# Cloud Developer using Azure Nanodegree

## 2. Azure Applications
### Lesson 1: Introduction to Azure Development

cloud - collection of servers on internet to store, manage data, run apps, deliver content.
- private - on premise servers
public - Azure, AWS, GCP
hybrid

cloud computing - delivery of software + storage over internet.
- benefits - cost - no hardware cost, pay as you go.
scale - elasticity
reliability - disaster recovery, backup
security - managed

cloud developer responsibilities - plan + design cloud apps
monitor, maintain, support cloud apps
develop work flows + process

- downsides - potential downtime
- security - sensitive data
- vendor lock-in

Cloud stakeholders - users
- cloud service providers
- IT department - allow + roles
- finance department - costs
- executives - approval
- need to understand cloud
- staffing needs

Azure - scalability - managed + maintained by MS
- self-service for developers
- scale up or down (elasticity)

- availability - global data centers
- SLA 99.5% uptime
- disaster recovery

- security - platform + end-user control
- multi-factored authentication
- application password

- delivery pipeline - source control
- unit testing
- integration testing
- delivery
- live tools + applications

Azure Products
- App Service - HTTP web service, REST API, mobile backends
- .NET, .NET Core, Java, Ruby, Node.js, PHP, Python

- Virtual Machine - Windows, Linux
- images from Marketplace or custom

- Blob Storage - unstructured data

- SQL database - scalable, relational database

- Active Directory - identity + access service
- ~~sign~~ single-sign on

- Azure Monitor - logs + alerts.

| On-Premises | IaaS (Infrastructure) | PaaS (Platform) | SaaS (Software) |
|---|---|---|---|
| ✓ Applications | ✓ | ✓ | x |
| ✓ Data | ✓ | ✓ | x |
| ✓ Runtime | ✓ | x | x |
| ✓ Middleware | ✓ | x | x |
| ✓ OS | ✓ | x | x |
| ✓ Virtualization | x | x | x |
| ✓ Servers | x | x | x |
| ✓ Storages | x | x | x |
| ✓ Networking | x | x | x |

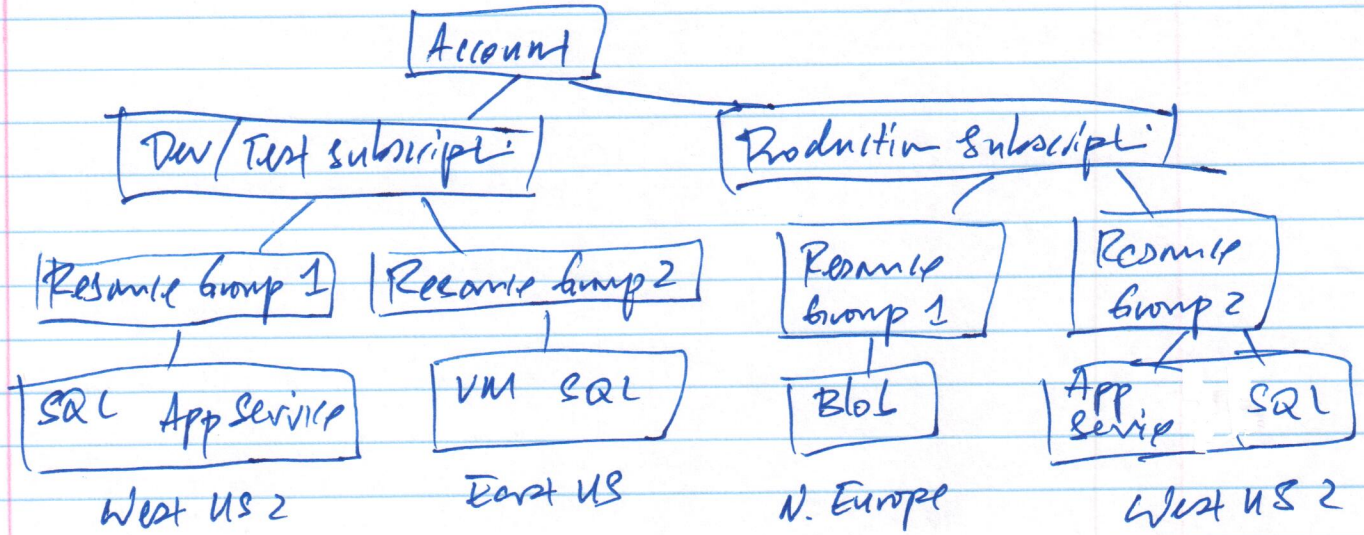
you manage (Applications, Data, Runtime, Middleware, OS)

others manage (Virtualization, Servers, Storages, Networking)

## Lesson 2: Azure Compute Services

Azure Compute Services

- VM
- App Services
- Batch - large scale, high performance
- Functions - serverless, event-driven compute on demand
- Container Instances - serverless docker containers
- Service Fabric - MS own Kubernetes
- Kubernetes Service




Create Resource Group:

- Azure Portal → Create a Resource Group → Subscription, Resource Group Name, Region

Azure CLI:
- login using 'az login'
- create using 'az group create --name Resource-Group-Name --location westus2'
- list locations using 'az account list-locations -o table'

VM - provide IaaS by allowing creation of allen VM in cloud
- full access to VM
- no hardware
- support Linux, Windows
- flexible types, sizes available (CPU, RAM)
  - compute optimized
  - memory optimized
- support custom images
- high availability + scaling

- limitations - more expensive
- labor intensive

App Service - HTTP-based service for hosting web applications, REST API, mobile backends
- .NET, .NET Core, Java, Ruby, Python, Node.js, PHP
- high availability autoscaling
- vertical + horizontal scaling
- Linux, Window supported
- Continuous Deployment supported
- PaaS
- cost based on App Service Plan
  - Dev/Test (free option)
  - Production
  - Isolated

- limitations - limited access to host server
- cost even when not in use
- hardware limits - max 14GB RAM, 4 vCPUs

VM - need full control of OS or custom software
  - testing + development
  - need more 14G RAM, 4 vCPU
App Service - lightweight applications or services
  - non-high-end compute needs.

