What The Hack – Proctor’s Guide

Deploying and Operating a Microservices Application in Azure

The intent of this Hack is to give attendees the ability to quickly deploy a running Microservice Application in Azure. There are a few caveats that should be emphasized up front:

- Everything being deployed is a container: As a core tenet of microservices is smaller, independent services that focus more on team independence (ex. To allow a polygot approach) that deemphasize platform. Pick the right platform (.Net/Java/Node JS/etc..) that fits both the problem at hand and skills of the team.
- This hack models is a dev/test model, with the pick of technologies and configuration management approach. In a Production mode, a given solution would have a container orchestrator (Kubernetes, Docker Compose, Mesos, etc…) and the management of secrets for the application would be using something like Azure Key Vault.
- Goal of this hack is to not only emphasis the ease of standing up a solution in Azure but also some of the underlying complexities more components in an architecture bring. Remember Microservices isn’t the solution to everything 😊

Emcee or Proctor Pre-setup

Someone who’s kicking off the hack needs to have access to a running version of the solution to demonstrate the end goal.

1. Clone the https://github.com/andywahr/microservices-workshop.git repo to your local, in the main directory are 2 files “deployHack.sh” and “deployHack.ps1” you will need.
2. Login to Azure Portal
3. Start Cloud Shell
4. Upload the appropriate file thru Cloud Shell based on if you pick to your cloud drive
    i. BASH: deployHack.sh
    ii. Powershell: deployHack.ps1
    iii. For info on uploading, look at https://docs.microsoft.com/en-us/azure/cloud-shell/persisting-shell-storage and see the section on Transfer local files to CloudShell
5. Both scripts take 3 parameters
    i. BASH: deployHack.sh <AzureDataCenter> <SubscriptionId> <BaseName>
    ii. Powershell: deployHack.ps1 -loc <AzureDataCenter> -sub <SubscriptionId> -baseName <BaseName>
6. After successfully deploying it, open up the portal, and go to resource group you just deployed, and get the URL of the web site.
7. Copy the Load.WebTest to a temp directory locally, edit it and change the ContextParameter Name="WebServer1" XML Element’s Value attribute to the HTTP (not HTTPS) URL of the web site.
8. Login to https://dev.azure.com/ and select your Organization
9. Select Test Plans, Load Tests, add a new Load Test, and upload your copy of the Load.WebTest.

10. Specify decent amount of load, something like 25 max vUsers running for 10 minutes to drive some decent traffic.
11. Hit Go

Demo'ing the End State:

After clicking around for a few minutes, you should have data streaming to the App insights. Particular areas of App Insights to highlight:

1. Application Map
2. Live Metrics Stream (while also clicking on the site)
3. Performance
4. Browsers

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

Lecture:

- Join the MS Team for this hack.
- Brief Intro the Azure Cloud Shell and the Azure CLI

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.

- Login to the Azure Portal
- In a separate window, start the Azure Cloud Shell and list your subscriptions with the Azure CLI
    1. Pick either Powershell or Bash based on your preference!
    2. We will be running the Azure CLI, if you want to run it on your local machine, you can just make sure you have the latest version
- We are going to need the Subscription ID of that will be used today, in your active Shell of choice, set a variable called ***sub*** to that subscription's Id (Guid format).
- **Tip:** There are 2 different ways to open the Azure Cloud Shell!

Proctor Notes & Guidelines

- Azure Cloud Shell: https://docs.microsoft.com/en-us/azure/cloud-shell/overview
- Install the Azure CLI: https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest
- Azure CLI Reference: https://docs.microsoft.com/en-us/cli/azure/reference-index?view=azure-cli-latest
- Commands to run (examples need to file in the XXX):
    - az account list
    - For Powershell: $sub="XXX"
    - For Bash: sub="XXXX"
- **REMEMBER**: If the attendee closes the window/browser, they have to redefine the variables!

Challenge Set 1: Create the first thing!

