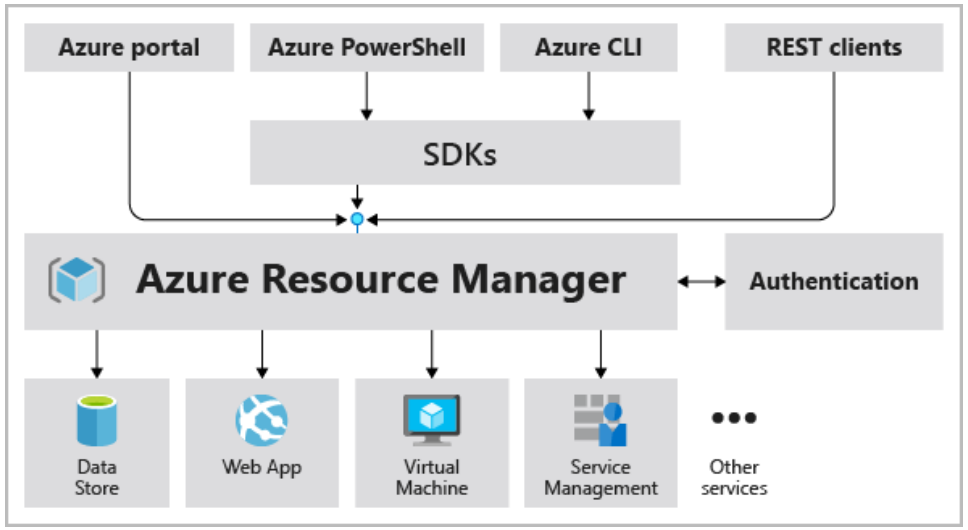
Create a VM

Infrastructure as a service

- There are many cloud data centres around the world. They are generally referred to as availability zones
- We will start off just using one to deploy a simple application
- This is not a cloud computing course so I'll stick to a need-to-know approach, you can research the rest if interested
- Azure (and others) is accessed using a web API where you can create, start, stop and remove resources
- You create a local area network, deploy virtual machines, install operating systems, deploy your application and data all via the web API
- The Azure resource manager creates resources you ask for. In Azure, all resources must belong to a resource group



- The portal is effectively an Azure web site that creates the API requests in a point and click approach. The portal is very convenient but not very practical other than for learning or trying stuff out
- Resources are normally built using code which interacts with the ARM through its API
- Code can be written in any language as long as long as it generates appropriate ARM API calls
- Or use the command line interface (CLI). The CLI needs to be installed. With this you can access the ARM API from your local machine with:
  - Microsoft Powershell with pre-written commandlets (basically pre-written functions)
  - Or shell script to issue CLI commands
- BASH and Powershell CLI is also available using Azure Cloudshell from a browser in the portal – this is an online CLI to write infrastructure as code rather than using your local machine
- So, quite a few options

- The free account only includes access to one type of VM for free - the B1S Burstable B series that's usable for up to 750 hours per month
- If you are using your student account, check the limitations. e.g. 4 cpu per region, 3 public IP addresses

Deploy to the cloud

- We are nearly ready to deploy apps to the cloud
- If you don't have a Microsoft Azure account, then:
- You will need a Microsoft account first https://account.microsoft.com/account/Account
- You should have a student account. This gives to $100 and this isn't used if you use free tier services
- Student account is quite limited - e.g., 4 cpu and 3 public IP limit but there are ways around its limitations that I'll discuss. Access at: https://azure.microsoft.com/en-us/free/students
- You could create a free account as well: https://azure.microsoft.com/en-gb/free Note: you will need a credit or debit card for this. You won't be charged if you remain within the free credit. There is no 4 cpu and 3 public IP limit using this approach
- Once you have your account, you will have root access. It is not advisable to use this unless you absolutely must, and it's worth setting 2FA on it
- It's worth then setting an admin account for yourself and use that generally. If your admin account is compromised, e.g. someone hacks you and changes the password, you can still access your account using root and create a new admin account. There is a step-by-step guide here: https://learn.microsoft.com/en-us/azure/role-based-access-control/role-assignments-portal-subscription-admin
- I can't show any of this stuff on a corporate account as I don't have access to these things