Create VM

- Azure portal → Create a VM → Subscription
  Resource Group Name
  VM name
  Region
  Image - Ubuntu
  Size - B1Ls 1 vCPU / 0.5GiB
  Authentication type - password
  username
  password
  public inbound port - allow selected
  ports - HTTP 80
  SSH 22

- copy web app to VM
  `scp -r webapp/ username@IP:/home/username`
  `ssh username@IP`
- install nginix
  `sudo apt-get -y update && sudo apt-get -y install nginix`
  web browser → IP - Welcome to Nginx
- disconnect default nginix, redirect to port 3000
  `cd /etc/nginx/sites-available`
  `sudo unlink /etc/nginx/sites-enabled/default`
  `sudo vim reverse-proxy.conf`
  ↳ server { listen 80;
  location / {
  proxy_pass http://localhost:3000;
  proxy_http_version 1.1;
  proxy_set_header Upgrade $http_upgrade;

```
                proxy_set_header Connection keep-alive;
                proxy_set_header Host $host;
                proxy_cache_bypass $http_upgrade;
            }
        }
```

'sudo ln -s /etc/nginx/sites-available/nemus-proxy.conf /etc/nginx/sites-enabled/nemus-proxy.conf'

'sudo service nginx restart'

- install webapp
  - 'pip install -r requirement.txt'
  - 'python application.py'
- webbrowser IP

Azure CLI:

- list IP of VM
  - 'az vm list-ip-addresses -g <Resource-Group> -n <VM_Name>
- copy webapp to VM
  - 'scp -r <source-dir> username@IP:<target-dir>
- connect to VM
  - 'ssh username@IP
- continue from Install Nginx

cleanup - delete VM.

- create VM
  - 'az vm create --resource-group "resource-group-test" --name "linux-test-vm" --location "westus2" --image "Ubuntu LTS" --size "Standard_B1ls" --admin-username "" --generate-ssh-keys --verbose

- open port
  az vm open-port --port 80
  --resource-group resource-group-name
  --name linux-west-vm
- delete vm
  ' az group delete -n resource-group-name

Create App Service / Web App
- Azure portal → Create Web App → Subscription
  Resource Group Name
  Instance Name (unique)
  Publish - code or docker container
  Runtime stack - python
  OS - Linux
  Region -
  Linux plan
  Sku and size - Dev/Test or Production or Isolated
  F1

→ App Service → Deployment Center → CI/CD GitHub
GitHub Actions
configure - Repo
Branch
Workflow - add an example
Python
version
generate YAML → commit to repo
deploy to web service

- clean up
  Azure portal → Resource Group → Delete

Azure CLI:
- create app service
  `cd webapp`
  `az webapp up --resource-group *name*`
  `-- name webapp-name (unique name)`
  `-- sku F1`
  `-- verbose`

- delete resource group
  `az group delete -n resource-group-name`
- delete app service
  `az webapp delete --name webapp-name`
  `-- resource-group resource-group-name`
- delete app service plan
  `az appservice plan delete --name name`
  `-- resource_group resource-group-name`

ARM templates - create within Azure Resource Manager to spin up a given set of resources.

## Lesson 3: Azure Storage

Storage benefits - automated backup + recovery
- data replication options (at different data centers)
- data analytics support
- data encryption
- supports multiple data types
- scalable - scale up or out
- pay as you go

Options - SQL Server, Database → Relational
- Blob → unstructured
- CosmosDB

- Disk storage
- Datalake storage
- HPC cache

SQL Databases - structured, relational data. multiple
- needs SQL server first to hold databases.

- Azure portal → Create SQL Database → Subscription
Resource Group
Database name
Server → create new
↳ name
admin login
password
locate.
SQL elastic Pool - No
Compute + storage - Basic

Networking → Connectivity method - Public endpoint
Firewall rules
allow Azure services - YES
add current client IP - YES

database → Query editor → SQL server login
run SQL commands

- Azure CLI:
create SQL server
' az sql server create --admin-user username
--admin-password password
--name server-name
--location
--enable-public-network true
--version

create firewall

```
'az sql server firewall-rule create -g resource_group_name
                                    -s SQL-server-name
                                    -n firewall-rule-name
                                    --start-ip-address 0.0.0.0
                                    --end-ip-address 0.0.0.0
                                    --verbose'
```

to allow Azure services to allow server.

```
'az sql server firewall-rule create -g resource-group-name
                                    -s SQL-server-name
                                    -n clientip
                                    --start-ip-address <my ip>
                                    --end-ip-address <my ip>'
```

to allow your computer to allow database

create database

```
'az sql db create --name db-name
                  --resource-group resource-group-name
                  --server SQL-server-name
                  --tier Basic
                  --verbose'
```

delete database/server

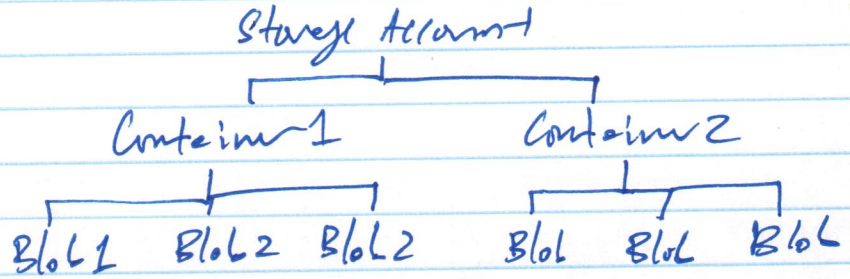
```
'az sql db delete --name db-name
                  --resource-group resource-group-name
                  --verbose'

'az sql server delete --name SQL-server-name
                      --resource-group resource-group-name
                      --verbose
```

--server SQL-server-name

Blob - Binary Large Object
- unstructured data e.g. images, videos
- high latency - not to be accessed frequently.

