Managing Azure from Haskell — Summary

Premise Managing cloud infrastructure is an inherently transformational process, and nothing models transformations better than Functional Programming. This dissertation is meant to solidify the claim by presenting the building of *Haskzure*, a high-level Haskell library for interacting with Microsoft's Azure cloud platform.

Example To showcase *Haskzure*'s capabilities; a complete code sample is presented below in accordance with the following scenario:

- two virtual networks need creating following the same set of basic configuration options but in two different Azure locations.
- after creation; both need to be modified to add a particular server's IP address to their list of DNS providers to be used by VM's in those networks.

In the following we will further discuss this example (particularly the definition of deployVirtualNetworks) so reviewing it now is advised.

Analysis Upon looking at the example; there is little indication of anything extraordinary. As per the language bindings for Azure of any other programming language; we follow the canonical "define our parameters — run our functions on them" pattern. There are, however, some extremely subtle but invaluable things going on under the hood:

- all the data structures which comprise (and include) the mainline VirtualNetwork are rigid, typesafe datastructures. WestUS and NorthEurope, for example, are not plain strings, but both options of the well defined Location enum.
- most notably, there is an utter lack of any sort of verbose credential passing or cumbersome error checking steps after each operation. These are in no way ignored, but are handled entirely transparently through Haskell's most fundamental abstraction method: the monad (in this case, the Azure monad).

```haskell
import Control.Monad
import Data.Default

import Haskzure.Core      (Azure (..), Location(..),
                           ResourceId, createOrUpdate)
import Haskzure.Resources (VirtualNetwork(..), DhcpOptions(..), Subnet(..))

-- base configuration of all our virtual networks:
baseVN :: VirtualNetwork
baseVN = (def :: VirtualNetwork) {
    dhcpOptions = DhcpOptions { dnsServers = ["8.8.4.4", "8.8.8.8"]},
    addressSpace = "10.7.0.0/16",
    subnets = [Subnet "10.7.0.0/24"]
}

-- we define a list of two virtual networks based on the above configuration;
-- one in the Western US Azure region; the other in Northern Europe:
virtualNetworks :: [VirtualNetwork]
virtualNetworks = [baseVN { location = WestUS }, baseVN { location = NorthEurope }]

deployVirtualNetworks :: Credentials -> Azure [ResourceId]
deployVirtualNetworks = do
    -- create our two virtual networks
    vns <- mapM createOrUpdate virtualNetworks

    -- now add "9.9.9.9" as a DNS server to both the virtual networks.
    -- for this we define a function:
    let newDnses = DhcpOptions (("9.9.9.9" :) . dnsServers . dhcpOptions)

    -- and map it over our two virtual networks:
    vns <- mapM (\vn ->
            createOrUpdate $ vn { dhcpOptions = newDnses vn }) vns

    return $ map virtualNetworkId vns

main :: IO ()
main = do
    -- note the initial passing in of the credentials:
    vnids <- runAzure deployVirtualNetworks credentials
    print $ "The virtual network ID's are: " ++ show vnids
```

Implementation Azure is an incredibly rich cloud platform, with it's most recent count of individual API calls ranging in the *1400s*. As such, the writing of a client library for a particular language is a gargantuate task.

Haskzure works by generating its internal data structures based on the *Swagger* API description format in which Microsoft itself publishes the specifications for their Azure Resource Manager (ARM) APIs. Provided these data structures, generic low-level operations are used to interact directly with Azure, which are then unified under a single high-level set of functions and structures.
At the top of this layering lies the Azure monad, which provides a simple imperative-like domain-specific language for defining, querying and deleting numerous types of ARM cloud resources such as:

- compute resources: virtual machines (VMs) and their sub-resources.
- storage resources: virtual machine images and data volumes.
- networking resources: virtual networks with their subnets, network interface cards for VMs and other related resources.

In reality, the Azure monad is actually a monad stack providing a default base to start from, whilst the AzureT monad transformer contains its critical functionality and offers an easy means of being composed with other monads to allow for additional features such as logging, custom error handling and specific operation execution models (deploying resources either sequentially or in parallel). More details in this sense are provided in the dissertation itself.

Conclusions Overall, the power which comes with being able to manipulate cloud resources through concise functional code is a great one, and also opens the road for new and even more exciting possibilities further outlined in the full paper.