubectl config get-contexts

בחירת ה Kubernetes Cluster לעבודה: kubectl config use-context <K8S\_Cluster\_Name>

הצגת כל ה NameSpaces: kubectl get namespaces

הצגת כל ה Pod-ים תחת NameSpace כלשהו (עם פירוט): kubectl get pods -n <NameSpace> -o wide

הצגת כל הלוגים על Pod כלשהו וצפייה ב Live בלוגים: kubectl logs <PodName> -n <NameSpace> --follow

הצגת פרטים על Resource כלשהו: kubectl describe <ResourceType> <ResourceName> -n <NameSpace>

פתיחת Shell של Pod ספציפי kubectl exec -it <PodName> -n <NameSpace> -- /bin/bash

לבצע פריסה על פי Kubernetes.yaml לוקאלי: kubectl apply -f **Name**.yaml -n <NameSpace>

הצגת שימוש במשאבים של Pod: kubectl top pod <PodName> -n <NameSpace>

הצגת הסטטוס של Job: kubectl get jobs <JobName> -n <NameSpace> -o json | jq '.status.conditions’

שליפת ה Kubernetes.yaml של Deployment:

kubectl get deployment <DeploymentName> -n <NameSpace> -o yaml > **Name.**yaml

מחיקת כל ה Pod-ים בסביבה מסויימת על פי מילה המוכלת בשם

FOR /F %G IN ('kubectl get pods -n <Environment> -o name ^| findstr <Name>') DO "kubectl delete %G"

רשימת ממויינת של כל ה Pod-ים על פי השימוש בזיכרון (RAM), בהנחה שכל הגדלים הם Mi:

Kubectl top pods -A –no-headers | sed “s/Mi//g” | sort -g -k4

רשימת כל ה deployments תחת namespace כלשהו והאם יש להם מילה כלשהי ב yaml:

Kubectl get deploy -n **<NameSpace>** --no-headers | awk {‘print $1’} | xargs -I {} sh -c “if kubectl get deploy -n **<NameSpace>** {} -o yaml | grep **WORD** > /dev/null; then echo {} --> yes; else echo {} --> no; fi”

הצגת פקודת ה cURL שמתבצעת מאחורי הקלעים בעת ביצוע פקודת ה kubectl:

Kubectl get pods -n dev - l app=ui **--v=8**

הצגת ה memory limit מתוך container:

cat /sys/fs/cgroup/memory/memory.limit\_in\_bytes | awk {'print $1 / 1024 / 1024 / 1024'}

הצגת כל ה image-ים שבשימוש בכל הקלאסטרים המחוברים

kubectl config get-contexts --no-headers | awk '{print $2}' | xargs -I {} sh -c 'kubectl config use-context {}; kubectl get pods --namespace default -o jsonpath="{.items[\*].spec['\''initContainers'\'', '\''containers'\''][\*].image}" | tr -s "[:space:]" "\n" | sort | uniq -c'

**מדריכים + פקודות חיצוניות**

**התחברות ל AKS Node:**

* Create SSH key.
  + On Windows, by default located at c:\Users\<UserName>\.ssh\id\_rsa
* Update the ssh key of the AKS node:
  + Open **Powershell** and run the following command (By default the username is “azureuser”):

Az vm user update --resource-group <ResourceGroupName> --name <AKS\_NodeName> --username <username> --ssh-key-value <Path\_To\_”id\_rsa.pub”\_file>

* Create special pod for the ssh connection:
  + Open **CMD** and run the following command:

kubectl run --generator=run-pod/v1 -it --rm aks-ssh --image=debian

* + After the Pod’s shell opened, update apt-get and install openssh with the following command:

apt-get update && apt-get install openssh-client -y

**DO NOT CLOSE THIS WINDOW**

* Open **new CMD** and copy the id\_rsa file to the pod:

kubectl cp <Path\_to\_"id\_rsa"\_file> aks-ssh:/id\_rsa

* Go back to the Pod’s shell, change the permissions of the id\_rsa file that been copied and connect via ssh:

chmod 0600 id\_rsa

ssh -i id\_rsa <UserName>@<NodeIP>

**התקנת כלים לביצוע dotnet core memory dump על Pod (Ubuntu machine):**

* Register Microsoft key and feed and install procdump:
  + apt update
  + apt-get install wget
  + wget -q [https://packages.microsoft.com/config/ubuntu/**18.04**/packages-microsoft-prod.deb -O packages-microsoft-prod.deb](https://packages.microsoft.com/config/ubuntu/18.04/packages-microsoft-prod.deb%20-O%20packages-microsoft-prod.deb)
  + dpkg -i packages-microsoft-prod.deb
  + apt-get update
  + apt-get install procdump
* Install the debugger tools:
  + apt-get install lldb
  + apt-get install dotnet-sdk-3.1
  + dotnet tool install -g dotnet-sos
  + PATH="$PATH:~/.dotnet/tools"
  + dotnet-sos install
* Create the memory dump file:
  + coredump -p <PID\_Number>
* Open the memory dump file and investigate:
  + lldb `which-dotnet`-c <Dump\_Name>
  + sos DumpHeap -stat

**שליפת נתונים ישירות מה API:**

* Retrieve the Access Token for the specified cluster [This can be done in advance].
  + Kubectl config use-context <Cluster\_Context\_Name>
  + TOKEN=$(kubectl get secrets -o jsonpath="{.items[?(@.metadata.annotations['kubernetes\.io/service-account\.name']=='default')].data.token}"|base64 --decode)
* Use variable for the specified cluster.
  + export CLUSTER\_NAME="<Cluster\_Name>"
* Retrieve cluster API url.
  + APISERVER=$(kubectl config view -o jsonpath="{.clusters[?(@.name==\"$CLUSTER\_NAME\")].cluster.server}")
* Use the Curl command for retrieving information form the cluster.
  + curl -X GET $APISERVER/api/v1/namespaces/dev/pods?fieldSelector="status.phase%21%3DRunnin&labelSelector=app%3Dui" --header "Authorization: Bearer $TOKEN" --insecure
    - “**$APISERVER/api/v1/namespaces/dev/pods**” - All pods.
    - “**?**” - Where (find).
    - "**status.phase%21%3DRunning&labelSelector=app%3Dui**" - “status phase” is not “Running” and has label “app” with value “ui”.
      * **%21** - not
      * **%3D** - equals