What The Hack – Challenge Guide

Kubernetes as Infrastructure

Challenge Set 0: Pre-requisites - Ready, Set, GO!

Challenges:

- Make sure that you have joined the Teams group for this track. The first person on your team at your table should create a new channel in this Team with your team name.
- Install the recommended toolset:
    - Windows Subsystem for Linux
    - Azure CLI
        - Update to the latest
        - Must be at least version 2.0.42
        - **NOTE:** If you're running into issues running Azure CLI command on Windows, Disable Global Protect (VPN)
    - Visual Studio Code
- **NOTE:** You can start the next challenge even if this one is still running by using the the Azure Cloud Shell.
- **Tip:** You can complete almost all of the challenges with the Azure Cloud Shell! But be a good cloud architect and make sure you have experience installing the tools locally.

Challenge Set 1: Got Containers?

Challenges:

- Deploy build agent VM with Linux + Docker using provided ARM Template & parameters file in the "Files" tab of the Team's General channel.Run the Fab Medical application locally on the VM & verify access
    - Each part of the app (api & web) runs independently.
    - Build the API app by navigating to the **content-api** folder and run "**npm install**".
    - To start the app, run "**nodejs ./server.js &**"
    - Verify the API app runs by hitting its URL with one of the three function names. Eg: "**http://localhost:3000/speakers**"
    - Repeat for the steps above for the content-web app, but verify it's available via a browser on the Internet!
    - **NOTE:** The content-web app expects an environment variable named "**CONTENT_API_URL**" that points to the API app's URL.
- Create a Dockerfile for the content-api app that will:
    - Create a container based on the node:8 container image
    - Build the Node application like you did above (**Hint:** npm install)
    - Exposes the needed port
    - Starts the node application
- Create a Dockerfile for the content-web app that will:
    - Do the same as the Dockerfile for the content-api
    - Also sets the environment variable value as above
- Build Docker images for both content-api & content-web
- Run both containers you just built and verify that it is working.
    - **Hint:** Run the containers in 'detached' mode so that they run in the background.
    - **NOTE:** The containers need to run in the same network to talk to each other.
        - Create a Docker network named "fabmedical"
        - Run each container using the "fabmedical" network
        - **Hint:** Each container you run needs to have a "name" on the fabmedical network and this is how you access it from other containers on that network.
        - **Hint:** You can run your containers in "detached" mode so that the running container does NOT block your command prompt.

Challenge Set 2: Azure Container Registry

Challenges:

- Deploy an Azure Container Registry (ACR)
- Ensure your ACR has proper permissions and credentials set up
- Login to your ACR
- Push your Docker container(s) to the ACR
- List all images in your ACR

Challenge Set 3: Introduction to Kubernetes

Challenges:

- Install the Kubernetes command line tool (kubectl).
    - **Hint**: This can be done easily with the Azure CLI.
- Create a new, multi-node AKS cluster.
    - Use the default Kubernetes version used by AKS.
    - The cluster will use basic networking and kubenet.
    - The cluster will use Availability Zones for improved worker node reliability.
- Use kubectl to prove that the cluster is a multi-node cluster and is working.
- Use kubectl to examine which availability zone each node is in.
- **Optional**: Bring up the Kubernetes dashboard in your browser
    - **Hint**: Again, the Azure CLI makes this very easy.
    - **NOTE 1:** This will not work if you are using an Ubuntu Server jump box to connect to your cluster.
    - **NOTE2**: You will need to look up how to enable the special permissions needed to access the dashboard.

Challenge Set 4: Your First Deployment

Challenges:

- **NOTE:** If you have not or cannot deploy your containers to the Azure Container Registry, we have staged the FabMedical apps on Docker Hub at these locations:
    - **API app**: whatthehackmsft/content-api
    - **Web app**: whatthehackmsft/content-web
- Deploy the **API app** from the command line using kubectl and YAML files:
    - **NOTE**: Sample YAML files to get you started can be found in the Files section of the General channel in Teams.
    - Number of pods: 1
    - Service: Internal
    - Port and Target Port: 3001
    - CPU: 0.5
    - Memory: 128MB
- We have not exposed the API app to the external world. Therefore, to test it you need to:
    - Figure out how to get a bash shell on the API app pod just deployed.
    - Curl the url of the "/speakers" end point.
    - You should get a huge json document in response.
- Deploy the **Web app** from the command line using kubectl and YAML files
    - **NOTE**: Sample YAML files to get you started can be found in the Files section of the General channel in Teams.
    - **NOTE**: The Web app expects to have an environment variable pointing to the URL of the API app named:
        - **CONTENT_API_URL**
    - Create a deployment yaml file for the Web app using the specs from the API app, except for:
        - Port and Target Port: 3000
    - Create a service yaml file to go with the deployment
        - **Hint**: Not all "types" of Services are exposed to the outside world
    - **NOTE:** Applying your YAML files with kubectl can be done over and over as you update the YAML file. Only the delta will be changed.
    - **NOTE**: The Kubernetes documentation site is your friend. The full YAML specs can be found there: https://kubernetes.io/docs
- Find out the External IP that was assigned to your service. You can use kubectl for this.
- Test the application by browsing to the Web app's external IP and port and seeing the front page come up.
    - Ensure that you see a list of both speakers and sessions on their respective pages.
    - If you don't see the lists, then the web app is not able to communicate with the API app.