Storage accounts - General Purpose v2 (latest features)
GPv2
- GPv1 (no cooled, archive tiers)
- Blob Storage (legacy)




Blob storage lifecycle:
- Hot - in Hot tier
- Cool - 30 days after last modification
- Archive - 90 days " " "

- Azure Portal → Create storage → Subscription
Resource Group
Storage Account Name
Location
Performance
Account kind - GPv2
Replication - default
Access tier - hot

Networking → Connectivity method - public endpoint
Routing preference - default

Data protection

Advanced → Blob publication - enabled

Storage Account → Containers → + Container → Name
Public Access level

Container-Name → Upload
↳ URL of file

- Azure CLI:
create storage account
'az storage account create --name storage-account-name
--resource-group resource-group-name
--location
defaults to hot

create container
'az storage container create --account-name storage-account-name
--name container-name
--auth-mode login (Azure login)
--public-access container

Storage Account → Configuration → Access tier - hot to cool
→ Blob Service → Lifecycle Management
→ + New Rule
→ rule name
• move blob to cool storage
# days
• move blob to archive
# days
• delete blob
# days

Storage account → Settings → Access keys
key 1, key 2

Connect Storage to app: from endpoints defined in each Azure service

Config.py:

```
class Config(object):
    SECRET_KEY
    SQL_SERVER =                    .database.windows.net
    SQL_DATABASE
    SQL_USER_NAME
    SQL_PASSWORD
    SQLALCHEMY_DATABASE_URI

    BLOB_ACCOUNT
    BLOB_STORAGE_KEY
    BLOB_CONTAINER
```

pip install azure-storage-blob

BlobServiceClient class

- get_blob_client(container, blob = filename)
- upload_blob(file)
- delete_blob()

```
from azure.storage.blob import BlobServiceClient
blob_container = app.config['BLOB_CONTAINER']
storage_url = "https://{}.blob.core.windows.net/"
                .format(app.config['BLOB_ACCOUNT'])
blob_service = BlobServiceClient(account_url = storage_url,
                credential = app.config['BLOB_STORAGE_KEY'])
blob_client = blob_service.get_blob_client(container = blob_container,
                blob = filename)
blob_client.upload_blob = file
...
blob_client.delete_blob()
```

endpoints - SQL Server - {}.database.windows.net
App Service - {}.azurewebsites.net
Storage Account - {}.blob.core.windows.net

## Lesson 4: Security + Monitoring Basics

| | On-Premises | IaaS (Infrastructure) | PaaS (Platform) | SaaS (Software) |
|---|---|---|---|---|
| • Data governance + rights management | | ✓ | ✓ | ✓ |
| • Client endpoints | | ✓ | ✓ | ✓ |
| • Account + access management | | ✓ | ✓ | ✓ |
| • Identity + directory infrastructure | | ✓ | - | - |
| • Application | | ✓ | - | X |
| • Network controls | | ✓ | - | X |
| • Operating system | | ✓ | X | X |
| • Physical hosts | | X | X | X |
| • Physical network | | X | X | X |
| • Physical datacenter | | X | X | X |

customer (IaaS, PaaS) · shared (PaaS, SaaS) · cloud provider (IaaS, PaaS)

Security
- Active Directory - single sign
  - multi-factor authentication
- App Configuration - store application settings in 1 secure location
- Key Vault API - store application keys + secrets in 1 secure location
- Managed Identities - part of Azure AD
  - helps streamline providing app or app user access to other Azure Resources
- Shared Access Signatures - give external parties certain limited access to different Azure resources.

- Role-Based Access Control (RBAC) - help internally manage who has access to what resources, and what they can do to said resources.

- Transient faults - loss of network connectivity, service unavailability or other timeouts. App should appropriately handle.
- Azure Monitor - monitoring services such as log analytics, metrics, alerts.
- Application Insights - part of Azure monitor
  - monitor performance, other key metrics

Azure AD - Single sign on
Microsoft Authentication Library (MSAL) - to use "Sign in with Microsoft" button

tenant - Azure AD represent organization.

Azure Portal → Active Directory → Create a tenant
- Azure AD
- organization name
- domain name

→ App Registrations → New Registration
- user facing name
- who can use application
  - single tenant
  - multiple tenant
  - " " + personal MS accounts

- copy Application client ID
→ Certificates + Secrets
  - + New client secret
    - description
    - validity
    - copy value (one-time display)

Authentication - checking if user is who they say they are.

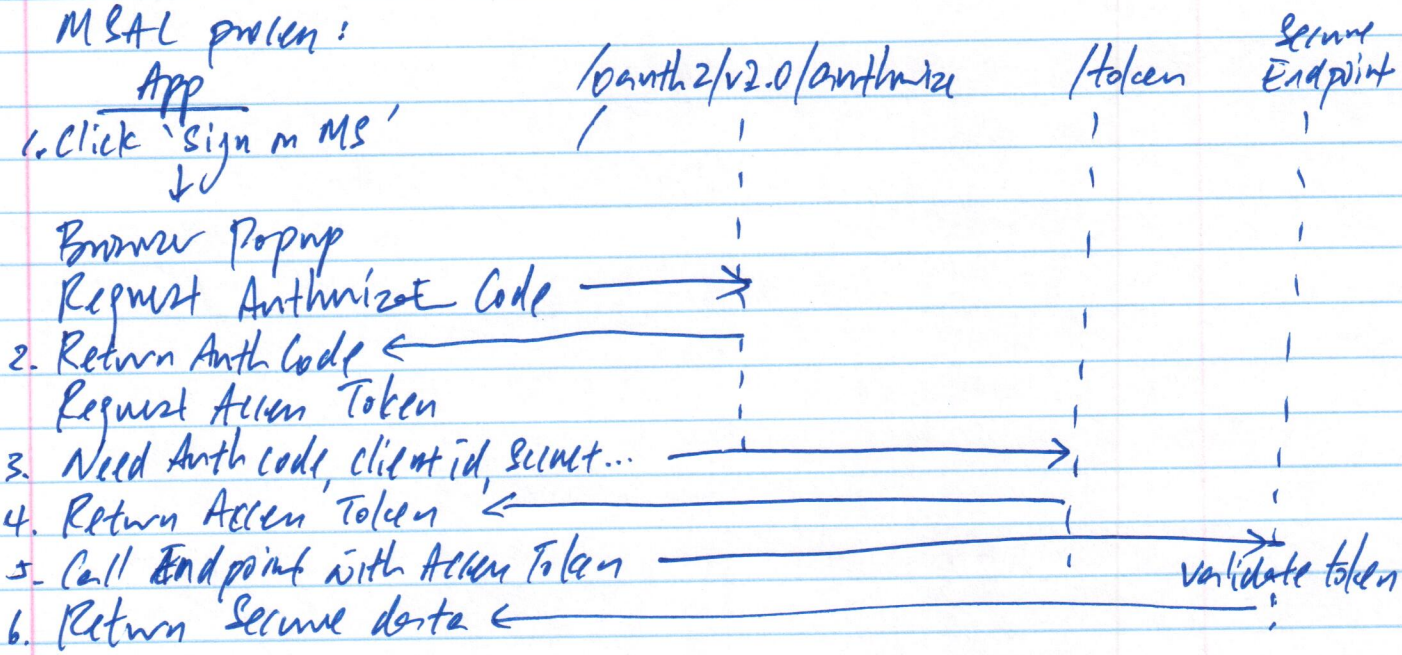
Authorization - checking if user is allowed to access data + what they can do with it.

