# ***What The Hack: Hotel CaliVegas Dev Hack – Proctor’s Guide***

# *Note: You should read the entire set of challenges before starting. The first 4 challenges should be completed before moving on to the remaining challenges. Some of the additional goals can be started and completed in nearly any order. Some can be performed simultaneously, and some goals can only be achieved with certain platform or tool choices. Any pre-requisites are listed with the challenge.*

# **Scenario**

Build a cloud native application to allow guests to find, connect and share their Vegas hotel experiences using Azure PaaS and Serverless. Common application scenarios can be but not limited to building an online solution to support hotel customers in Vegas who wish to receive extra casino loyalty awards by sharing their posts and pictures at that hotel.

As part of building cloud native applications, you will also want to leverage continuous integration/continuous delivery practices from the very beginning. This means deploying your applications into your Azure subscriptions view Build and Release capabilities using Azure Pipelines (formally VSTS).

## Suggested Architecture (one architects interpretation)



# **Guidance and Recommendations**

* Please review all challenges before beginning
* Leverage PaaS and Serverless as much as possible (look at the technology hints to get clues on potential technology)
* For this hackathon, while working on the challenges is a goal, the more important goal is to work together as a team. Each team will have ‘rockstars’ who can easily complete the challenges. It is important to share knowledge and progress as a team, making sure everyone gets through the challenges together.

# **Pre-Requisites**

This is a list of pre-requisites needed to successfully complete the challenges. Some of these are items to deploy to your development machine. Some are decisions you should discuss and define as a team, like the language to use for development.

* Determine the language for the app and services. These can be the same or different depending on the skills of the team (ie ASP.NET Core Web app and Node.js API Service).
* Coding IDE – Visual Studio Code, Visual Studio Enterprise, *others?*
* Git for Windows – allows for working with Git repos locally on your machine as well as in VSTS. <https://gitscm.com/downloads>
* VSTS Project for your team.
  + Project Structure – the Azure DevOps (formally VSTS) project is pre-provisioned with all of the tasks required for each challenge, just not how they should be accomplished. This is where your team will work.
  + Under this project structure, you may organize your code repo as you see fit to complete the challenges and successfully work together.
* Determine Azure Subscription you will use for deployment

# **Challenges and Detail**

***Note***: Achieving goals 1-4 will provide basic functionality in Azure. Goals beyond that are stretch.

1. *Challenge 1 : Baseline app*

Description: This challenge creates the basic web app and is the core user experience. This application has features to display a list of hotels based on a search criteria.

Challenge considerations:

* 1. Have one team member provision your Azure DevOps project (<https://aka.ms/DTAHack>) and add the rest of the team as contributors (optional).
  2. Create and deploy your web application to Azure
  3. Ensure all source code is in a repo so that it can be shared amongst team members
  4. Build a UI to allow searching and displaying a list of hotels. The logic for this is in Challenge 2 and you can look at the json file in that challenge for the data structure. For now, you can mock the display with static data.
  5. Create a continuous integration build for your web application that is triggered when code is committed

**Proctor Info:**

* Suggested Components / Hints:
* Documentation and Github References:
  1. Create an ASP.NET Core web app in Azure: <https://docs.microsoft.com/en-us/azure/app-service/app-service-web-get-started-dotnet>
  2. Create an ASP.NET Core web app in Azure: <https://docs.microsoft.com/en-us/azure/app-service/app-service-web-tutorial-rest-api>
  3. VSTS Git Command Reference: <https://docs.microsoft.com/en-us/vsts/repos/git/command-prompt?view=vsts>
  4. Continuous Integration with VSTS: <https://docs.microsoft.com/en-us/vsts/pipelines/apps/aspnet/build-aspnet-4?view=vsts&tabs=vsts>
  5. Continuous Delivery to Azure with VSTS: <https://docs.microsoft.com/en-us/vsts/pipelines/apps/cd/deploy-webdeploy-webapps?view=vsts>

1. *Challenge 2 : Search Service API*Description: For this challenge, create the API for the hotel data. The API will require a City, State and radius for searching which returns a list of hotels. In addition, a hotel name can also be provided to filter the hotel list to a specific brand.  
     
   Challenge considerations:
   1. Prerequisites: Completion of Challenge #1 - Baseline App
   2. Consider splitting this app into another project v. keeping as part of the same project with respect to deployment and other challenges to come.
   3. This challenge includes a collection of json files. These files correspond to the collection of hotels your API will search.
   4. Ensure all source code is in a repo so that it can be shared amongst team members. It can be in the same or a different repo.
   5. Integrate this service into the core web app.
   6. Ensure this service is covered by a CI build.
   7. While mentioned in the general guidance, read all of the challenges to make sure decisions made during this challenge are inline with the requirements of future challenges.

