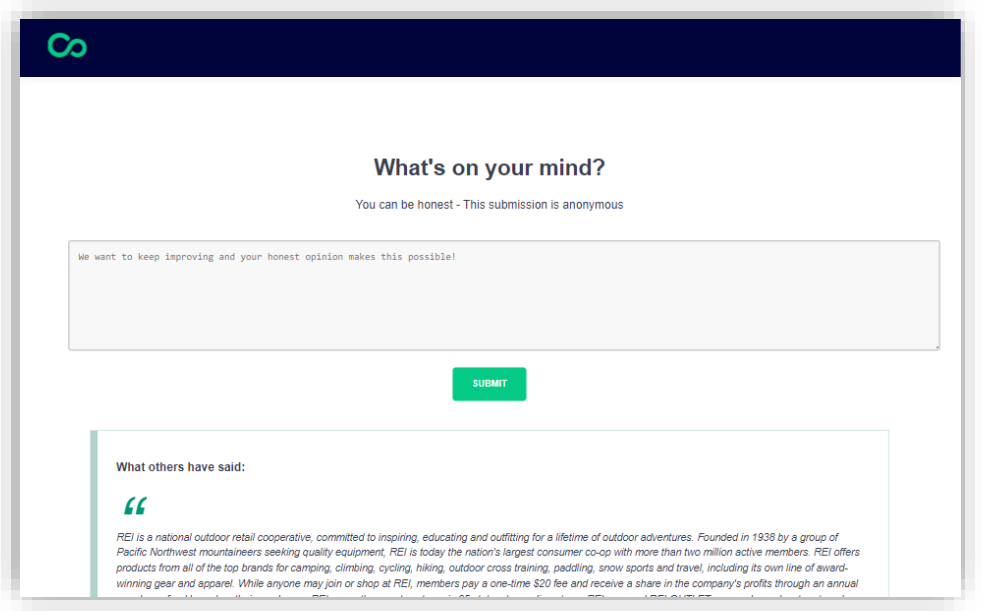
Summer of Azure 2022-2023

AzureAnonymous



A Real-World Project Created For:

COTISS

By Chris Chong

A quick 10-minute demo of this project can be found at https://youtu.be/PwTDgw8_Rc8

Contents

Personal goals

The real-world project brief and scope was quite open to interpretation and direction, so Christelle and I took the first 3 days to experiment and decide how we wanted to progress. My initial goal was to jump straight to Azure Functions and automated scripts, but after starting on that, I realised that meant that I would miss out on quite a few networking basics. I felt that this would mean that my knowledge wouldn't be as easily transferable to other cloud providers, which isn't ideal since I'm concurrently learning to implement a highly available/highly resilient architecture on AWS for work.

I am quite fortunate to have collated a broad understanding of how networking works over the years, but have not actually had the opportunity to implement any of it on a practical basis. Because of this, I set my personal goals as:

1. To design, configure, and implement the infrastructure and application from the ground up
2. To learn how to build the simplest and cleanest version of the application from scratch, but with healthy documentation and version control basics setup
3. To learn how to setup a database and create a script that allows it to talk to the application – Something that I've personally wanted to learn for a while now
4. To try and understand a little more about costs – What can be skimped on, and where money is best spent

While my application turned out looking extremely basic at the end of the two weeks (both in architecture as well as application design), I ended up with a much clearer understanding of how a basic application is designed, hosted, and secured, as well as some of the ways that Azure can help manage parts of the infrastructure and platforms on our behalf.

User Installation Guide

Prerequisites

This application was built and tested to be run on the following technology stack:

- **VM size:** Standard B1ls (1 vcpu, 0.5 GiB memory, suitable for test loads/env only)
- **OS:** Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-1029-azure x86_64)
- **Webserver:** Nginx/1.18.0 (Ubuntu)
- **Packages** installed on webserver:
    - Python 3.8.10
    - PyMongo==4.3.3
    - python-dotenv==0.21.0
    - Flask 2.2.2
      Gunicorn (version 20.0.4)
- **Database:** CosmosDB, using MongoDB API
- **Scripts:** Python and HTML/CSS

Application Setup

The Azure Anonymous application can be downloaded at:
https://github.com/csidon/AzAnonMontygo
Follow the installation steps in readme.txt

If you have been given access to an Azure account (instructor access only), you will need to:

1. Go to Resource Group **AzureAnonymous-dec2022-web-rg**

2. Start the VMs
3. **SSH** into the VMs (private key will be sent to you via Slack)
   (Note that SSH port 22 is exposed to the Internet. This should not be the case in a production environment but has been set as such for easy setup)
4. Login as **azureuser**
5. Change directory into project file
   cd AzAnonMontygo
6. Start gunicorn and set it to keep running in the background
   (these steps need to be carried out each time the VM is restarted. Ideally this should be a script that runs automagically each time the VM is started up)
   gunicorn --workers=3 runner:app --daemon
7. Open <<VM-IP-addresses>> in browser and check if it's working
   For the Dec 2022 submission these IP addresses are:
   vm1-az1: *40.82.200.169*
   vm2-az2: *20.70.235.122*
   main-static-ip: *20.213.168.157. This should receive internet traffic and load balance them amongst the two VMs.*

Infrastructure (Azure)

Our instructor will be given access to an Azure account that has already been setup with the following testing setup:

AzureAnonymouse-dec22-web-rg

- 1 x captured Image of the latest version of AzureAnonymous
- 1 x SSH public key (private key can be provided on request)
- 1 x Virtual Network with:
  a. 1 x Network Security Group configured for web servers
  b. 1 public subnet (10.0.1.0/24)
  c. 1 x Load Balancer, connected to:
     i. 2 x VMs (identical), each with their own static IP addresses

cotiss-cosmos-mongo-rg

- 1 x Azure Cosmos DB for MongoDB account

Account Access (SSH, private key can be provided on request)

Database login as: azureuser
Password: NIL

If this is not something that you have access to, you will need to sign up for your own Azure Portal access and setup the infrastructure and technology stack detailed in the Network Architecture Design portion of this report. A full step-by-step guide can be found in the SysAdmin User Guide: https://github.com/csidon/AzAnonMontygo

Application Functionality Requirements for End Users

- Access the application from any browser
- Submit their feedback anonymously
- Randomly view other reviews whenever they refresh the page/hit a button

Functionality Requirements for Company

- Be able to view all feedback provided

Non-Functional Requirements

- Needs to appear trustworthy and professional
- Kept stylistically consistent with Cotiss' company branding so that it's recognizable and trusted
- Minimalistic design with simple design elements and layout
- Be as lightweight as possible*

* Note: The application produced is an MVP and has yet to be optimized for minimizing calls to the database.

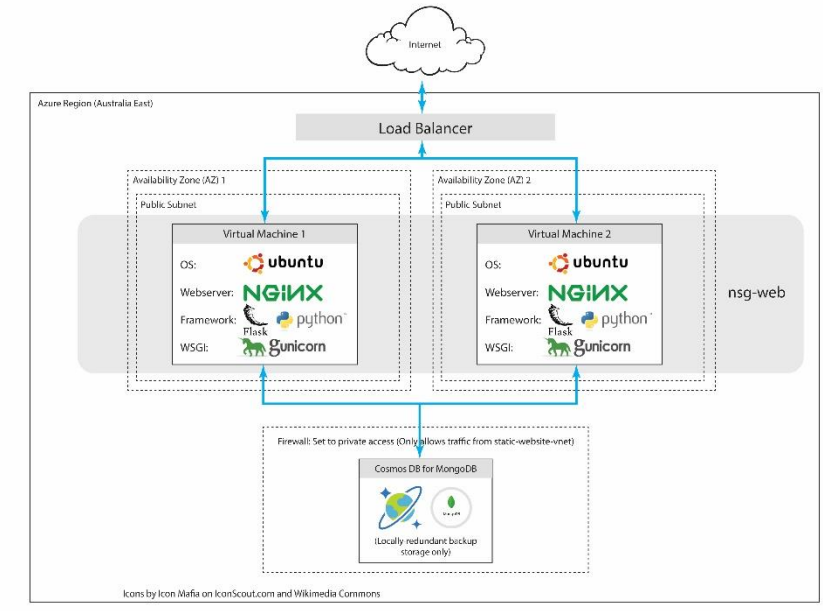
Network Architecture Design & Costing

This report does not explore the costs involved in scaling up or scaling out for traffic and focuses largely on the costs and availability of the network design.