OAuth 2.0 - industry standard protocol for authorize.
- apps delegate authorization responsibility to centralized identity provider.
- creates token to authorize user with 3rd party.

Azure AD - centralized identity provider in cloud to implement OAuth 2.0 or 'Sign in with MS'.
- provides identity as a service.

MSAL Python library
- uses tokens to allow access to API
- pip install msal

MSAL process:




Similar workflow for logout - /oauth2/v2.0/logout

1. Get Authorization Code
   - Create a Confidential Client Application
     - require Client ID, Client Secret from Azure AD
     - require "authority" based on single or multi tenant
       https://login.microsoftonline.com/com
       " / tenantName
     - can also pass cache session to avoid re-requesting information

2. Get Authorize Request URL
   - get_authorization_request_url
     - requires scope, e.g. USER.READ
       state
       redirect URI

   - Azure AD → App registrations → Redirect URI - add
     + Add a platform
       web
       enter URL
       - https://{}/getAToken

3. Get Token
   - Re-use Confidential Client Application using a cache value
   - acquire_token_by_authorization_code
     - requires authorization code, scope
       optional redirect URI

4. logout
   return redirect(Config.Authority + '/oauth2/v2.0/logout' +
   '?post_logout_redirect_uri=' + url_for('login'
   - external=True)

Benefits of logging:
- troubleshoot problem or prevent potential new ones
- improve applicat performance
- automate operations

Azure monitoring + logging
- monitor metrics
- app-based logging
- send log to Storage
- create alerts
- Applicat Insights, Log Analytics, etc...

- use built in logger in Flask
- set different severity levels of events - info, warning, error, critical

- Azure storage account - cool tier
  web app → Log stream
  → Diagnostic setting
  ✓ App ServiceConsoleLogs
  Retent
  Destinat - Storage account

→ Alerts -
+ New Alert Rule
- Condit - signal
- Action group - Resource Group
- Acti Type - Azure Functi
Email or SMS

Allow to Azure AD - requires tenant allow or 'authority'
- can't reach with enterprise account.
- need to set web app name
- 'client_ID' 'client_secret'
- 'redirect_path', logout URI

3. Azure Microservices

Lesson 1: Introduction to Azure Microservices

Microservices - each section of application is responsible for 1 function only.
- breaks up application into a set of smaller, independent services.
- each service independent from other services, but may need data feed in from other services via API.
- minimize failure at one point of failure.

Benefits of microservices:

1. Agile product launch
   - each microservice is separate component, so can be deployed independent of other services.
   - if there's bugs, isolate issues to that particular service, while rest can keep running.
   - for monolith, entire app must be taken down to debug.

2. Small (2-pizza size) teams
   - keep team size small
   - large teams have management overhead + communications problems.

3. Manageable code base
   - technical debt from large, legacy code base is wasteful.
   - larger code bases are harder to debug + manage.
   - microservices split code into smaller units.
   - easier to add features since not all components are intertwined.

4. Polyglot computing
   - flexible programming languages - JS, Ruby, Python, Java web apps
   - flexible team formation

5. Better fault handling
   - each service is decoupled.
   - failure of one service is isolated, not affecting others.

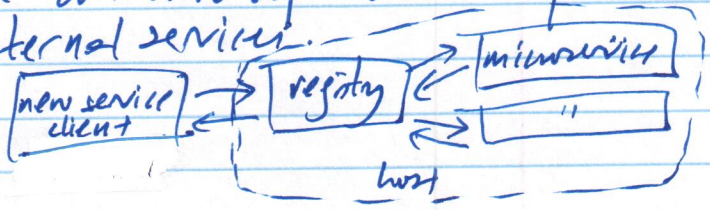
6. Decoupled databases
   - makes any changes to data scheme separate from other services

7. Scalable infrastructure
   - horizontally or vertically scaled
   - easier for instances of heavy traffic or need more computing power.

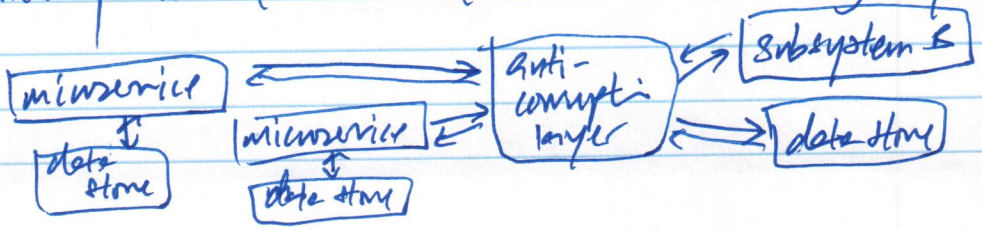
Serverless Architecture
- function as a service (FaaS)
- Azure Functions - serverless compute.
  Azure Event Grid - event routing service.