**Proctor Info:**

* Suggested Components / Hints: Cosmos DB for the json content storage. Can be uploaded once the collection is created.
* References:
  1. Create an Azure Search Instance: <https://docs.microsoft.com/en-us/azure/search/search-create-service-portal>
  2. Crawl a SQL DB with Azure Search: <https://docs.microsoft.com/en-us/azure/search/search-indexer-tutorial>
  3. Index CosmosDB with Azure Search: <https://docs.microsoft.com/en-us/azure/search/search-howto-index-cosmosdb>
  4. <https://docs.microsoft.com/en-us/azure/cosmos-db/geospatial>
  5. ASP.NET Core Web API help pages with Swagger / Open API: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-2.1>

1. *Challenge 3 : Image Support*

Description: As part of the app functionality, users have the capability to upload images and associate them with a specific hotel. A user can see the list of images they uploaded as well as delete an image.

Challenge considerations:

* 1. Prerequisites: Challenge 1 - Baseline app & Challenge 2 – Search Service API
  2. It is ok to modify code from previous challenges to include additional functionality to support this challenge.
  3. Consider the data required to support this challenge and all possible storage options available in Azure for storing it.
  4. Often, photos taken from cell phones include more than just the image. There is metadata related to the camera and location of the picture. You may want to consider storing this metadata to support additional challenges.

**Proctor Info:**

* Suggested Components / Hints: Azure Storage Account blob storage for images. Data store for the associating the user and hotel with the images as well as image metadata. Possible options include Azure Table Store, Cosmos Db, Azure SQL, Azure MySQL, Azure PostGres.
* References:
  1. Azure Storage: <https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction>
  2. Azure CosmosDB <https://docs.microsoft.com/en-us/azure/cosmos-db/>
  3. Upload image data in the cloud with Azure Storage: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-upload-process-images?tabs=net

1. *Challenge 4 : Add Notifications*

Description: Objects landing in Azure can be used to trigger other Azure components that can then process this information asynchronously from the application that sent the data. Create a process in Azure that sends a message to the DTA Hackathon Team Member’s cell phone when a new image is uploaded in the app.

Challenge considerations:

* 1. Prerequisites: Add image upload to web app
  2. When sending SMS message to the DTA Hackathon Team Members, be sure to include your group name in the message so we know who it is from.

**Proctor Info:**

* Suggested Components / Hints: Logic Apps. On the proctor side, we will have our own logic app that is triggered when a new message is posted to the same group. We can use this to populate a web page showing the list of groups getting to this point in the challenge.
* References:
  1. Manage Messages in Twilio with Azure Logic Apps: <https://docs.microsoft.com/en-us/azure/connectors/connectors-create-api-twilio>
  2. Twilio Connector for Logic Apps: <https://docs.microsoft.com/en-us/azure/connectors/connectors-create-api-twilio>
  3. Sending messages to Yammer for Logic Apps: <https://docs.microsoft.com/en-us/azure/connectors/connectors-create-api-yammer>

1. *Challenge 5 : Secure Application Configuration Secrets*

Description: Throughout the first 4 challenges, you needed to use secrets or connection strings to write or read information from the various Azure components. Since this is a hackathon, you likely stored them in configuration files or even hard coded them to complete the challenge quickly. Well, in this challenge, it is time to do the right thing and store them securely outside of your code.

Challenge considerations:

* 1. Prerequisites: Challenges 1-4
  2. Now that you are moving these secrets out of your code, you also need to handle this new storage location as part of your release pipeline in VSTS.

**Proctor Info:**

* Suggested Components / Hints: Leverage Key Vault – code should have the functionality to call KV to get the keys.
* References:
  1. Configure an Azure Web App to Read a Secret from Key Vault: <https://docs.microsoft.com/en-us/azure/key-vault/tutorial-web-application-keyvault>

1. *Challenge 6 : Create Thumbnails of Uploaded Images*

Description: Back in Challenge 3, you included the functionality to upload images through the web app. These images can be quite large depending on the camera used. It would be great to have reduced size images to use when just listing the collection of images uploaded. Include functionality to use these reduced sized images when viewing the list of images uploaded by a user from Challenge 3.

Challenge considerations:

* 1. Prerequisites: Challenge 3 – Image Support
  2. Depending on how you stored your images, you may want to create segregated or different locations for these reduced sized images.
  3. You can take advantage of previously created components to trigger this event.