Lecture:

- What is a Resource Group

Challenges:

- Figure out which Azure data center is closest, and you will be using for this hack.
- Put the scripting name for the Azure Data Center (aka No Spaces 😉) that the Azure CLI uses in a variable called **loc**
    1. **Hint**: South Central US scripting name is ***southcentralus***
- In your shell, create a Resource Group in that data center
- **Tip:** Create a 6 to 8 letter label that is unique to you that can be used to compose names for Azure resources, since several items we are creating today will have public DNS names that can collide. This can help you create unique names.
    1. For example, if your name is Bob John Smith, maybe you use ***bjshack***.
- Put the name of the Resource Group in a variable called ***rg.***

Proctor Notes & Guidelines

- Azure Naming Conventions: https://docs.microsoft.com/en-us/azure/architecture/best-practices/naming-conventions
- Commands to run (examples need to file in the XXX):
    - For Powershell:
        - `$loc = "centralus"`
        - `$rg = "rg-XXX"`
    - For Bash:
        - `loc="centralus"`
        - `rg="rg-XXX"`
    - `az group create --name $rg --location $loc --subscription $sub`

Challenge Set 2: Always need App Insights

Lecture:

- What is App Insights
- What is an ARM template

Challenges:

- In your shell, deploy a public ARM Template for deploy an App Insights resource to have the various applications log to
    1. **ARM Template URL: https://raw.githubusercontent.com/andywahr/microservices-workshop/master/src/azuredeploy-appinsights.json**
        - **Two Parameters:**
            - **name:** Name of the App Insights Resource to Create
            - **regionId:** Scripting Name of the region to Provision in
- Put the InstrumentationKey of the App Insights resource you just created in a variable called ***appInsightsKey***

Proctor Notes & Guidelines

- App Insights: https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview
- ARM Templates: https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-authoring-templates
- Commands to run (examples need to file in the XXX or YYYY):

    -
      ```
      az group deployment create --name XXX --template-uri https://raw.githubusercontent.com/andywahr/microservices-workshop/master/src/azuredeploy-appinsights.json --parameters name=YYYY regionId=southcentralus --resource-group $rg --subscription $sub
      ```
    - For Powershell:

        -
          ```
          $appInsightsKey=az resource show --resource-group $rg --subscription $sub --resource-type Microsoft.Insights/components --name YYYY --query "properties.InstrumentationKey"
          ```
    - For Bash:

        -
          ```
          appInsightsKey=$(az resource show --resource-group $rg --subscription $sub --resource-type Microsoft.Insights/components --name YYYY --query "properties.InstrumentationKey" | tr -d '"')
          ```

Challenge Set 3: Get some Data!

Lecture:

- What is a CosmosDB

Challenges:

- In your shell, create a CosmosDB Account
- Put the PrimaryMasterKey of the CosmosDB Account you just created in a variable called ***cosmosPrimaryKey***

Proctor Notes & Guidelines

- Cosmos DB: https://docs.microsoft.com/en-us/azure/cosmos-db/
- Commands to run (examples need to file in the XXX or YYYY):

    -
      ```
      az cosmosdb create --subscription $sub --resource-group $rg --name XXX
      ```
    - For Powershell:

        -
          ```
          $cosmosPrimaryKey=az cosmosdb list-keys --resource-group $rg --subscription $sub --name XXX --query primaryMasterKey
          ```
    - For Bash:

        -
          ```
          cosmosPrimaryKey=$(az cosmosdb list-keys --resource-group $rg --subscription $sub --name XXX --query primaryMasterKey | tr -d '"')
          ```

Challenge Set 4: Deploy some containers to ACI!

Lecture:

- What is a Container
- What is ACI

Challenges:

- There are three different containers you are going to deploy the Azure Container Instances (az container)
    1. Data API**: microservicesdiscovery/travel-data-service**
    2. Itinerary API**: microservicesdiscovery/travel-itinerary-service**
    3. DataLoader utility (setup and seed Cosmos DB)**: microservicesdiscovery/travel-dataloader**
- All of them need 3 Environment Variables defined
    1. **DataAccountName**: Name of the Cosmos DB Account
    2. **DataAccountPassword**: Primary Key of the Cosmos DB Account
    3. **ApplicationInsights__InstrumentationKey**: Instrumentation Key of the App Insights Resource
- The first two (Data API and Itinerary API), both need to specify a DNS Name so they are easily addressable by the Web Site
- After you get the Data API deployed, use the Azure CLI to query the deployed container for the FQDN and store it in a variable called: **dataServiceUri**
- After you get the Itinerary API deployed, use the Azure CLI to query the deployed container for the FQDN and store it in a variable called: **itineraryServiceUri**
- Verify the Data Service by browsing the URL: http://$ will return a list of Airports
- Verify the Itinerary Service by browsing the URL: http://$ will return a 204
    1. **It should return a 404, but to get a positive test response, 204, which designates no content (as in the Itinerary was not found) is returned to provide the service is up and running.**
- After you get the DataLoader deployed use the Azure CLI to attach to the running container to see the console output.