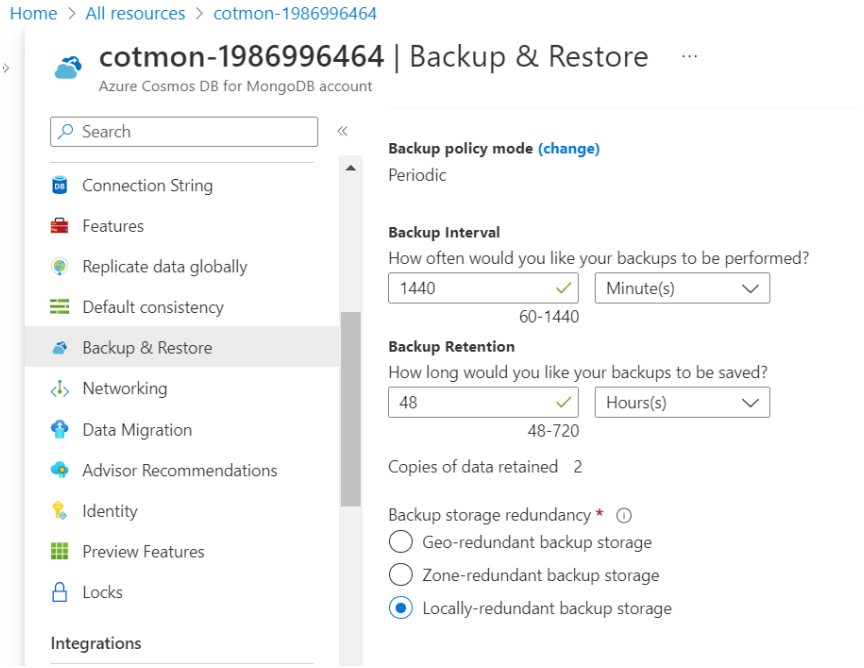
Designing for Minimum costs

Costing Estimate at $68.14/month pay-as-you-go (PAYG):
https://azure.com/e/c754f91991e74b8c8c19c453927e0d5c
Note: If this is a low volume application and is shifted to serverless Database Operations, there is the potential for database costs to drop significantly.

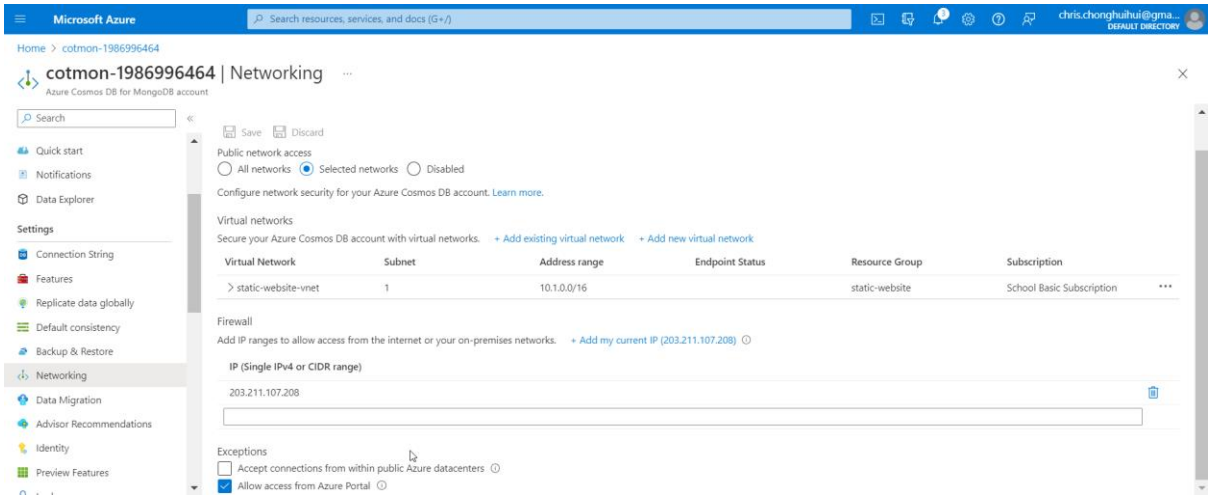
We like networks to be able to self-heal and the brief specifies for us to explore a highly available setup, however, realistically, this application probably doesn't require a high SLA (if any) at the moment. The Azure account that you've been given access to is set up with the most basic of networks for relative high availability, and can be seen as follows:



The term “relative availability” is used because there are 2 VMs in 2 Availability Zones (AZs) that are load balanced, so there should always be at least one webserver up, but with only one database available. The reason for this decision is due to the type of data that is being collected - The data is not highly vital, doesn’t contain any Personal Identifiable Data (PID), and the current brief does not specify any need to keep the data for long term evaluation and analysis. Because of this, it is reasonable to use the lowest backup and restore settings for CosmosDB to optimise costs. The database can be queried on a regular basis and cached locally in the VM.