Billing alert

- Set up a billing alert so you don't leave something running by mistake and run up a bill
- From the portal:
  - Search for budgets
  - Add
  - Unique name
  - Amount
  - Next
  - Type & percentage to trigger first alert
  - Add email address
  - That's it. You will get an email if your spend or projected spend hits x% of your limit
- Note: this is an alert. It will not stop you being billed so make sure your email isn't in spam

Create a VM on Azure

- Various sizes and operating systems available
- Various methods available to create them - we will start with the Azure portal
- I've provided a couple of videos on how to do this from the portal
  - "Create an ssh key pair in Azure in 4 minutes.mp4" (4 mins)
  - "Create linux vm on Azure with ssh keys in 3 minutes.mp4" (3 mins)

- To connect remotely to a VM you need to create it with either a username and password or using an asymmetric key pair where the public part can be held by the cloud provider and the secret part by you. The key is used to encrypt the secure shell session SSH from your client machine to the VM
- Once the VM is setup, we have options. We could install nodejs then deploy our files and run the code in the same way we do on the local machine. You can install a database in the same way we have on our local machine so there is nothing new there, but the more modern and scalable approach is containerisation

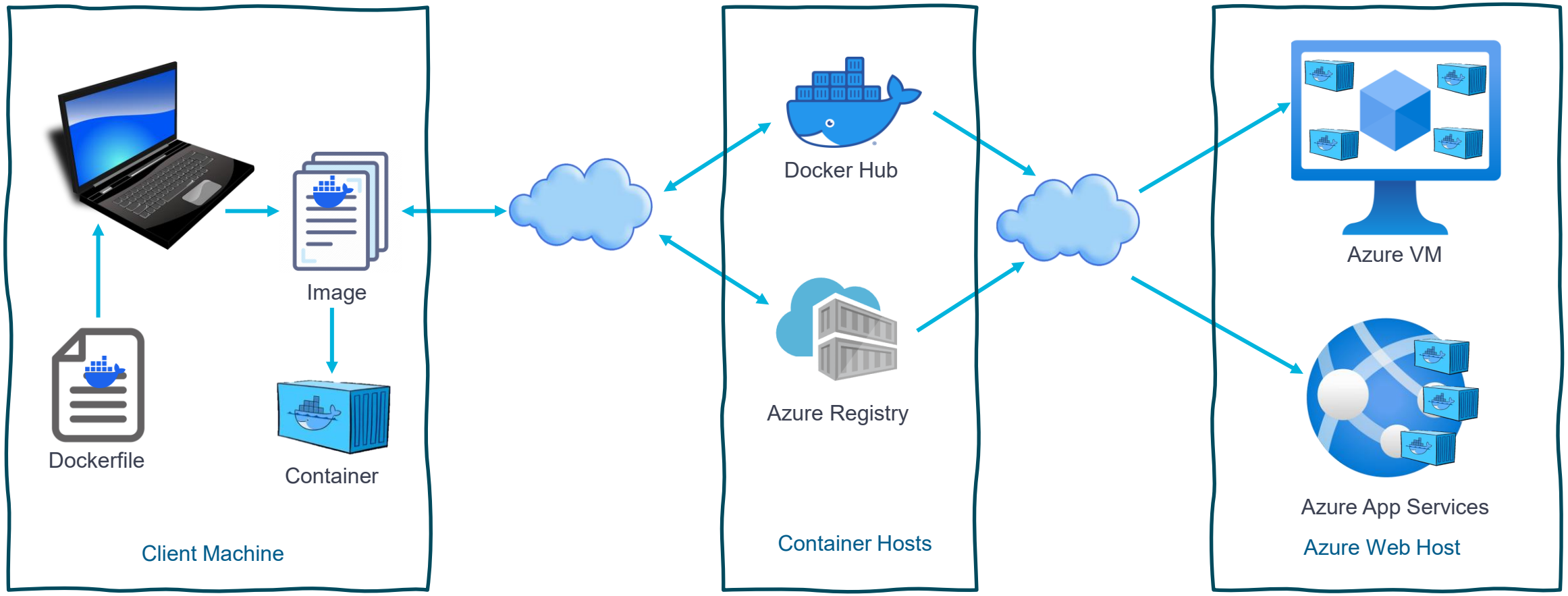
- Create a key pair
- Create a linux vm from the portal
- Connect to the vm from the client (or cloudshell - but you need to create a disk for this which is not free)
  - **ssh -i filepath/name.pem username@ipaddress**
  - or **ssh username@ipaddress** then enter the password when asked