Service Orchestrator
- orchestrators manage deployment of services -
  - e.g. monitoring of service health, rebooting dead service, balancing network traffic, scaling instances, etc.

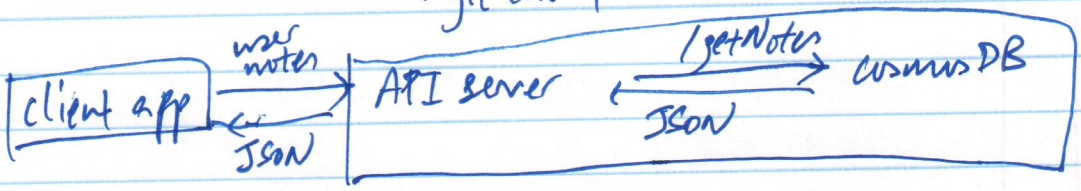
Microservices Design Patterns
1. Ambassador
   - helper service to negotiate connectivity-related requests from outside clients to internal services.
   - e.g. registry service




2. Anti-corruption layer
   - service to deal with different versions of the same application by acting as a facade between new and legacy apps.

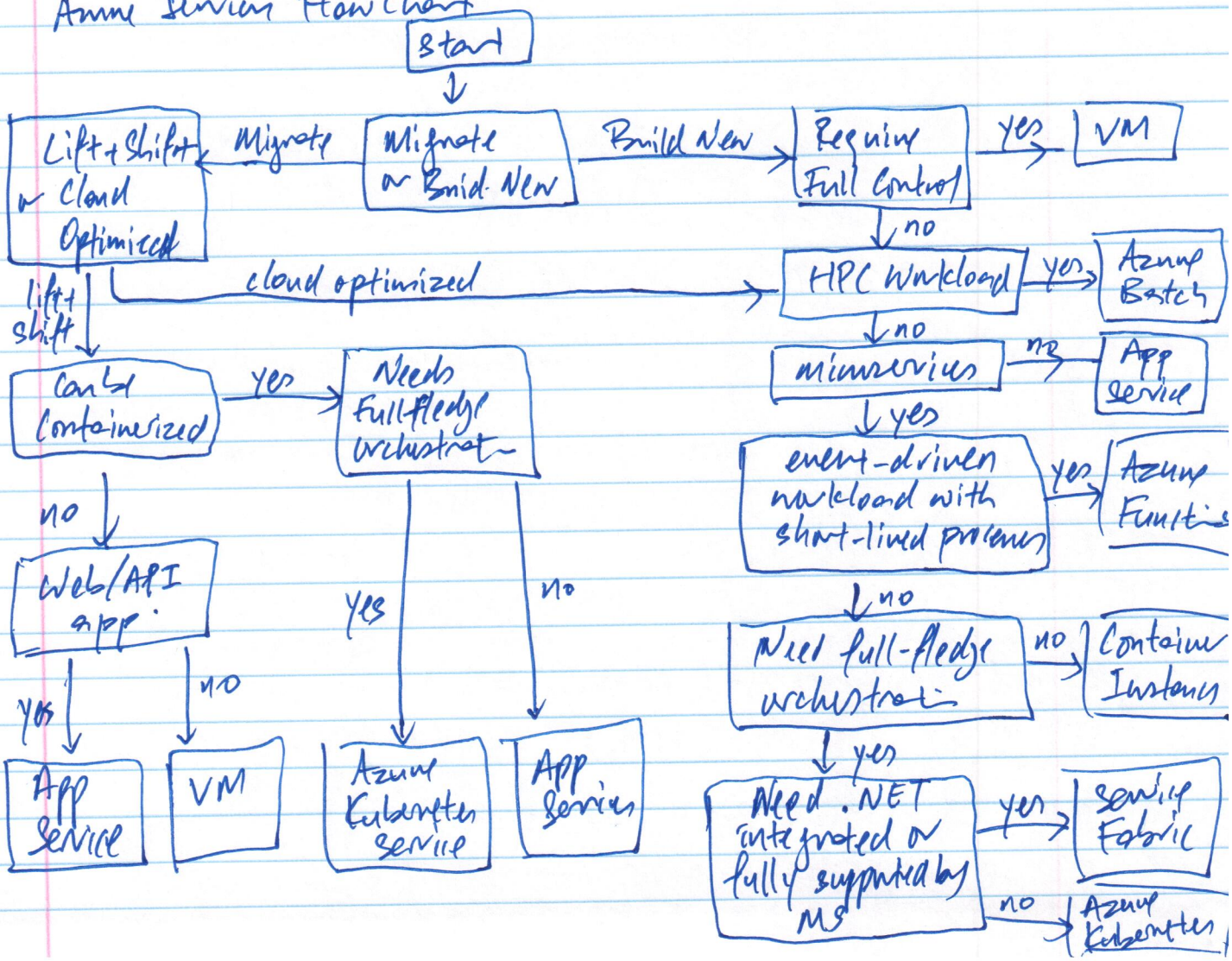


3. Backends for frontends (Backend as a service)
- create separate backend services for clients to consume.
- possible to have multiple clients using backend services.
or single client




4. Bulkhead
- minimize failures by making sure all services are available by isolating important resources.
- important in connection pools and handling multi thread requests.

Azure Services Flow Chart



- Shift from capital expenditures oriented (CapEx) budgeting towards operational expenditures (OpEx).
- CapEx - hardware costs. larger and upfront
  OpEx - day-to-day cost of running services. smaller and on-going, no upfront.

Functions
Azure offering plans:
1. Consumption
   - maximum time is 10 mins.
   - charges for run time of functi app.
   - automatically scales up if more computing is needed or down.
   - on-the-fly plan where no reservations needed.

