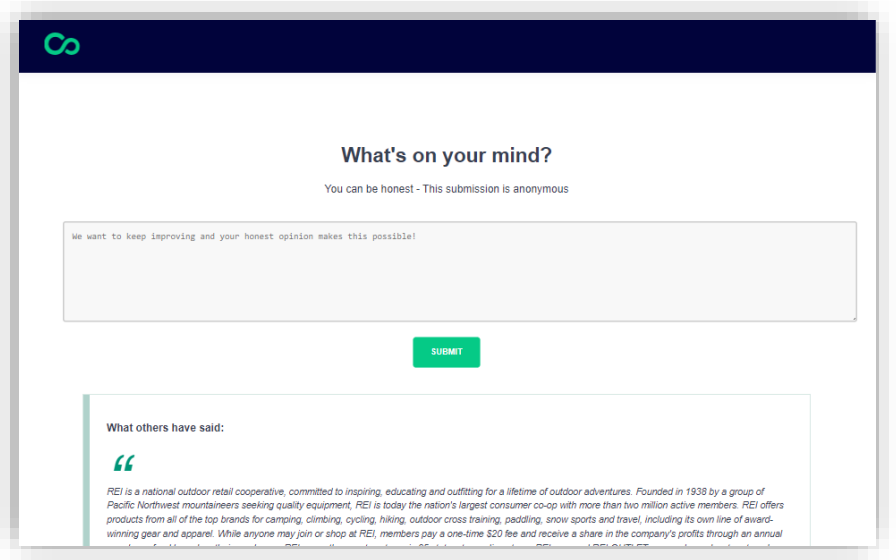
Summer of Azure 2022-2023



AzureAnonymous

Application Process and Installation Guide

A Real-World Project Created For:

COTISS

By Chris Chong

Table of Contents

Note: This document does not cover the configuration of the overall Azure Virtual Network and load balancing solution, and focuses on providing an overview of the application development process as well as in-depth information on how to get the application up and running via the Github repo.

Overview

AzureAnonymous is a simple application created with Cotiss in mind. Cotiss is built specifically to help organisations be more effective in how they buy & supply goods and services. They have requested a simple website where employees can anonymously submit feedback.

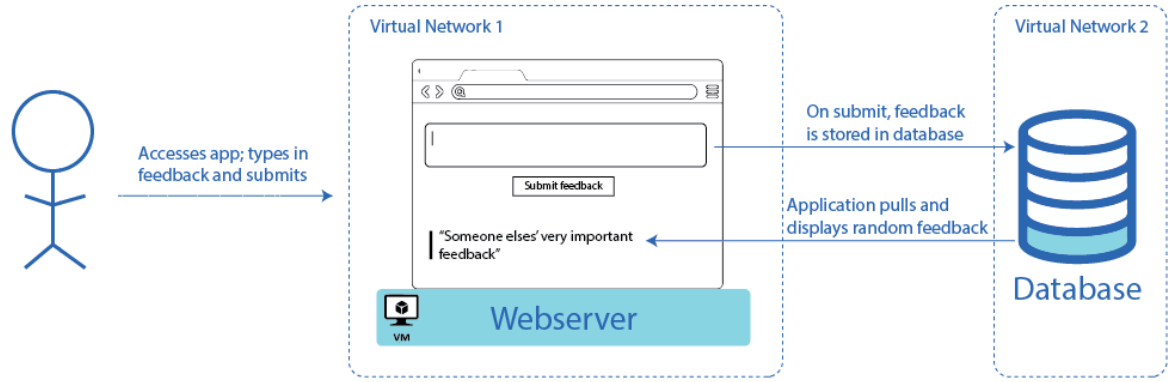
The UX design requested is simple; a random piece of feedback displayed on each page load, and a box at the bottom of the page with a submit button to add new feedback to the feedback bank.

The application itself was not the main goal of this project. The focus lies in the implementation of a high availability application on Azure.

Azure has many PaaS and SaaS solutions that can help eliminate the need to run this application on a virtual machine. However, given that we only had 2 weeks to create this solution, and since this is the first time that I have created a web app from the ground up, I chose to create this using the fundamental basics of web application architecture.

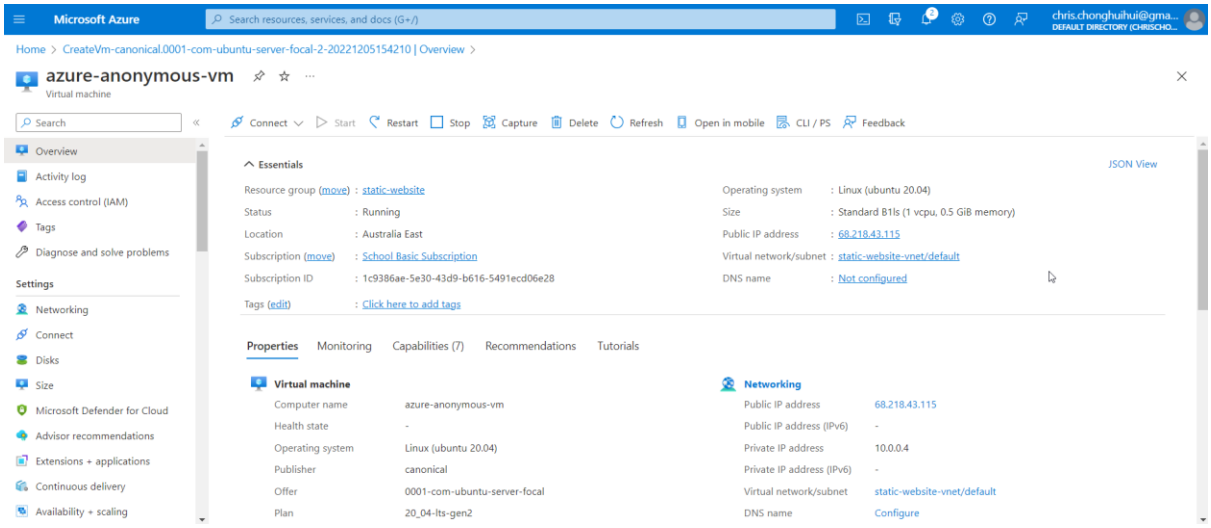
Approach

The most basic setup and interaction that I could identify is as shown:

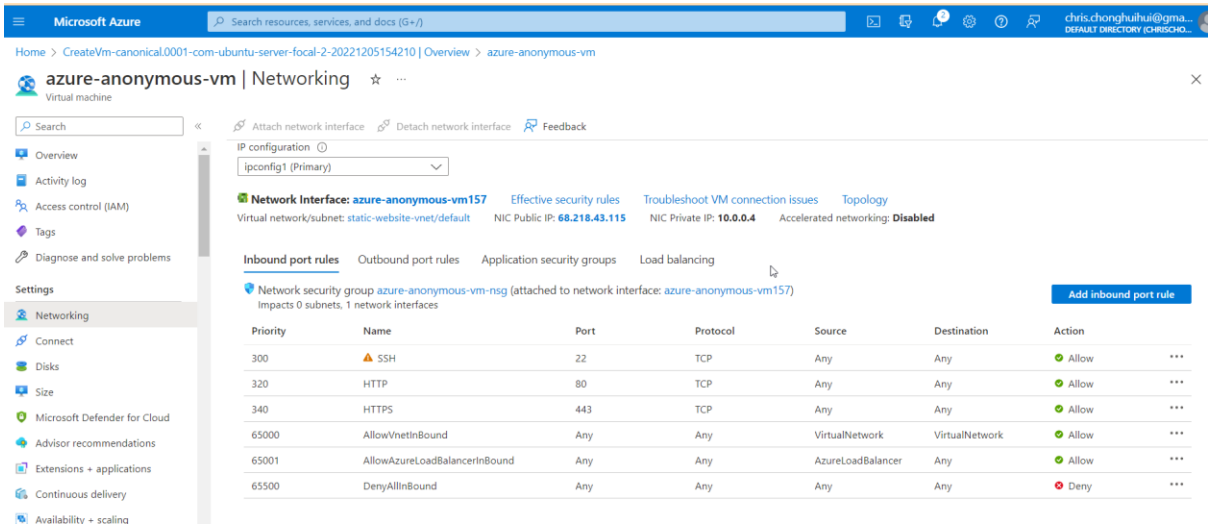


Stage 1: Setting up and load balancing webservers in Azure

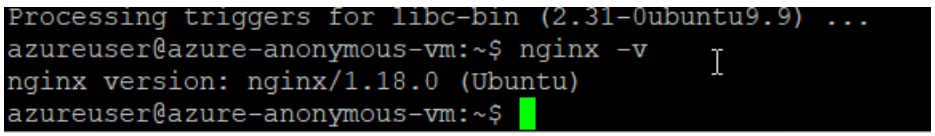
My first thought was to learn how to set up a webserver on Azure, learn how to capture and image, and load balance it. I have prior experience in using NGINX as a reverse proxy on AWS and decided to use that for my solution.