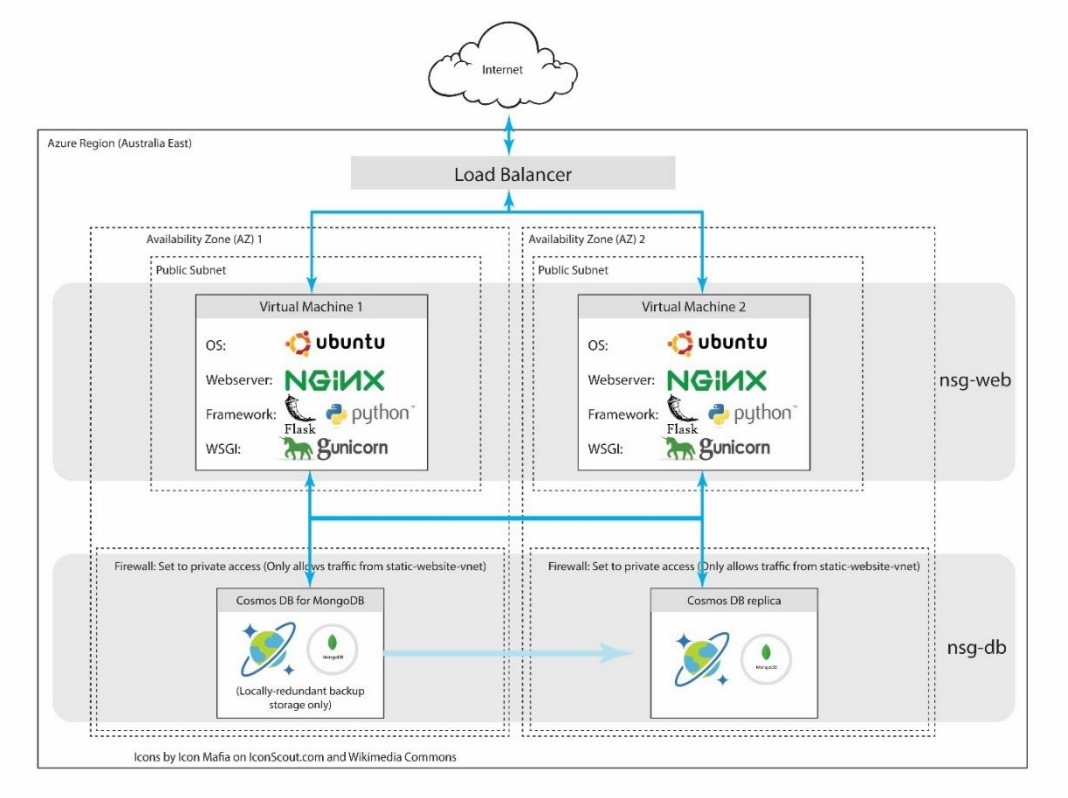
Even though the database doesn’t hold any vital data, it’s still good practice to protect it from public access since that ensures that i.e. the data is harder to corrupt. Therefore the Public Network Access should be limited only to specific networks.