Challenge Set 5: Scale and High Availability

Challenges

- Scale the nodes in the AKS cluster from 3 to 1. Make sure you watch the pods after you perform the scale operation. You can use an Azure CLI command like the following to do this:

  az aks nodepool scale --resource-group wth-rg01-poc --cluster-name wth-aks01-poc --name nodepool1 --node-count 1

- Scale the **Web** app to 2 instances
  - This should be done by modifying the YAML file for the Web app and re-deploying it
- Scale the **API** app to 4 instances using the same technique as above.
- Watch pods using kubectl with its special watch option (the docs are your friend!).
  - You will find an error occurs because the cluster does not have enough resources to support that many instances.
  - There are three ways to fix this: increase the size of your cluster, decrease the resources needed by the deployments or deploy the cluster autoscaler to your cluster.
- To fully deploy the application, you will need 4 instances of the API app running and 2 instances of the Web app.
  - **Hint:** If you fixed the issue above correctly (look at pod resource request!), you should be able to do this with the resources of your original cluster.
- When your cluster is fully deployed, browse to the "/stats.html" page of the web application.
  - Keep refreshing to see the API app's host name keep changing between the deployed instances.
- Scale the API app back down to 1, and immediately keep refreshing the "/stats.html" page.
  - You will notice that without any downtime it now directs traffic only to the single instance left.

Challenge Set 6: Deploy MongoDB to AKS

Challenges:

- Deploy a MongoDB container in a pod for v2 of the FabMedical app
- **Hint:** Check out the Docker Hub container registry and see what you can find.
- Confirm it is running with:
    - **kubectl exec -it <mongo pod name> -- mongo "--version"**

Challenge Set 7: Updates and Rollbacks

Challenges:

- We have staged an updated version of the app on Docker Hub with id and version:
  - **whatthehackmsft/content-web:v2**
  - **whatthehackmsft/content-api:v2**
- For version two, you will also need an initializer container available on Docker Hub at:
  - **whatthehackmsft/content-init**
  - Use the content-init "Job" yaml provided to run the initialization of MongoDB for our new version of the app.
- Perform a rolling update of content-web on your cluster to the new version of content-web
  - You'll be doing this from the command-line with a kubectl command (remember, Kubernetes docs are your friend!)
  - With kubectl and its watch feature you should be able to see new pods with the new version come online and the old pods terminate.
  - At the same time, hit the front page to see when you're on the new version by refreshing constantly until you see the conference dates updated to 2019.
- Now we are going to roll back this update.
  - Again, this is done from the command-line using a (different) kubectl command.
  - Confirm that we are back to the original version of the app by checking that the conference dates are back to 2017.
- Perform the update again, this time using the blue/green deployment methodology.
  - This time, make sure you update BOTH content-web and content-api.
  - You will need a separate deployment file using different tags.
  - Cut over is done by modifying the app's service to point to this new deployment.
- **NOTE**: The new version of content-api will need to know how to reach the MongoDB server. You will need to pass it an environment variable named: **MONGODB_CONNECTION** set to its URL:
  - mongodb://mongodb:27017/contentdb

Challenge Set 8: Storage

Challenges:

- Make sure that you are using the latest version of the Fabmedical container images:
  - **whatthehackmsft/content-api:v2**
  - **whatthehackmsft/content-web:v2**
- Destroy the previous MongoDB pod created in the Challenge Set 6.
- In this challenge you will provision the MongoDB pod with a persisted disk volume.
- Create two Azure data disks (one for the MongoDB configuration and another one for data)
- Create a deployment yaml for MongoDB to be deployed with the necessary configuration for using the volume as an Azure Data Disk.
  - Find the reference template in the Teams Files section: **template-mongodb-deploy.yml**
  - **NOTE**: You can use the same MongoDB container image from Docker Hub that you used in a previous challenge.
- Verify that MongoDB is working fine by connecting to the corresponding MongoDB Pod in the interactive mode. Make sure that the disks are associated correctly (Highlighted below)