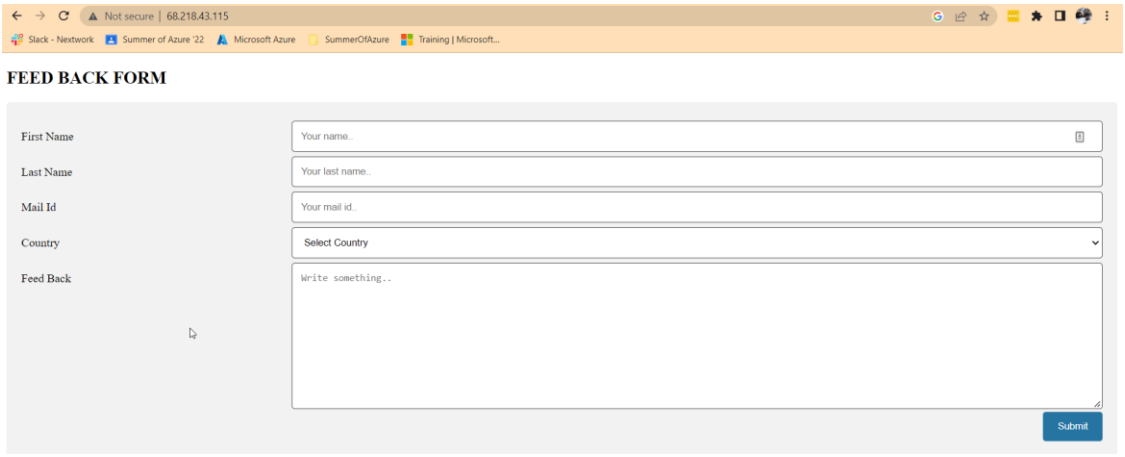
Screenshot 1: Creating a tiny Ubuntu VM in Azure



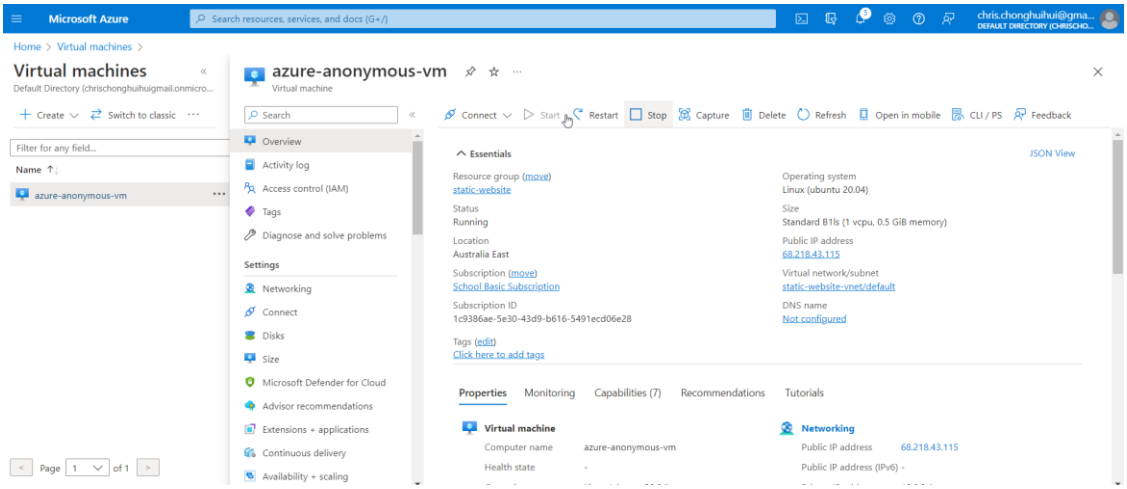
Screenshot 2: Opening ports 80 and 443 (which I can close off later if things change) since website doesn't have any hugely secure data or keep any PIM



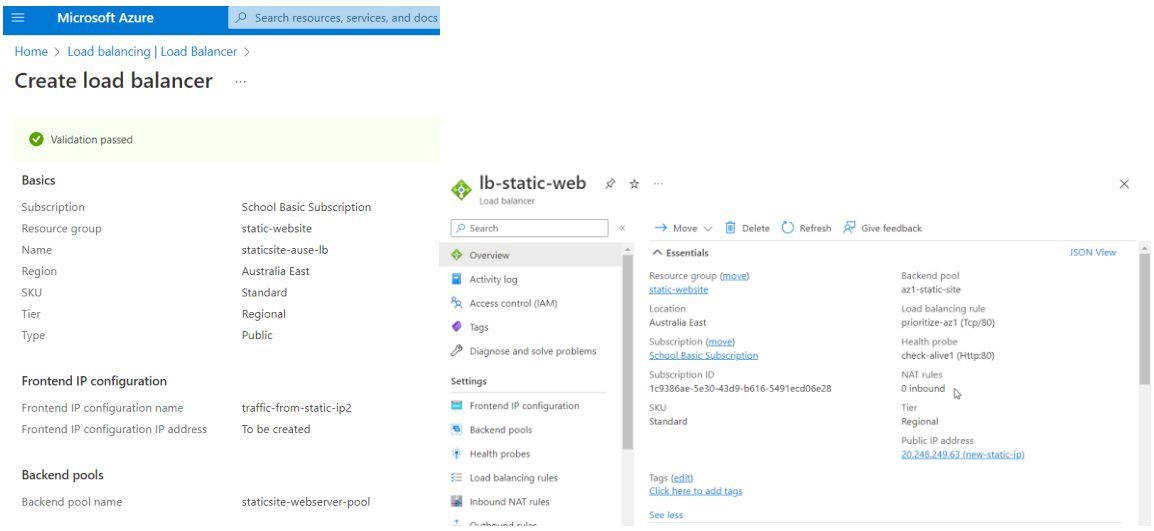
Screenshot 3 & 4: Updating packages, installing Nginx, then checking to make sure that the implementation was successful



Screenshot 5: Creating a test HTML feedback form using code from the following article: https://www.c-sharpcorner.com/article/creating-a-feedback-form-using-html/



Screenshot 6: Capturing VM image for easy replication

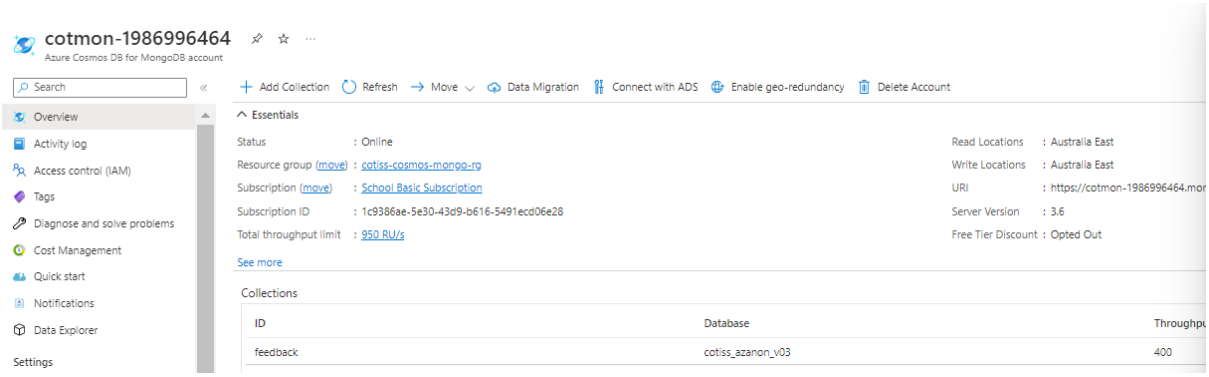


Screenshot 7: Creating a load balancer that evenly directs traffic to VMs in the pool

Stage 2: Creating a database

Once I was sure that the webserver was running as expected, I created a database in Azure using CosmosDB. I have heard of MongoDB being a non-relational database, and chose this because it would allow for flexibility in analysing the feedback, if needed.

The Quickstart Guide for Azure Cosmos DB for MongoDB for Python was really good as a general overview. I installed MongoDB on my local machine and CosmosDB on Azure, then used this guide to help me learn how to connect the two:
https://learn.microsoft.com/en-us/azure/cosmos-db/mongodb/quickstart-python?tabs=azure-powershell%2Cvenv-windows%2Cdotenv



Screenshot 8: Setting up CosmosDB database

Unfortunately, I forgot to take screenshots of the code here, but it involved something similar to the following:

```python
import os
import sys
from random import randint
from pprint import pprint

import pymongo

import bson
from dotenv import load_dotenv

from flask import Flask, request, render_template, jsonify
import random



load_dotenv(verbose=True)
CONNECTION_STRING = "mongodb://cotmon-1986996464:tE7Tlv9V8CMhM7fxoo2lvG9QMkqFH0BmLBuW53i2qLRXv2A0mHcCoak


DB_NAME = "cotiss_azanon_v03"
COLLECTION_NAME = "feedback"
client = pymongo.MongoClient(CONNECTION_STRING)

# Create database/collection if it doesn't exist
db = client[DB_NAME]
collection = db[COLLECTION_NAME]
```

Screenshot 9: Getting Python script to talk to CosmosDB database

When I was sure that the calls were actually creating entries in my Azure database (not locally!), I set the next goal of having my index.html form submissions make calls those calls to the database directly. I have a tiny bit of experience in Python, and remember being told that Flask can be used to develop web apps using Python and decided to use that as my web framework.

Stage 3: Getting index.html form submits to talk to database

```
21  COLLECTION_NAME = "feedback"
22  client = pymongo.MongoClient(CONNECTION_STRING)
23
24  # Create database/collection if it doesn't exist
25  db = client[DB_NAME]
26  collection = db[COLLECTION_NAME]
27
28  # Check the number of documents (records) in database
29  def countdocs():
30      num_docs = collection.count_documents({})
31      print("You have >>>", num_docs, "<<< records in your database")
32      # Set the id for the new feedback
33      return num_docs
34
35  # Gets and returns a piece of random feedback
36  def get_rand_feedback():
37      rand_feedback_index = 0
38      rand_feedback = ""
39      rand_feedback_index = random.randint(1, countdocs())
40      print(rand_feedback_index, "<<<<<< This is the random record I'm trying to pull")
41      rand_record = list(collection.find({"_id": rand_feedback_index}))[0]
42      rand_feedback = rand_record['feedback']
43      return rand_feedback
44
45  # For debugging: print("Upserted document with _id {}\n".format(result.upserted_id))
46  print("Databases available are: ", client.list_database_names())
47  print("Collections available are: ", db.list_collection_names())
48  # for records in collection.find():
49  #     print(records)
50  print("CONNECTION STRING IN RUNNER IS ", CONNECTION_STRING, "<<<<<<<<\n")
51
52
53  # define the flask app
54  app = Flask(__name__)
55  app.config["MONGO_URI"] = CONNECTION_STRING
56
57  # define the home page route
58  @app.route('/')
59  def hello_world():
60      clean_feedback = ""
61      clean_feedback = get_rand_feedback()
62      return render_template("index.html", feedback=clean_feedback)
63
```