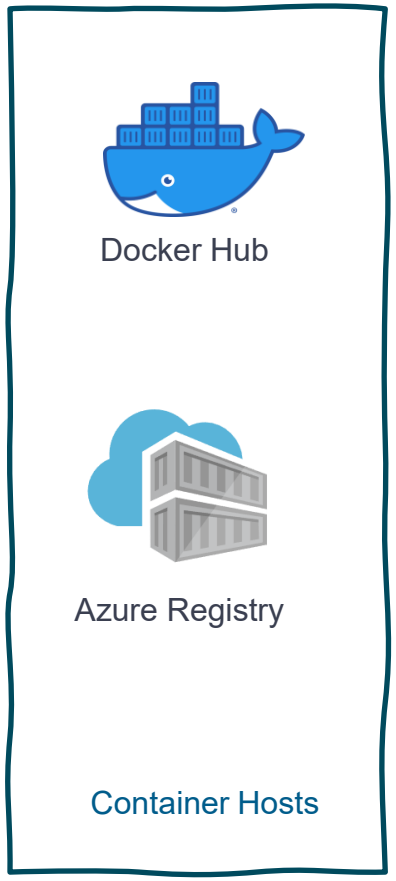
- ssh uses port 22 so this needs to be open in the vm firewall (network security group)
- Issue a command to see the server is alive
  - lsb_release -a
  - move around between dirs

create basic vm

Container deployment

Container deployment process





- A registry is a place you store your images - typically online
- This enables access to the image to others to create a container from it
- There are various container registries: Amazon Elastic Container Registry (ECR), Azure Container Registry (ACR), Docker Hub Container Registry, GitHub Container Registry and others
- You can make images available for others to download or you use theirs. I will use Docker Hub as it is the most popular by far, and free

Create a container registry

There are various options but a popular one is Docker hub

Create a Docker hub registry

- Go to https://hub.docker.com/ and Sign up
- Enter docker ID - needs to be unique to you. Fill in other stuff - choose the free one unless you want to pay
- Note: you can now only have **one** private repo. You can have many public repos
- `docker login docker.io -u oldgitdocker` or, `docker login -u oldgitdocker` assumes docker.io
- To push an image called oldgitdocker/tst, use **docker push oldgitdocker/tst**