  - **kubectl exec -it <mongo-db pod name> bash**

    ```
    root@mongo-db678745655b-f82vj:/# df -Th
    Filesystem     Type     Size  Used Avail Use% Mounted on
    overlay        overlay   30G  4.2G   25G  15% /
    tmpfs          tmpfs    1.7G     0  1.7G   0% /dev
    tmpfs          tmpfs    1.7G     0  1.7G   0% /sys/fs/cgroup
    /dev/sdc       ext4     2.0G  304M  1.5G  17% /data/db
    /dev/sdd       ext4     2.0G  3.0M  1.8G   1% /data/configdb
    /dev/sda1      ext4      30G  4.2G   25G  15% /etc/hosts
    shm            tmpfs     64M     0   64M   0% /dev/shm
    tmpfs          tmpfs    1.7G   12K  1.7G   1%
    /run/secrets/kubernetes.io/serviceaccount
    tmpfs          tmpfs    1.7G     0  1.7G   0% /sys/firmware
    ```

  - ```
    root@mongo-db678745655b-f82vj:/# mongo --version
    MongoDB shell version v3.6.1
    connecting to: mongodb://127.0.0.1:27017
    MongoDB server version: 3.6.1
    ```

- Initialize sample content (Speakers & Sessions data) in the mongo DB by running the content_init nodeJS application as a Kubernetes Job. Reference template is can be found in the Files area in Teams, called: **template-content-init-deploy.yml**
  - Logs for content-init will provide the detailed logs showing whether it was able to successfully connect and add the contents to the MongoDB. You can use the Kubernetes dashboard or kubectl to check the logs.
  - **NOTE**: If the AKS cluster was created using the default Service Principle then we must grant it permission to pull images from the ACR.
    - **Hint**: Have a look here: https://docs.microsoft.com/en-us/azure/container-registry/container-registry-auth-aks)
- Make sure that the "contentdb" database is populated by connecting to the MongoDB pod with an interactive terminal and verify the database collections.
    ```

```
root@mongo-db678745655b-f82vj:/# mongo
MongoDB shell version v3.6.1
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.1
>
> show dbs
admin       0.000GB
config      0.000GB
contentdb   0.000GB
local       0.000GB
```

- Destroy the MongoDB pod to prove that the data persisting to the disk
  - **kubectl delete deployment <mongo-db-deployment>**
- Recreate the Mongo Db Pod
  - **kubectl apply -f <mongo-db-deployment>**
- Once the Pod is created, verify that data is persisted to the Azure disks by following the previous MongoDB verification step.
- Update the MongoDB connection string in the content-api deployment YAML and deploy it, eg:
  - ```
    env:
      - name: MONGODB_CONNECTION
        value: mongodb://mongodb:27017/contentdb
    ```
- Verify the API can retrieve the data by calling the speaker / session end points with curl:
  - curl http://localhost:3001/speakers
  - curl http://localhost:3001/sessions
    ```

Challenge Set 9: Helm

Challenges:

- Fetch the script for installing Helm to the local machine where you will be using Helm
    - **curl https://raw.githubusercontent.com/helm/helm/master/scripts/get -o get_helm.sh**
- Set permissions that will make the script executable on the machine
    - **chmod 700 get_helm.sh**
- Install Helm client locally
    - **./get_helm.sh**
- Initial Helm and install in on the Kubernetes cluster
    - **helm init**
- Helm charts from a local package
    - Deploy the specified app for this challenge using the steps and yaml files provided. You will have to install the namespace, deployment and service yaml in that sequence.
    - Verify that the app has been deployed successfully by browsing the web app via the LoadBalancer IP address at the defined port number.
    - Redeploy the app to use v2 of the image and verify that the update is visible in the web app. Repeat these steps with v3 and v4 of the container images.
    - Convert these yaml files that were just used to deploy the app into a Helm chart using v1 of the container image.
    - Create a Helm package on the local machine for each version of the web app.
    - Remove the previously deployed app by deleting the namespace that was created via the yaml file
    - Deploy the helm chart with v1 of the image you just created.
    - Verify that the app has been deployed successfully
    - Make a note of the difference in number of steps involved in the deployment using individual yaml files vs the Helm chart
- Helm charts from a remote repo in Azure Container Registry
    - Push the Helm chart you just packaged to the remote ACR repo
    - Remove the package locally
    - Uninstall the app and redeploy it using the Helm chart from the ACR repo
    - Verify that the app has been deployed successfully

Challenge Set 10: Networking

Challenges:

- Delete the existing content-web deployment and service.
- Get the AKS cluster DNS host name from Azure Portal
- Install the nginx ingress controller.
- Deploy the content-web service and create an Ingress resource for it.
    - The reference template can be found in the Files section in Teams: template-web-ingress-deploy.yaml
    - Change the ACR & AKS DNS Name to match yours.
- Verify the DNS records are created, and if so, access the application using the DNS name, e.g http://fabmed.[YOUR_AKS_DNS_ID].[REGION].aksapp.io

Challenge Set 11: Operations and Monitoring

Challenges:

- Find the logs for your application's containers
    - Using the Kubernetes Dashboard
    - Using kubectl
    - Notice how you can check the logs of any of your pods individually.
- Start a bash shell into one of the containers running on a pod and check the list of running processes
- Find out if your pods had any errors.
- Azure Monitor:
    - Enable "Azure Monitor for Containers" on the AKS cluster
    - Show a screenshot of CPU and memory utilization of all nodes
    - Show a screenshot displaying logs from the frontend and backend containers
- Kibana:
    - Install Fluentd and Kibana resources on the Kubernetes cluster to use an external ElasticSearch cluster
    - Create a Kibana Dashboard that shows a summary of logs from the front-end app only
    - Create a Kibana Dashboard that shows a summary of logs from the back-end app only
    - Create a Kibana Dashboard that gives a count of all log events from the kubernetes cluster for the last 30 minutes only.