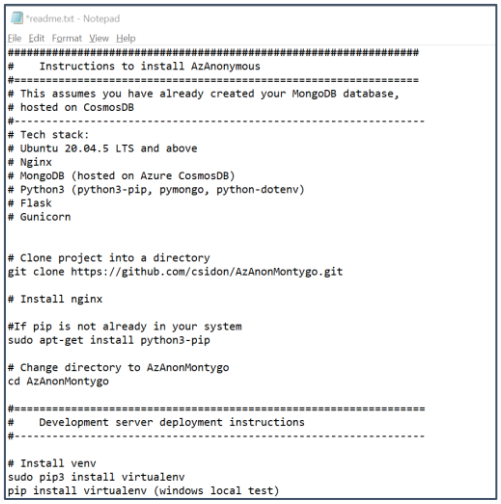
Screenshot 10: Getting HTML form to push and pull data from CosmosDB

I went back to the very basics to refresh myself on how HTML forms/queries work. This video was one that I found super helpful: https://www.youtube.com/watch?v=TCEgdiN0A8s

These two articles helped heaps in providing me with a general idea of how to write the Flask script to make the calls linking my index.html page 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

Stage 4: Installing application on VM

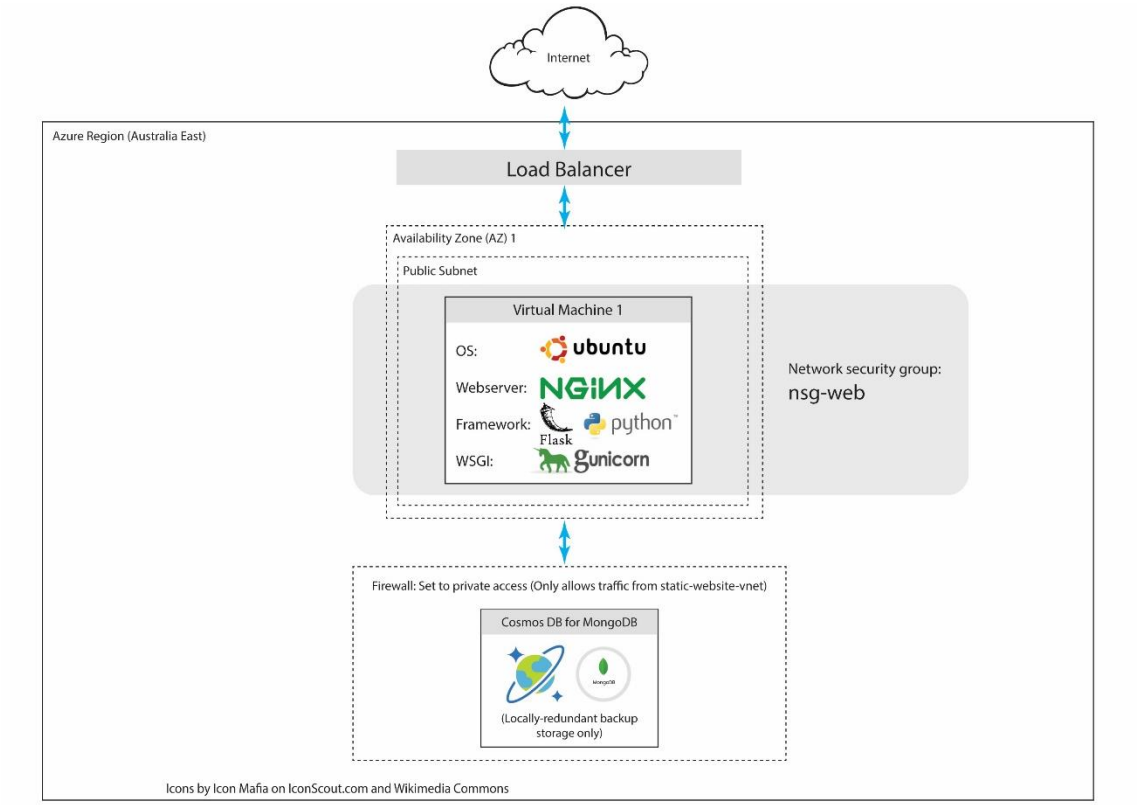
To reduce the complexity of i.e. ports/connectivity etc, Stages 2 and 3 were done on my local machine. The challenge now was to port this over to my Azure VM, so that I could capture an image for easy setup. I had been "checking" my commits into my repository on Github, so this made the process of sharing application information much easier. I found that I went back and forth on my local (Windows) machine and VM (Ubuntu) quite a bit to try and figure out what worked. In the process, I also created a simple readme.txt file to help me remember the varying commands, but also to help make this repo useful to "future me" and other people might find this project interesting/useful.



```
*readme.txt - Notepad
File  Edit  Format  View  Help
#####################################################################
#     Instructions to install AzAnonymous
#====================================================================
# This assumes you have already created your MongoDB database,
# hosted on CosmosDB
#--------------------------------------------------------------------
# Tech stack:
# Ubuntu 20.04.5 LTS and above
# Nginx
# MongoDB (hosted on Azure CosmosDB)
# Python3 (python3-pip, pymongo, python-dotenv)
# Flask
# Gunicorn


# Clone project into a directory
git clone https://github.com/csidon/AzAnonMontygo.git

# Install nginx

#If pip is not already in your system
sudo apt-get install python3-pip

# Change directory to AzAnonMontygo
cd AzAnonMontygo

#====================================================================
#    Development server deployment instructions
#--------------------------------------------------------------------

# Install venv
sudo pip3 install virtualenv
pip install virtualenv (windows local test)
```

Screenshot 11: Documenting commands to install/deploy application

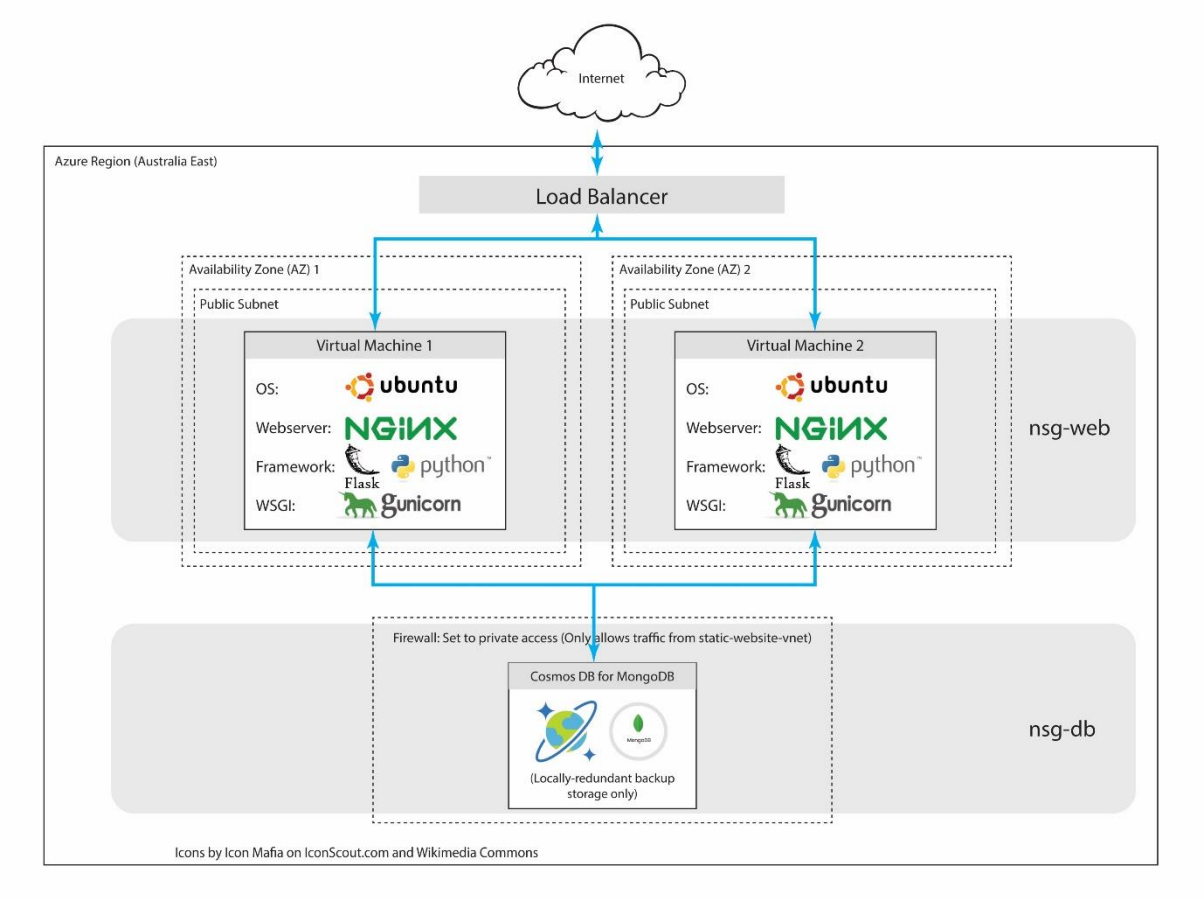
At this stage, I had the following basic setup:



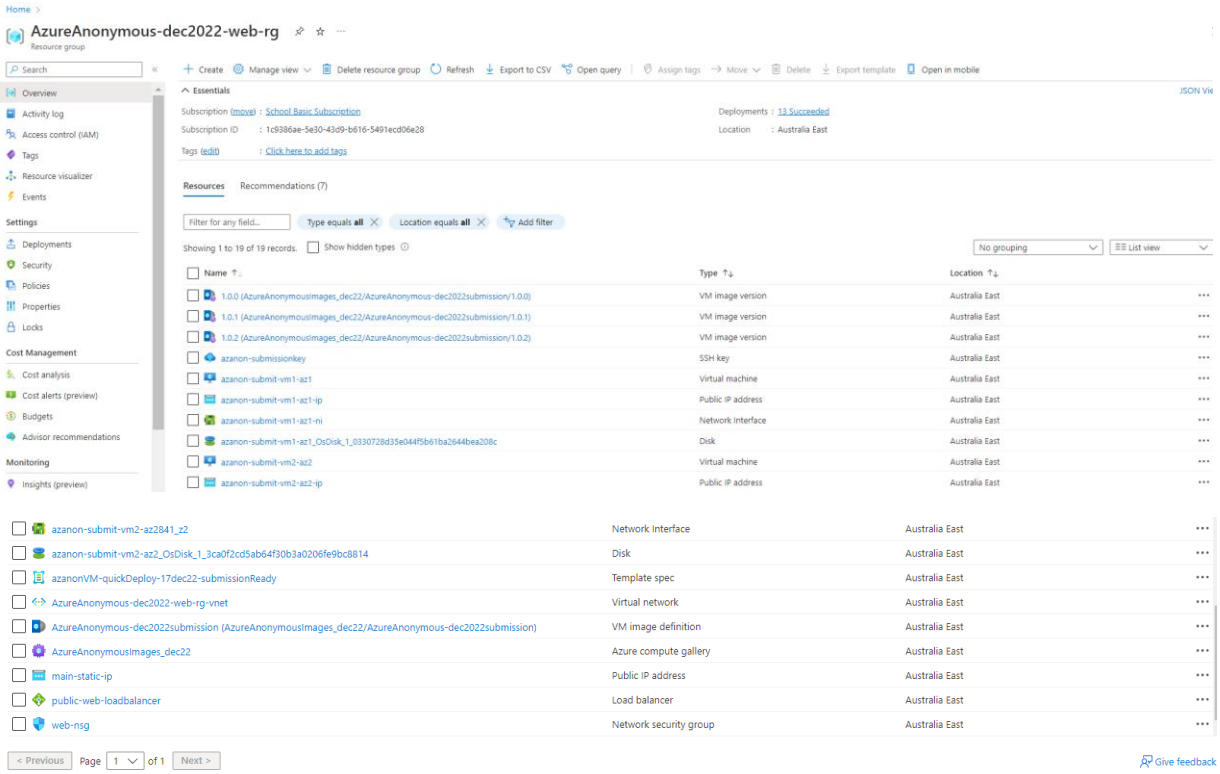
Screenshot 12: Basic application architecture

Stage 5: Testing for High Availability

Once the application was up and running, I captured an image of the VM and looked into the simplest “high availability” solution for this application. My intention was to submit the following network design, with captured VM images that would allow easy deployment of more resilient architectures.



Screenshot 13: Simplest High Availability application architecture



Screenshot 14: Resources required for the implementation of Minimum Cost application architecture (Screenshot 13)

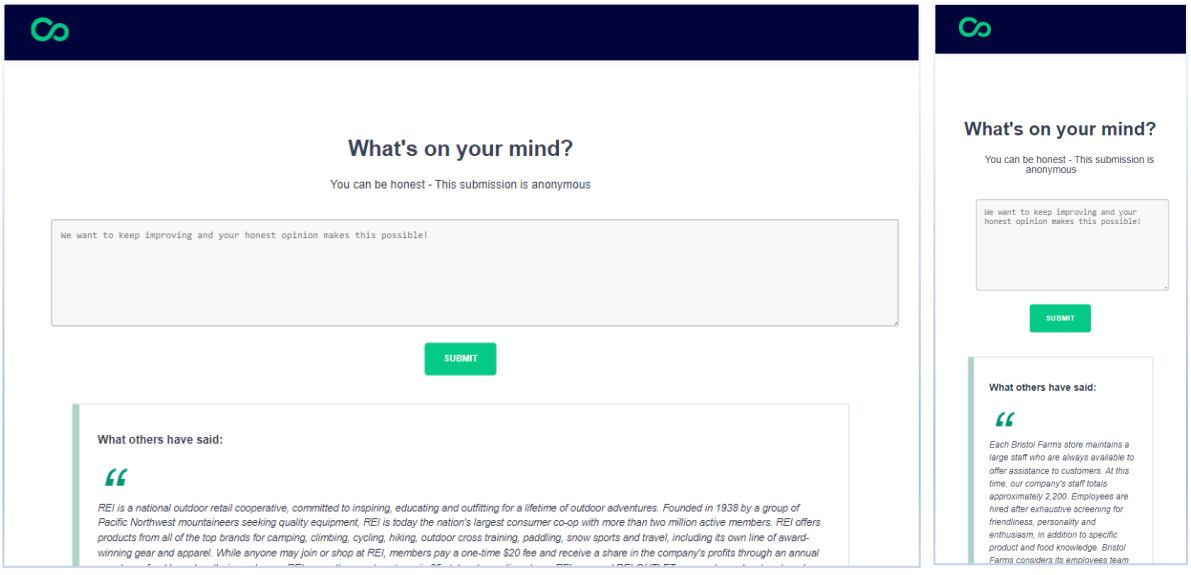
Stage 6: Making it pretty

While this project's focus isn't on the UI/UX of the application, I felt that I needed to (at the very least) make it look like it was an application for Cotiss. With this in mind, I spent about a day or so making it look pretty.

runner.py | index.html

```
136
137   .quote-text {
138     font-size:14px;
139     font-style: italic;
140     line-height:1.5;
141   }
142
143   .quote-text--intro {
144     font-weight: bold;
145     font-style:normal;
146     text-align: left;
147   }
148
149   .quote-text::before {
150     content: "\201c";
151     font-size:100px;
152     color:#009578;
153     display: block;
154     margin-bottom: -70px;
155     margin-top:-20px;
156   }
157   </style>
158   </head>
159
160   <!-- HTML PORTION OF THE FILE -->
161
162   <header class="k1-header">
163     <nav class="k1-navbar content-wrapper">
164       <a href="/?hsLang=en">
165         <img src="https://www.cotiss.com/hubfs/Fill%2013.svg" alt="Cotiss" style="max-width: 100%; height: auto;" loading="" class="k1-navbar__logo">
166       </a>
167     </nav>
168   </header>
169
170   <body>
171     <div class="container">
172       <center>
173         <h1>What's on your mind?</h1>
174
175         <label for="feedback">You can be honest - This submission is anonymous </label>
176       </center>
177     </div>
178     <div class="container">
179       <center>
```

Screenshot 15: Snippet of HTML + internal CSS styling code



Screenshot 16: Final desktop and mobile UI of the feedback form

The rest of the time was spent on tidying up documentation and report writing.

Technical Specifications & Installation

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

For Users with Azure Account Pre-Setup Access

This account access will only be available for a limited time period, and it is likely that you won't have access to this. 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 section. A full step-by-step guide on how to setup an Azure VM and install the application can be found in the next section.

If you have been given access to an Azure account (instructor access until 1 Mar 2023 only):

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.*

Azure Resources

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

Step-by-step guide to setting up AzureAnonymous VMs

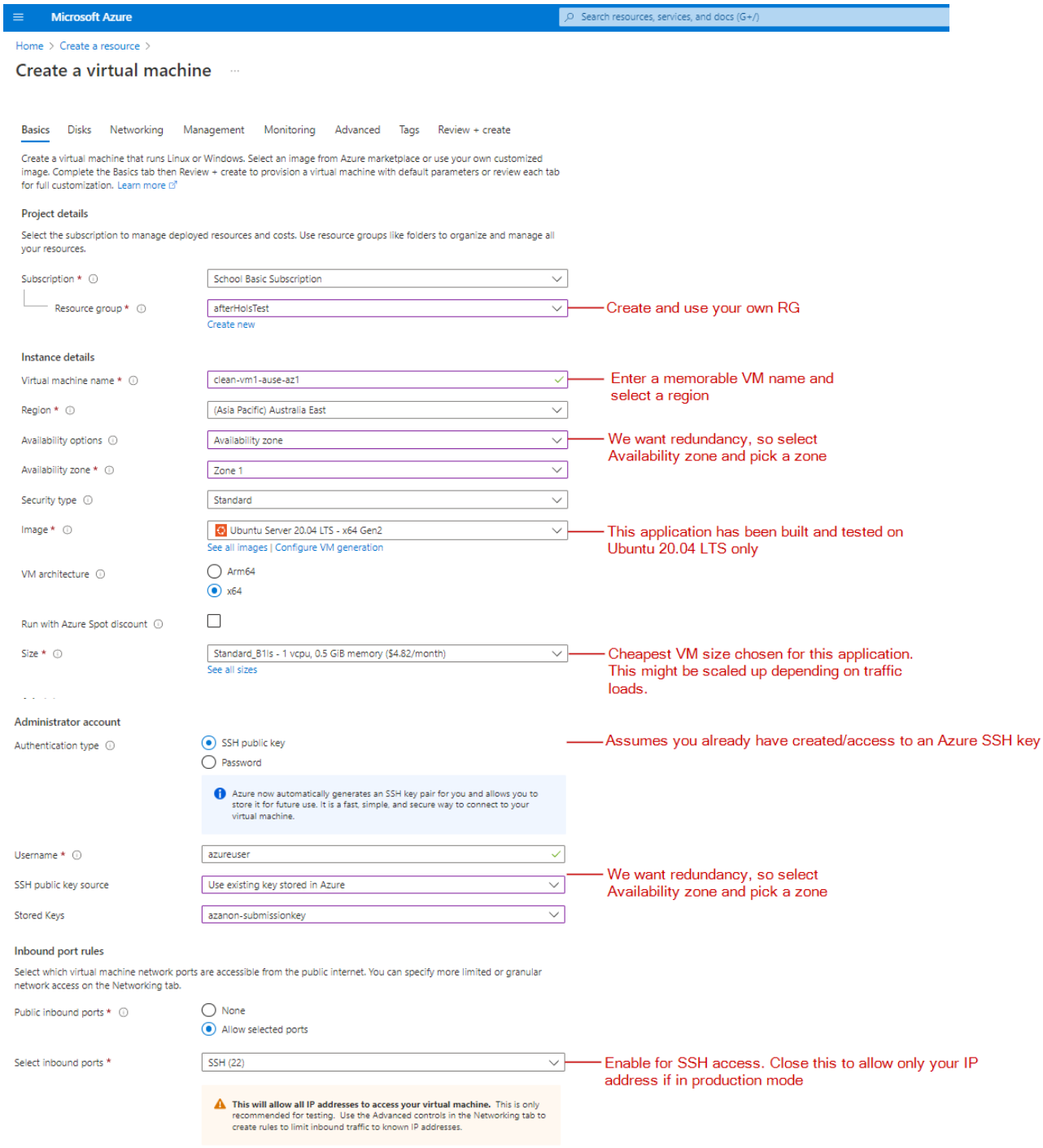
*For Users **without** CSidon Azure Account Access*