2. Premium
   - VMs support 1, 2 or 4 core instances
   - time out 30 mins default.
     maximum time is 60 mins.
   - pay for number of cores + memory used.

3. Dedicated (App Service)
   - more control over manual VM scaling.
   - choice when have dedicated VMs with other apps running.
   - 3 runtime versions - 1.x, 2.x, 3.x
     (1.x: unlimited default max timeout; 2.x, 3.x: 30 mins default time out, unlimited max. time out)
   - legally require dedicated VMs - e.g. healthcare, legal, defense sectors
   - must enable 'Always on' setting
   - App Service plan - region
     - count of VM instances
     - size of VM "
     - price tier of plan (Free, Shared, Premium)

Other costs
- Storage account (required)
- Applicat = Insights for monitoring
- Network bandwidth for outbound data transfers to another region will incur costs.
- Azure Functions may depend on Azure CosmosDB for hosting database.

## Lesson 2: Serverless Functions

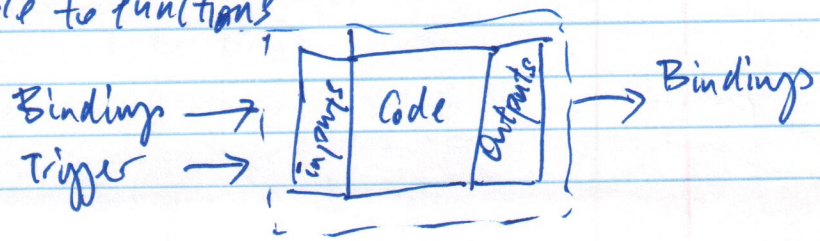
Azure Functions
- serverless - no provisioning of servers
- event-driven initiated by triggers
- compute on demand
- a la AWS Lambda
- bulk-data processing, integrating systems, support IoT.
- C#, Java, JS, Python, Powershell.

Triggers
- Schedule
  HTTP
  Blob storage / Cosmos DB
  Event Hub event / Event Grid
  Azure queue storage
  Service Bus queues + topics

  — [trigger] → Azure Functions

- 1 functi to 1 trigger

Bindings
- connect another resource to functions
- function.json



Create Functi App:

- Azure Portal → Create Storage Account
  → Create Function App → Add
  Basic → Subscripti
  Resource Group
  Functi App Name
  Publish — Code
  Runtime stack - Python is experimental
  edit in VS Code or locally
  Versin
  Regin

  Hosting - Storage account
  OS - Windows
  Plan - Consumpti

  Functi App → Functions → Add → HTTP Trigger
  → New Functin Name
  Authorizati - Anonymous ✓
  Functi
  Admin

- Azure CLI:
  ' az login '
  Storage Name =
  Function App Name =
  regin =
  myResource Group =

  - create resource group
  ' az group create -- name $myResource Group
  -- locati $ regin

  - create storage account
  ' az storage account create -- name $storage Name
  -- locat $ regin

--resource-group $myResourceGroup
--sku Standard_LRS

- create funct app
`az functionapp create --name $functionAppName
--storage-account $storageName
--consumption-plan-location $region
--resource-group $myResourceGroup
--functions-version 3
--os-type Linux
--runtime python

- create Azure funct
`func init LocalAzureFuncProject --python'
`cd LocalAzureFuncProject'
`func new --name PythonHttpExample
--template "HTTP trigger"
--authlevel "anonymous"

⇒ creates subfolder under name
contains a code file ~~and~~ (__init__.py) and function.json.
--init__.py triggered by function.json, contains Bindings

- run funct locally
`cd LocalAzureFuncProject'
to invoke funct → `func start'
navigate to URL func output + append with '?name=Azure%20Funct
should see output that trigger executed funct successfully.

- deploy funct
func azure functionapp publish functionAppName

Secure function

Authentication - Anonymous - no keys requirements to L4 API exposed to anyone

- Function - API key required
- ~~Host~~ Admin - admin keys required to allow all functions in a single function app.

Function App → Function → function
→ Function keys

Function → App keys

- edit function.json - authLevel

Azure Portal → Cost Management → Cost alerts
- Name
- reset period - monthly
- creation date
- expiration date
- Budget amount

- Alert conditions
- email recipient

Connect Azure Functions to Cosmos DB

- Azure Portal → Create Cosmos DB → Add
  Basics → Subscription
  Resource group
  Name
  API - mongoDB, Cassandra, Azure Table, Gremlin (graph)
  location
  ~~Account - Non-production~~ (obsolete)
  capacity - serverless
  Networking → connectivity method - All networks

Encrypt → Data encrypt → Service-managed key

Cosmos DB → Connect String → Read-write keys
- Primary Connect String
Secondary " "

Functi App → Functi → Configurations → New Applicat Settings
- Name
- Value

cosmos DB → Data Explorer → New Collecti
- Database name
- collecti name - unsharded
- autoscale

Data Explorer → open mongo shell
'show dbs'
'use database_name'

- Azure CLI
'az login'
resourceGroupName
locati
accountName
serverVersion
databaseName
collectionName
group
'az create -n $resourceGroupName -l locati'
'az cosmosdb create -n $accountName
-g $resourceGroupName
-- kind mongoDB
--server-version $serverVersion
--default-consistency-level Eventual

--enable-automatic-failover false

- define index policy for collection with _id

```
printf '
[
  {
    "key": {"keys": ["_id"]}
  }
]' > idxpolicy-$uniqueId.json
```

- create mongodb collection

```
'az cosmosdb mongodb collection create -g $resourceGroupName
                                       -a $accountName
                                       -d $databaseName
                                       -n $collectionName
                                       --shard 'id'
                                       --throughput 400
                                       --idx @idxpolicy-$uniqueId.json
```

- Create an Azure Function triggered by cosmosdb
  - create resource group
  - create storage account
  - create function app
  - create cosmosdb

```
'az cosmosdb create --name $functionAppName
                    --resource-group $resourceGroupName
                    --default-consistency-level Eventual
                    --locations regionName='West US 2'
                                failoverPriority=0
                                isZoneRedundant=False
                    --locations regionName='East US 2'
                                failoverPriority=1
                                isZoneRedundant=False
```