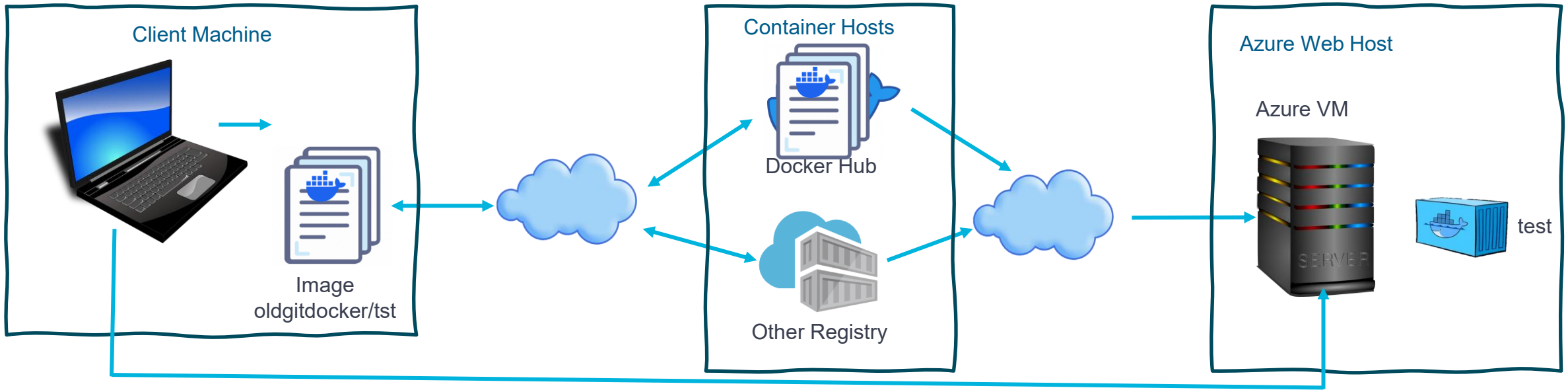
Deploy to a VM basic instructions

- Initial prerequisites:
- Create a vm - make sure you open appropriate ports - use password or key (aws doesn't support password)
- See: Create linux vm on Azure with ssh keys in 3 minutes.mp4
- Copy docker install shell script to install docker. Use winscp or other GUI app or just scp at the command line
- `scp -i c:\your_key docker-az.sh you@your_ip:/home/you/docker-az.sh`
- `scp -i c:\your_key compose.yaml you@your_ip:/home/you/compose.yaml`

- If using WinSCP, see **using winscp.mp4**
- Execute the docker install script on the remote machine by logging in to VM and entering
  - `bash docker-az.sh`
  - `It takes a couple of minutes to install`
- We can now build and deploy docker images and start them as containers on the VM

Deploying image

- The following illustrates the build and deploy process




```
docker compose build
docker login -u oldgitdocker (push)
docker push oldgitdocker/tst
ssh -i c:\key\az.pem tony@40.81.137.204
```

```
docker compose up
http://40.81.137.204:4000/tst/tony?count=30&format=json
docker compose down
```

```
scp -i c:\key\az.pem path\compose.yml tony@40.81.137.204:/home/tony
```

/demos/docker/node-docker-demo

Container vs VM

- Both are used to create an isolated virtual environment - but they are different
- VM pros:
  - Less expensive than physical computer as you can run more than one on the same hardware
  - Necessary for cloud as you have no access to the physical machines
  - Single hypervisor can manage all VM resource allocation and management
  - Each process is isolated at the hypervisor level. Each can have a different OS
- VM cons:
  - Take up a lot of resources - they are large - especially Windows VMs, many GBs so use a lot of disk space and RAM
  - A VM rarely uses all allocated resources, so is less cost efficient, but, still more cost efficient than a physical machine
- Container pros:
  - Can be small - a few MBs so potentially 1000 times smaller than a VM
  - Due to their size and not having to deploy a whole OS, they can be deployed and quickly scale through replication
- Container cons:
  - Unlike a VM, processes in a container are only isolated at the host kernel level so a little less secure
  - Because the kernel is shared, each must work with the same OS that hosts Docker
  - If one container misbehaves and causes instability in the kernel, it could affect the others
  - If data needs to persist on container deletion, or restarted, a volume needs to be created on the host or network

When to use one over the other

- A VM may be a better choice if you need to:
  - Manage a variety of operating systems
  - Manage multiple apps on a single server - i.e. make use of task switching
  - Run an app that requires all the resources and functionalities of an OS
  - Ensure full isolation and security

- A container may be a better choice if you need to:
  - Maximize the number of apps running on a server - as they are small
  - Deploy multiple instances of a single application
  - Have a lightweight system that quickly starts
  - Develop an application that runs on any underlying infrastructure
  - If it runs on the test machine, it's highly likely to run on the target machine