This section has been created for developers and sysadmins (and future me) that don't have access to the Azure Account with a VM image already set up the application ready to go, and only have access to the Github repository: https://github.com/csidon/AzAnonMontygo

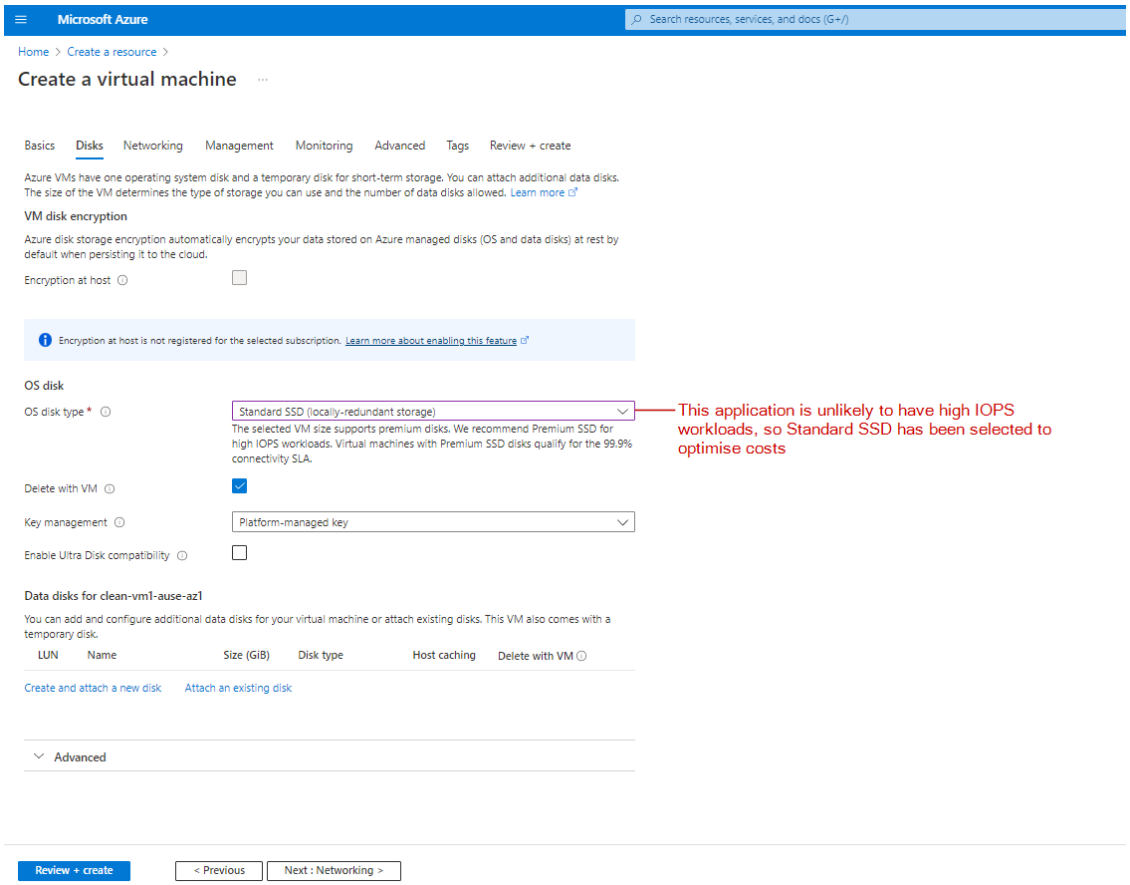
Create an Azure Virtual Machine

Create a virtual machine with the following settings:

Basics

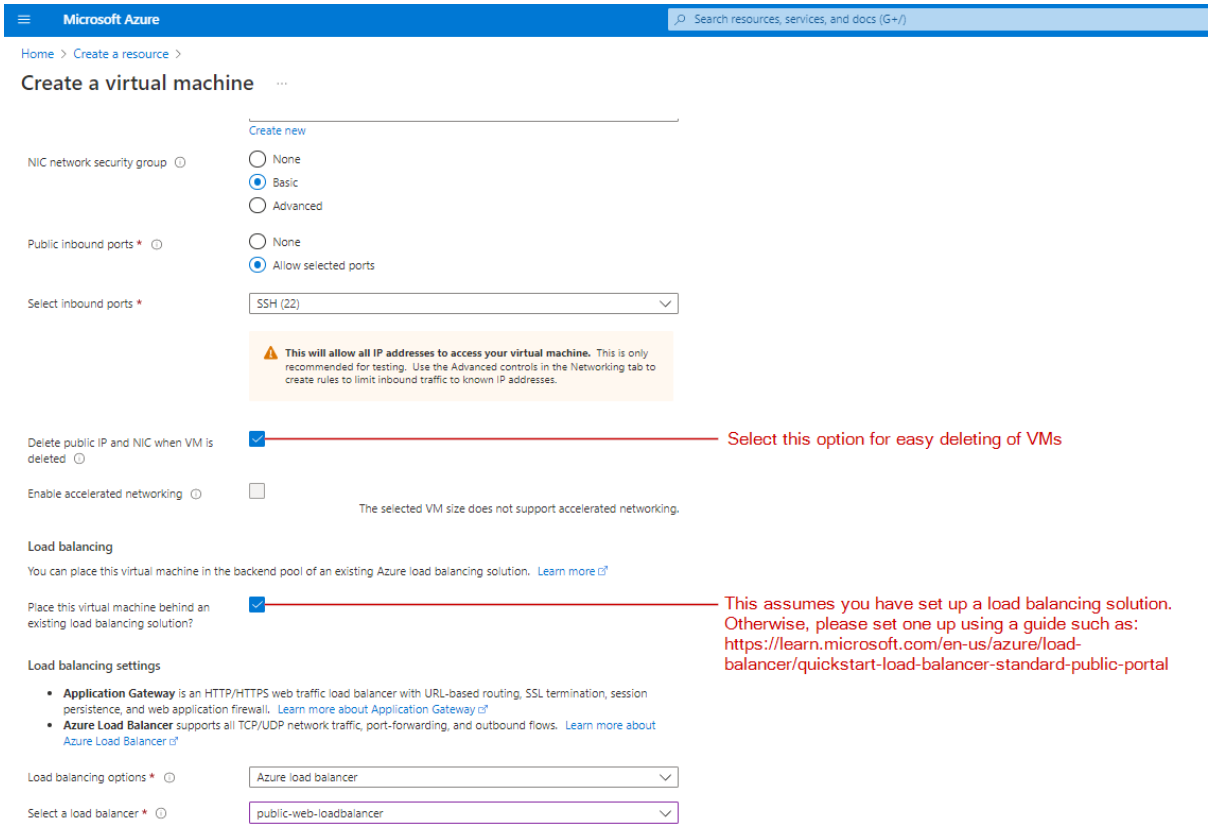


Disks



This application is unlikely to have high IOPS workloads, so Standard SSD has been selected to optimise costs

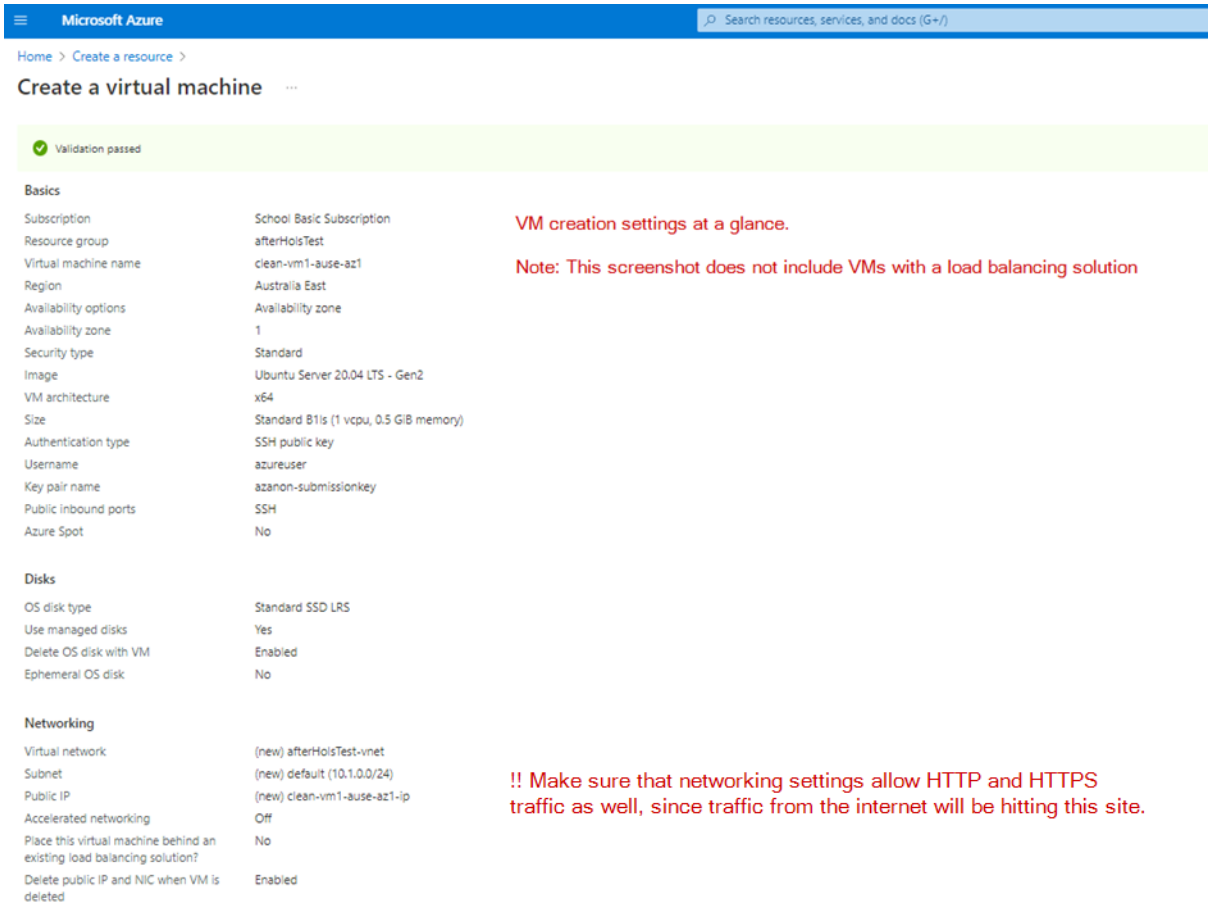
Networking



Select this option for easy deleting of VMs

This assumes you have set up a load balancing solution. Otherwise, please set one up using a guide such as: https://learn.microsoft.com/en-us/azure/load-balancer/quickstart-load-balancer-standard-public-portal