Proctor Notes & Guidelines

- Azure Container Instances: https://docs.microsoft.com/en-us/azure/container-instances/
- Commands to run (examples need to file in the XXX#):

  Data API:

  - `az container create --subscription $sub --resource-group $rg --name XXX1 --image microservicesdiscovery/travel-data-service --dns-name-label XXX1 --environment-variables DataAccountName=$cosmoAccountName DataAccountPassword=$cosmosPrimaryKey ApplicationInsights__InstrumentationKey=$appInsightsKey`
  - For Powershell:
    - `$dataServiceUri=az container show -g $rg -n XXX1 --query "ipAddress.fqdn"`
  - For Bash:
    - `dataServiceUri=$(az container show -g $rg -n XXX1 --query "ipAddress.fqdn" | tr -d '"')`

  Itinerary API:

  - `az container create --subscription $sub --resource-group $rg --name XXX2 --image microservicesdiscovery/travel-itinerary-service --dns-name-label XXX2 --environment-variables DataAccountName=$cosmoAccountName DataAccountPassword=$cosmosPrimaryKey ApplicationInsights__InstrumentationKey=$appInsightsKey`
  - For Powershell:
    - `$itineraryServiceUri=az container show -g $rg -n XXX2 --query "ipAddress.fqdn"`
  - For Bash:
    - `itineraryServiceUri=$(az container show -g $rg -n XXX2 --query "ipAddress.fqdn" | tr -d '"')`

  DataLoader:

  - `az container create --subscription $sub --resource-group $rg --name XXX3 --image microservicesdiscovery/travel-dataloader --environment-variables DataAccountName=$cosmoAccountName DataAccountPassword=$cosmosPrimaryKey ApplicationInsights__InstrumentationKey=$appInsightsKey --restart-policy OnFailure`
  - `az container attach --subscription $sub --resource-group $rg --name XXX3`

Challenge Set 5: Deploy the Web Site

Lecture:

- What is an App Service

Challenges:

- In your shell, create a Standard Linux App Service Plan
- In your shell, create a Web App and set the **microservicesdiscovery/travel-web** as the container image for the Web App
- The following Application Settings need to be added:
    1. **DataAccountName**: Name of the Cosmos DB Account
    2. **DataAccountPassword**: Primary Key of the Cosmos DB Account
    3. **ApplicationInsights__InstrumentationKey**: Instrumentation Key of the App Insights Resource
    4. **DataServiceUrl:** The URL to the Data Service, only over HTTP
    5. **ItineraryServiceUrl:** The URL to the Itinerary Service, only over HTTP
- **HINT:** A FQDN (Fully Qualified Domain Name) is a name that resolves to IP Address(es). A URL also has a Scheme, like HTTP.
    1. FQDN: **www.contoso.com**
    2. URL with HTTP Scheme: **http://www.contoso.com**
- BROWSE THE URL of the App Service

Proctor Notes & Guidelines

- App Service Plans: https://docs.microsoft.com/en-us/azure/app-service/overview-hosting-plans
- App Services: https://docs.microsoft.com/en-us/azure/app-service/
- Commands to run (examples need to file in the XXX or YYYY):
    - `az appservice plan create --name asp-discovery --resource-group $rg --is-linux --location $loc --sku S1 --number-of-workers 1 --subscription $sub`
    - `az webapp create --subscription $sub --resource-group $rg --name $baseName-web --plan asp-discovery -i microservicesdiscovery/travel-web`
    - `az webapp config appsettings set --resource-group $rg --subscription $sub --name $baseName-web --settings DataAccountName=$cosmoAccountName DataAccountPassword=$cosmosPrimaryKey ApplicationInsights__InstrumentationKey=$appInsightsKey DataServiceUrl="http://$dataServiceUri/" ItineraryServiceUrl="http://$itineraryServiceUri/"`