**Proctor Info:**

* Suggested Components / Hints: Event Design (e.g. asynchronous 🡪 Event Grid), Cognitive Services and Image Tagging. This could also be an additional step in the Logic App created to send the notification.
* References:
  1. Automate resizing uploaded images using Event Grid: <https://docs.microsoft.com/en-us/azure/event-grid/resize-images-on-storage-blob-upload-event?tabs=net>

1. *Challenge 7 : Add Realtime Image Updates*

Description: Over the course of the challenges, you included a number of asynchronous processes to handle images and send messages. This challenge sends updates back to the web app so they are visible without refreshing the page.

Challenge considerations:

* 1. Prerequisites: Challenges 3 - View Thumbnails & Challenge 6 - Add image support
  2. Use the smaller sized images when presenting images as part of the list of images uploaded.

**Proctor Info:**

* Suggested Components / Hints: Azure SignalR Service
* References:
  1. Create a Chat Room with SignalR Service: <https://docs.microsoft.com/en-us/azure/azure-signalr/signalr-quickstart-dotnet-core>
  2. Azure SignalR Samples: <https://github.com/aspnet/AzureSignalR-samples>

1. *Challenge 8 : Add authentication*

Description: The ability to upload images should be a function that requires authentication. Secure the Add Image functionality of the app while maintaining anonymous access to the rest of the application.

Challenge considerations:

* 1. Prerequisites: Challenge 3 - Add image support
  2. The web app upload page requires an authenticated user to post images.
  3. The Add Image Service should also be secured so users cannot simply call the endpoint outside of the webapp.

**Proctor Info:**

* Suggested Components / Hints:
* References:
  1. Authentication and authorization in Azure App Service: <https://docs.microsoft.com/en-us/azure/app-service/app-service-authentication-overview>
  2. Protect an API by using OAuth 2.0 with Azure Active Directory and API Management: <https://docs.microsoft.com/en-us/azure/api-management/api-management-howto-protect-backend-with-aad>

1. *Challenge 9 : Add DevOps practices*

Description: Now that the app includes the full list of features, there are some things that can be included to showcase DevOps practices. One of those is the inclusion of unit testing of important components of the application. In a real world app, you would likely do these while creating each portion of the application. We are placing this step at the end of the process to allow the hackathon to focus on the getting a working application in place during the time allowed.

Challenge considerations:

* 1. Prerequisites: Challenge 1-4
  2. Add unit tests to web app and ensure test results are reported in builds
  3. Add unit tests to search service and ensure test results are reported in builds

**Proctor Info:**

* Suggested Components / Hints:
* References:
  1. Get Started with Unit Testing: <https://docs.microsoft.com/en-us/visualstudio/test/getting-started-with-unit-testing?view=vs-2017>

1. *Challenge 10 : Add Infrastructure as Code*

Description: Now that we have all of the components together, let’s move this current ‘development environment’ to a new environment inside of your Azure subscription. Recreate the current environment in a new resource group to use for your ‘production environment’.

Challenge considerations:

* 1. Prerequisites: Baseline App
  2. Capture all Azure resources (including any Logic Apps, etc.) as ARM templates or Terraform and commit to source control
  3. Add resource provisioning/updates from ARM/Terraform to a new or existing build/release pipeline
  4. Implement blue-green deployments to minimized down-time

**Proctor Info:**

* Suggested Components / Hints:
* References:
  1. Export an Azure Resource Manager template from existing resources: <https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-export-template>

# **Dependency Map** What can be done concurrently vs. sequentially

# Working table from the call / from the Jeopardy board

|  |  |  |  |
| --- | --- | --- | --- |
| App Dev | Hint | Bonus | DevOps |
| Service to Send SMS Text Messages to Hack Team | **Logic Apps & Twilio Connector** |  | **VSTS Site**  **Repo for Code** |
| Service to List Vegas Hotels for Web & Mobile Clients | **Azure Function** | **Swagger**  **APIM** | **CI/CD** |
| Display & search Vegas hotel list on a web site | **Azure Web Apps** | **SignalR**  **Containerized**  **Maps** | **Blue/Green** |
| Image upload to web site from hotel guests, display thumbnails | **Eventing**  **Function**  **Cognitive Services** | **Image classification using Cognitive Services** |  |
| Allow authorized access hotel guests | **Azure AD**  **Key Vault for Secrets** | **Oauth between all services** |  |
|  |  |  |  |