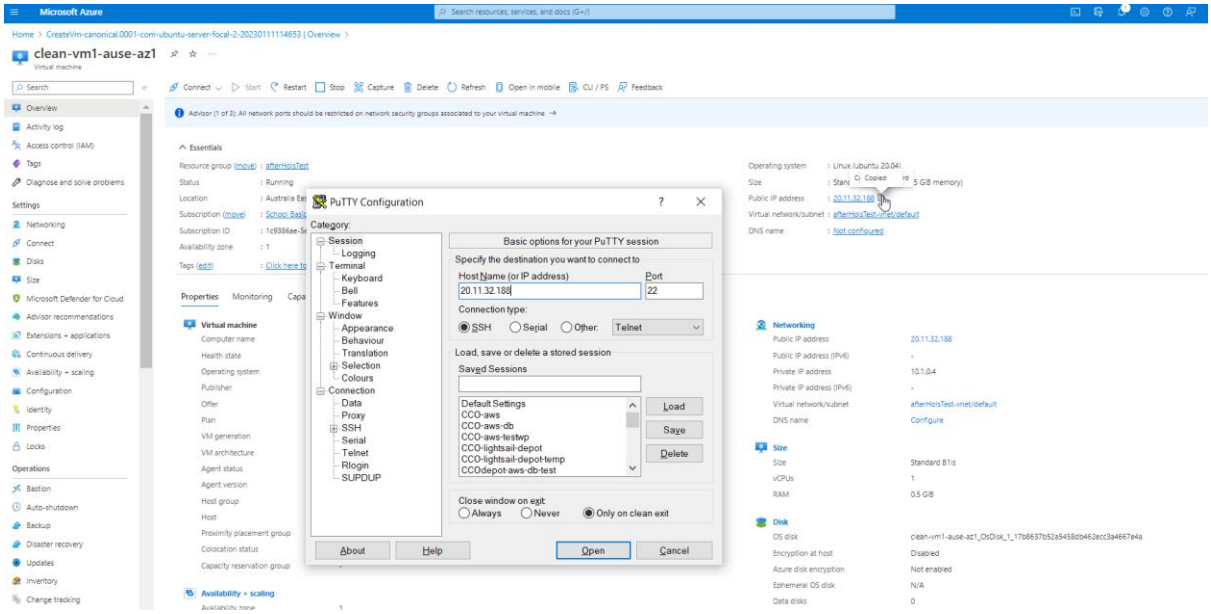
Settings Overview

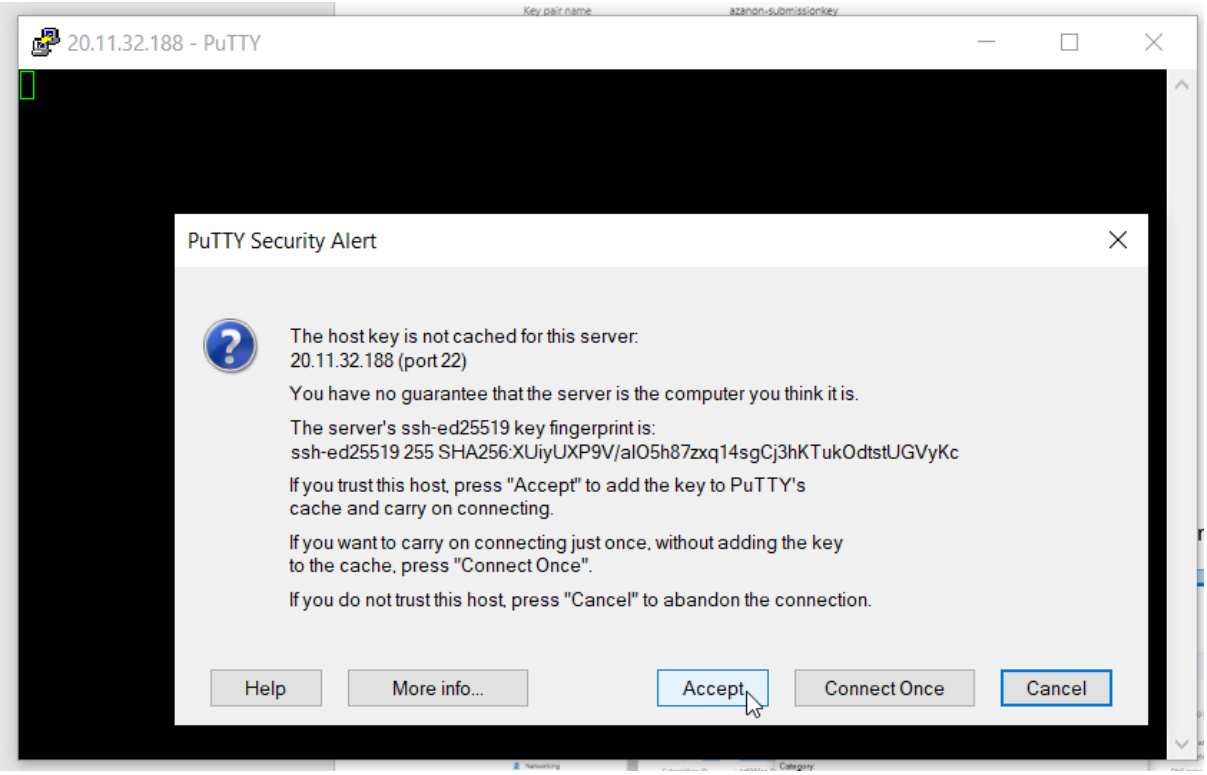


Configuring your VM

SSH into your VM

This assumes that you know how to use Putty to SSH into another machine





Login to VM and update/install packages

Login with your username (in this case, azureuser), then

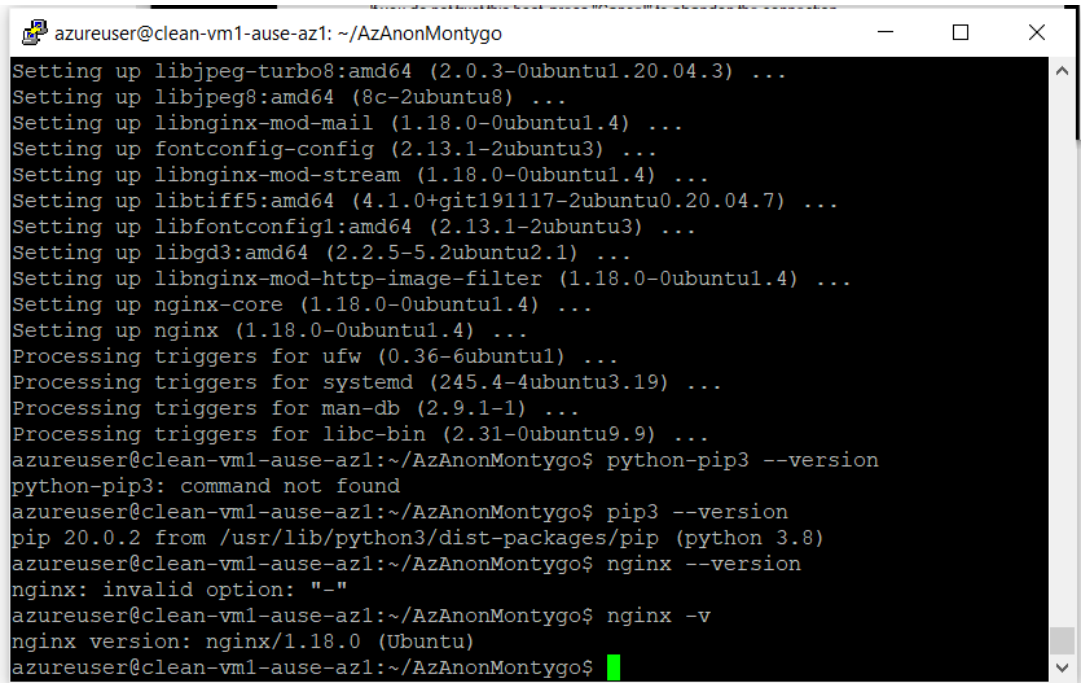
>> sudo apt-get update

Install python, nginx and venv

>> sudo apt-get install python-pip3 -y

>> sudo apt-get install nginx -y

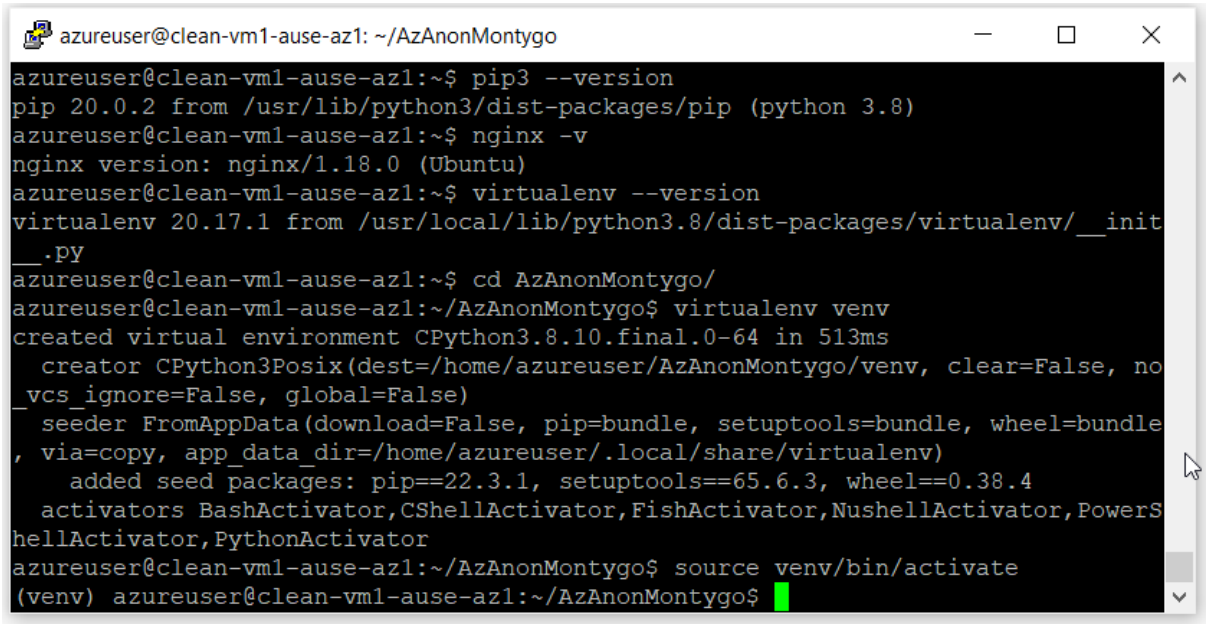
>> sudo pip3 install virtualenv -y



Clone project, enter directory, create venv

Clone the git project and cd into it, then create a virtual environment and activate it

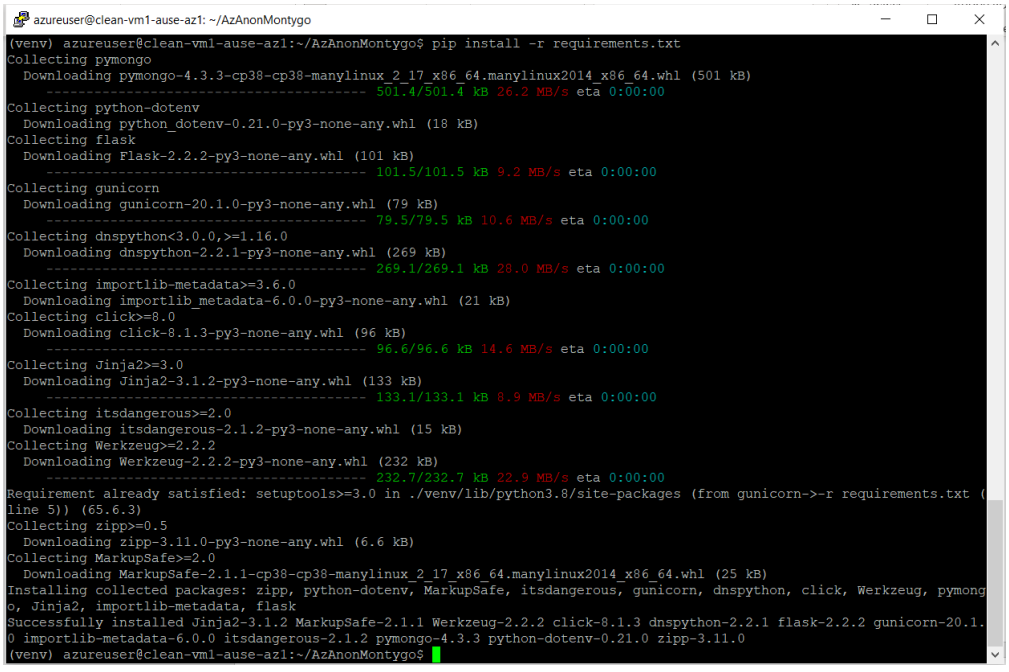
>> git clone https://github.com/csidon/AzAnonMontygo.git
>> cd AzAnonMontygo
>> virtualenv venv
>> source venv/bin/activate



Note: Use this article if you're running into venv installation issues
https://techoverflow.net/2022/02/03/how-to-fix-tox-attributeerror-module-virtualenv-create-via_global_ref-builtin-cpython-mac_os-has-no-attribute-cpython2macosarmframework/