17

- create SQL API database

```
'az cosmosdb sql database create -a $functionAppName
                                 -g $ResourceGroupName
                                 -d $databaseName
                                 -n $collectionName
```

- connect Cosmos DB to function app

```
'endpoint = $(az cosmosdb show --name $functionAppName
                               --resource-group $resourceGroup
                               --query documentEndpoint
                               --output tsv)'

'key = $(az cosmosdb list-keys --name $functionAppName
                               --resource-group $resourceGroupName
                               --query primaryMasterKey
                               --output tsv)'

'az functionapp config appsettings set --name $functionAppName
                                       --resource-group $resourceGroupName
                                       --setting CosmosDB_Endpoint=$endpoint
                                                 CosmosDB_Key=$key
```

- create function cosmosdb trigger to function

• Azure portal → Function App → Functions → Add
- Development environment - Develop in portal

Select a template - cosmos
- cosmos DB trigger ←

Function name - name
Cosmos DB connection - select
Database name - name
Collection name - name
Collection name for leases - leases

Funct App → funct trigger → Code + Test
- template-based funct code - run.csx
- expand Logs section → 'Connected' displayed

Azure Cosmos DB account → Data Explorer → database name
→ New Container
- database id - existing from database name
- container id - collection name
- partition key - /category
- container - autoscale

- Test Funct
→ Data Explorer → Database Name → Items → New Item
- enter -
```
{
    "id": "id1"
    "category": "category1"
    "description": "description1"
}
```
- Funct App → Functions → Logs to confirm trigger

## Lesson 3: Enterprise Logic Apps + Event Grid

### Logic App
- work flow triggered by event
- focus on business users
  - designer-first (declarative experience)
- out-of-the-box solutions with 200+ connectors
- great for SaaS solutions outside Azure
- consumption plan - pay-per-use
  standard plan - complex configurations.
- scales automatically, but can't enforce real-time constraints.

### Azure Function
- triggered by event.
- developer-first experience
- more flexibility + control.
- perfect for serverless web applications.

- Azure Portal → Create Logic App → + Add
  - Subscription
  - Resource Group Name
  - Name
  - Location

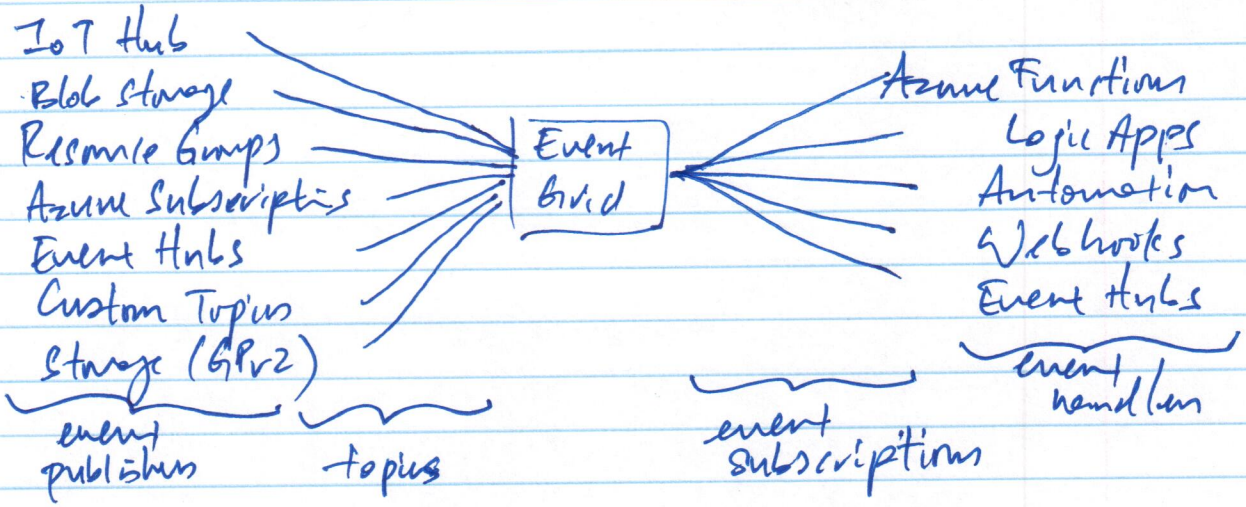
→ Logic Apps Designer → Common trigger selection - e.g. twitter, slack,
+ Blank template
trigger { - Built-in, shared, etc. search
- RSS e.g. frequency
- choose action
  - sendgrid - send email
  └ write custom code.

→ overview - list events
- events - run history

Event Grid
- centralizes management of events in Azure
- integrates with multiple Azure resources.
- Benefits of event grid
  - └ removes polling + associated costs + latency.
  - event publishers decoupled from event subscribers.
  - dynamic scaling + high availability.
  - offered in all availability zones!

1. Events - small information that describe what happen.
   contains event source - universally unique identifier.
2. Event Sources - where event took place.
3. Topics - endpoint locations where publishers send events.
4. Event subscriptions - tells Event Grid what topics to listen to.
   also allow event handlers to filter incoming events.

5. Event handlers - services reacting to events. e.g Azure Functions.




Publishers - individuals or organizations that pass events to Event Grid service.

Event Subscript expirations - events end date.

Batching - batch multiple events together into a array for publishing.

Redelivery of event - redeliver event if no confirmation of acceptance by subscribers endpoint.