Designing for High Availability

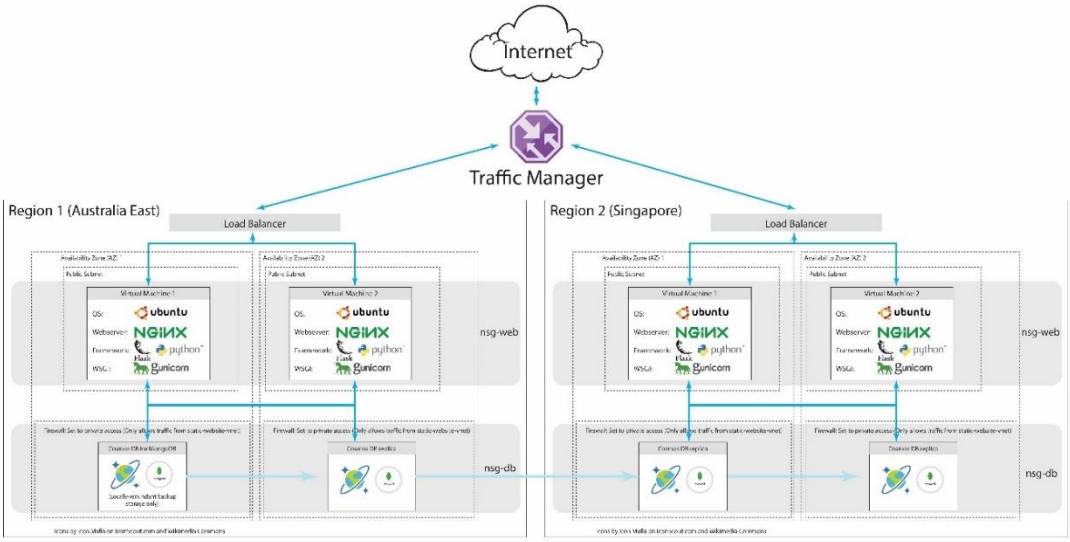
Costing Estimate at $116.04/month pay-as-you-go (PAYG):
https://azure.com/e/f8132a011f0b44e48a6ef8605de7faee

If there is a need for the applications database to be constantly available, then CosmosDB can be set up to allow regular replication within a region. This does almost double the cost though, as can be seen in the costing estimate above.



Designing for Maximum Availability

Costing Estimate at $216.65/month pay-as-you-go (PAYG):
https://azure.com/e/ace400f98a01405dacf4c99b06430a3b



If for some reason a requirement arises over time where there's a need for high availability across regions, the cost again doubles. However, if this application reaches that level of need, it is likely to have far higher traffic and usage. In that case, the number of VMs per AZ would first be increased, so this costing would not be very accurate.

Things I've learnt

I'm still a baby in networking, but this exercise has helped heaps in understanding/learning:

- Heaps more about the fundamentals of networking and application development
- The basics of Azure's (and other cloud providers) offerings and capability
- A glimpse into how Azure's PaaS and SaaS services can help companies (without their own networking team) save time and money
- How to provision VMs and set them up to be webserver ready
- How to capture images for easy replication
- How to setup Cosmos DB, get it to talk to the webserver, and limit public access for security measures
- Practicing using Github and creating readme files (please let me know if they were easy or difficult to follow!). It would have been good if this were pair work, however given the duration of this course and the need for all parties to be familiar with the system, this does end up posing a bit of a challenge.

I've also learnt that huge costs can be saved by shutting development VMs down overnight when not in use. It is easy to accidentally consume a lot of resources just by leaving things running overnight and getting a bill-shock the next day. It didn't help that I worked quite late on some days, so I couldn't schedule the virtual machines to shut down at, say, 10pm or midnight.

Interestingly enough, I found that one of the hardest things to do was to stick to the bare basics. Most Azure-related tutorials jumped immediately to serverless app development, and I had to extrapolate some information from tutorials that were AWS or Google Cloud related.

I also have a tendency to complicate and build in fancy things, so this was a good project to help build discipline, not allow scope creep, and to make sure that I took responsibility for using resources in an efficient manner.

I haven't focused heaps on application security given the scope of this project, and would like to look into it at some point, starting with first exploring and implementing the recommendations. First, however, I hope to keep doing small version updates and maybe even move this to become a serverless application, and would eventually like to learn how to provision VMs automagically (through scripting), and (wishlist) setup a deployment pipeline with automated checks and testing.

Helpful references

(For future me)

Getting MongoDB to speak with Flask:
https://learn.microsoft.com/en-us/azure/cosmos-db/mongodb/quickstart-python?tabs=azure-powershell%2Cvenv-windows%2Cdotenv
https://www.w3schools.com/python/python_mongodb_create_db.asp

Refresher - How HTML queries work:
https://www.youtube.com/watch?v=TCEgdiN0A8s

How to write a Flask scripts to make the calls linking index.html to the database:
https://www.linkedin.com/pulse/integrate-mongodb-flask-creating-simple-student-data-form-phatate
https://www.linkedin.com/pulse/integrate-python-flask-mongodb-abhijeet-karmakar