Install requirements into venv

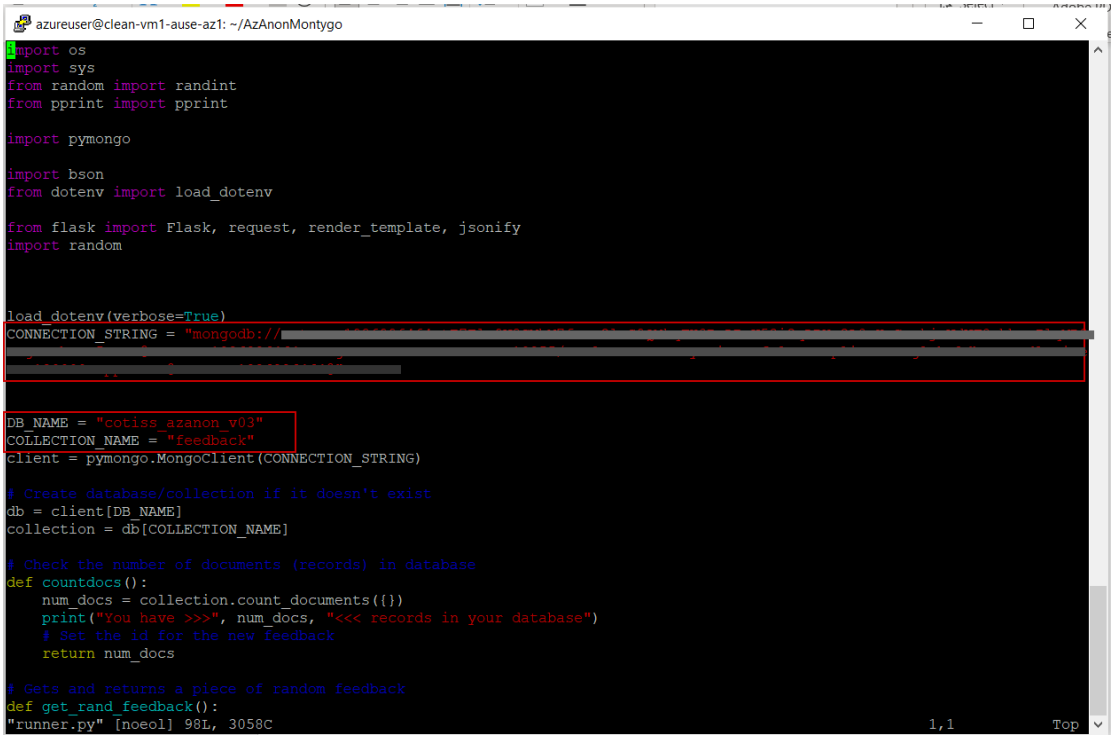
>> pip3 install -r requirements.txt



Amend DB details in runner.py

Open runner.py and replace CONNECTION_STRING, DB_NAME and COLLECTION_NAME with your database details, then run it

(This assumes that you have already created your MongoDB database, hosted on CosmosDB)



Test run the application

>> export FLASK_APP=runner.py

>>flask run -h 0.0.0.0

Follow the final steps in readme.txt to deploy for development server or production server.

Note: If you decide to stop/restart the VM, you will need to follow the steps in the For Users with Azure Account Pre-Setup Access section of this guide to get the application up and running again. Ideally this should be a script that runs automagically each time the VM is started up.

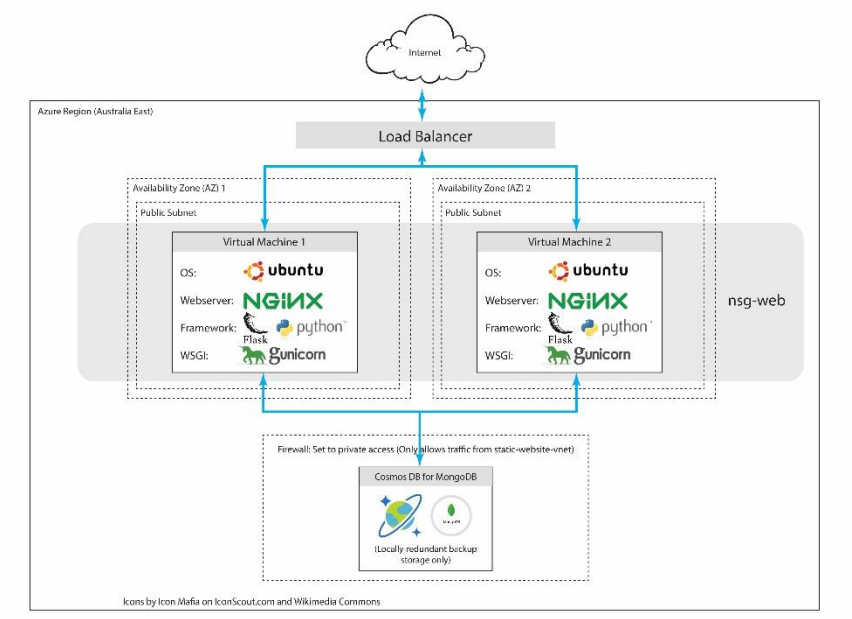
Network Architecture Design & Costing

This documentation 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.

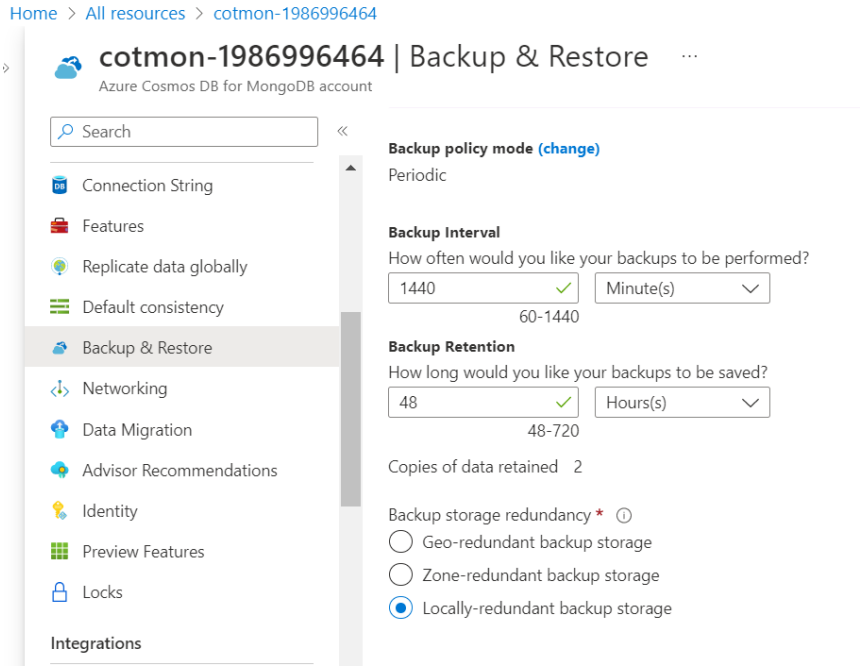
Designing for Minimum costs

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.

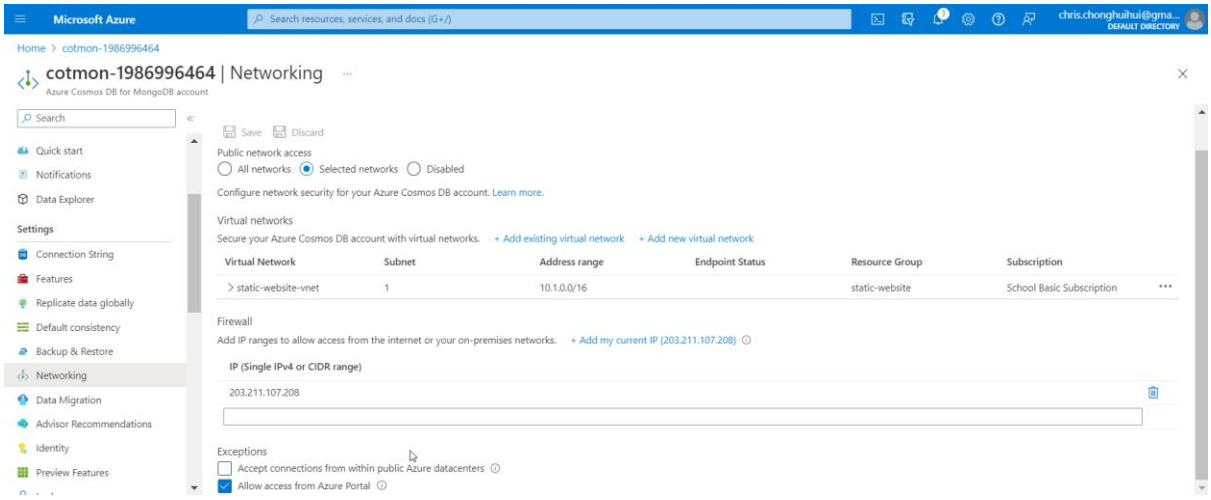
We like networks to be able to self-heal and the brief specifies for us to explore a highly available setup, however, realistically, this application probably doesn't require a high SLA (if any) at the moment. The Azure account that you've been given access to is set up with the most basic of networks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



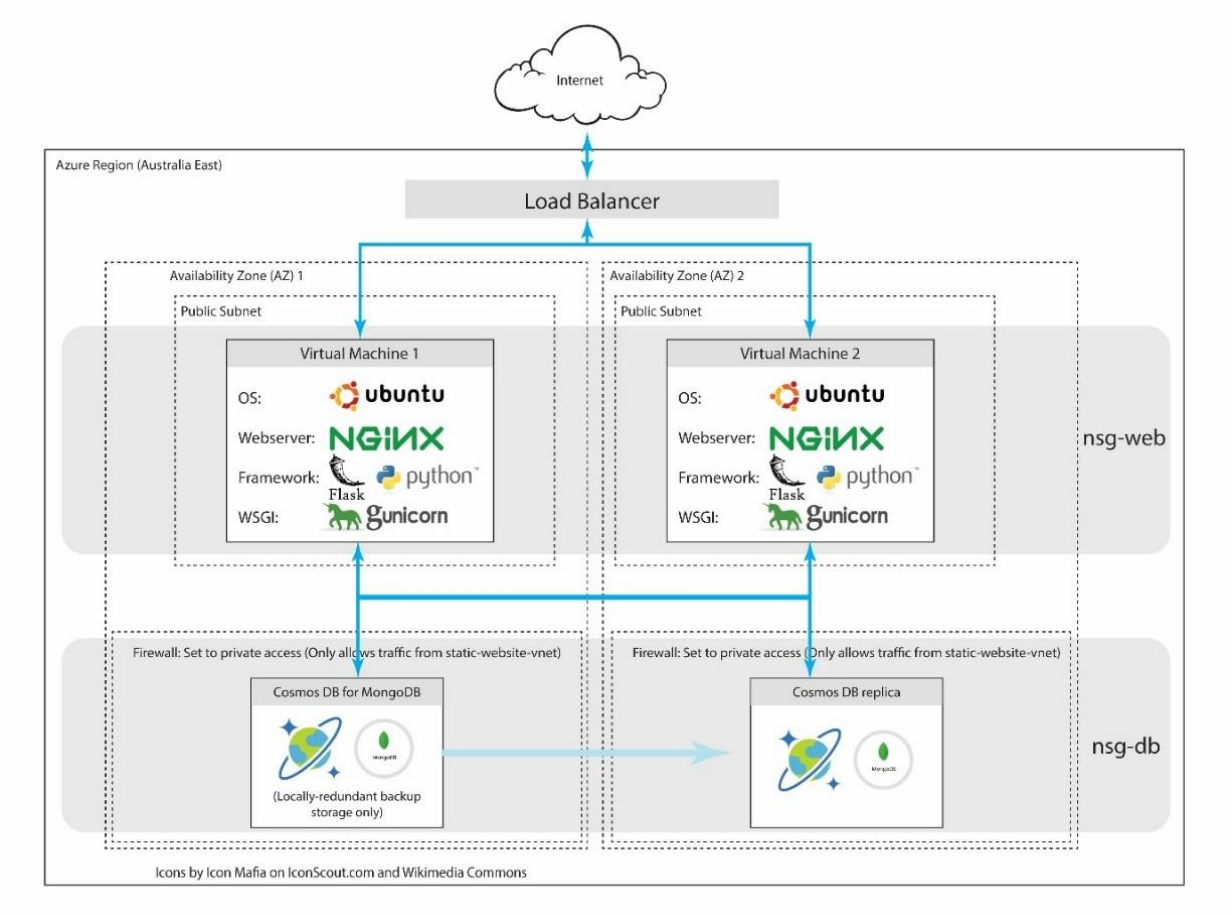
Even though the database doesn't hold any vital data, it is still good practice to protect it from public access since that ensures that i.e. the data is harder to corrupt. Therefore the Public Network Access should be limited only to specific networks.



Designing for High Availability

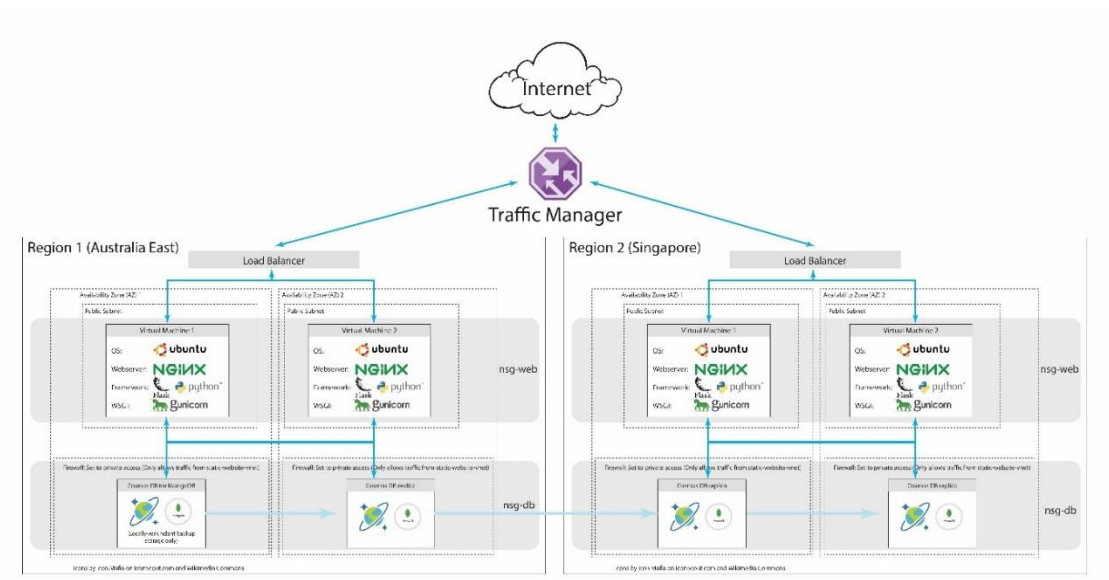
Costing Estimate at $116.04/month pay-as-you-go (PAYG):
https://azure.com/e/f8132a011f0b44e48a6ef8605de7faee

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.