- Azure Portal → Create Event Topic → + Add
  - Subscript
  - Resource Group Name
  - Name
  - Locati
  - Advanced - Event Grid Schema

topic - topic endpoint URL
Subscript ID
→ Access key → key 1/2

→ Create New Subscription
- Name
Event Grid Schema
Endpoint Type - Azure Functions
Web Hooks
: etc
Endpoint - function name

↳ VS Code editor → Functions → sign in
→ Function → Event Grid Trigger
- Name

automatically created - __init__.py
function.json

→ Functions → Deploy to Function App
(sends workspace to Azure Portal)

• Azure CLI
'az eventgrid topic create --name Name
--resource-group resourceGroupName
--location

Postman - create new POST request.
- enter endpoint URL: https://<topic>.<region>.eventgrid.azure.net/api/events
- Headers - aeg-sas-key - enter key
↑ allow
- enter subscription ID under topic in requestBody
"topic": "/subscriptions/<subscription ID>/resourcegroups/resourceGroupName/providers/Microsoft.EventGrid/topics/topicName"

Azure Portal → Event Grid Topic → TopicName → overview
→ metrics

## Lesson 4: Deployment

Deploy to Functions Backend:

Deploying with VS Code:
- VS Code → Functions → right click on Functi App name
  Select 'Deploy to Functi app'.

Deploying with Azure CLI:
- 'func azure functionapp publish <APP_NAME>'

- 'pipenv install' } test locally
  'funcn start' } Azure creates Docker images automatically

- check in Postmail - create GET request for /getNotes

Deploy client side flask App:
- in app.py

```
response = requests.get('https://<app-name>.azurewebsites.net/api/getnotes')

notes = response.json
return render_template("index.html", notes=notes)
```

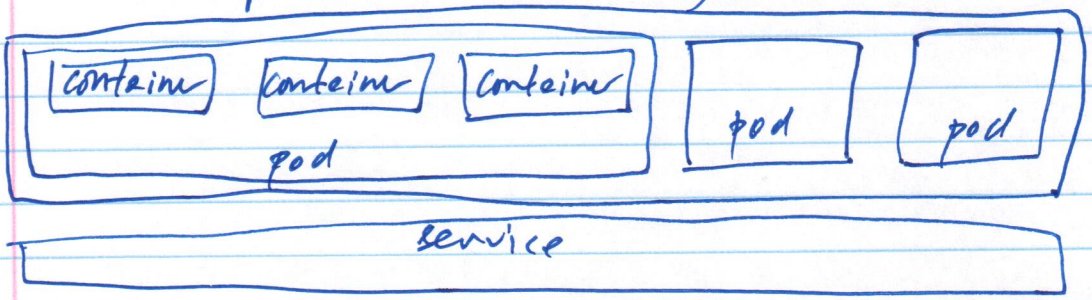
- 'python3 -m venv venv' } Azure uses virtualenv, which
  'source venv/bin/activate' } has requirements.txt
  'pip install -r requirements.txt'
  'export FLASK_RUN = app.py'
  'flask run'
  'az webapp up --sku <desired_sku> --name <app-name>
  --location eastus

Kubernetes - container orchestration system for automating
K8S app applications deployment.
- automate scheduling
- recover from failure.

- automate releases < detect latest code
- handles scaling < launch into containers

Pods - abstractions of multiple containers + ephemeral.
- containers often need to communicate to one another a

Services - abstraction of a set of pods
- applications often deployed with multiple replicas to help with load balancing + horizontal scaling.




Horizontal scaling - create additional replicas to handle more traffic.

Load balancing - handle traffic by distributing it across different endpoints.

Dockerize Azure Functions
- light weight implementation of VMs.
- lose all features of serverless computing but gain freedom to run Azure Functions independent of hosts + environments.
ie. portability, cloud agnostic.

Azure Container Registry
- create Dockerfile
'func init --docker-only --python'

- create Docker image
'docker build <path>' <path> is directory where Dockerfile is.

- Azure Portal → Create container registry → +Add
  - Subscription - Name
  - resourcegroup - resource group Name
  - registry name - name
  - locat
  - sku - basic

'docker tag <dockername> <registryname>.azurecr.io/dockername:v1'

'az acr login --name registryname'
'docker run -p 8080:80 -it mydockerimage'

Azure Kubernetes Service (AKS)
- kubernetes manages containers, grouped into pods.
- AKS hosted kubernetes service by MS Azure.

- Azure Portal → Create Kubernetes cluster
  - Basics - Subscripti
    - resource group
    - cluster name
    - regin
    - kubernetes versin
    - node size
    - node count

Node pools -
Authenicat -
Networking
Integratiom - container monitoring - Disabled

- Azure CLI:
  'az aks create --name <cluster-name>
  --resource-group <name>
  --node-count <2>
  --generate-ssh-keys'

  'kubectl get nodes'
  'az aks get-credentials --name <cluster name>
  --resource-group <name>

Containize App for AKS Deployment:
1. Create container registry:
- 'az acr create --resource-group <resource-group name>
  --name <app-registry>
  --sku Basic

- go to Azure Portal → Container Registries → Allen Keys
  → Admin user ON

- login to registry using 'username + password
  'docker login <name-of-registry>
  - username. azurecr.io
  - password

2. Containerize App:
- create docker file
  'func init --docker-only --python'

- build docker image
  'docker build . -t <image-name>'

- test image locally
  'docker run -p 8080:80 -it <image-name>'

- tag docker image
  `docker tag <image-name> <registry-name>.azurecr.io/<image-name>`

- push image to Azure Container Registry
  `docker push <registry-name>.azurecr.io/<image-name>`

3. Create Kubernetes cluster:
- create kubernetes cluster
  `az aks create --resource-group <resource-group-name> --name <app registry> --node-count <1> --generate-ssh-keys`

- credentials for AKS
  `az aks get-credentials --name <function_cluster> --resource-group <resource-group-name>`

- verify connection with cluster
  `kubectl get nodes`

4. Deploy App to Kubernetes:
- deploy app to container registry
  `func deploy --platform kubernetes --name <app-name> --registry <registry-name>`

- check deployment
  `kubectl config get-contexts`

4. Azure Migration

## Lesson 1: Introduction to Migrating to Azure

Benefits of migrating to Azure:

1. Scalability – increase or decrease resources to handle current or future load.

2. Managed service – infrastructure - full service, some or labor costs for maintaining + updates.

3. cost – pay-as-you-go.

4. Redundancy + resilience – backups in case of disaster + outages

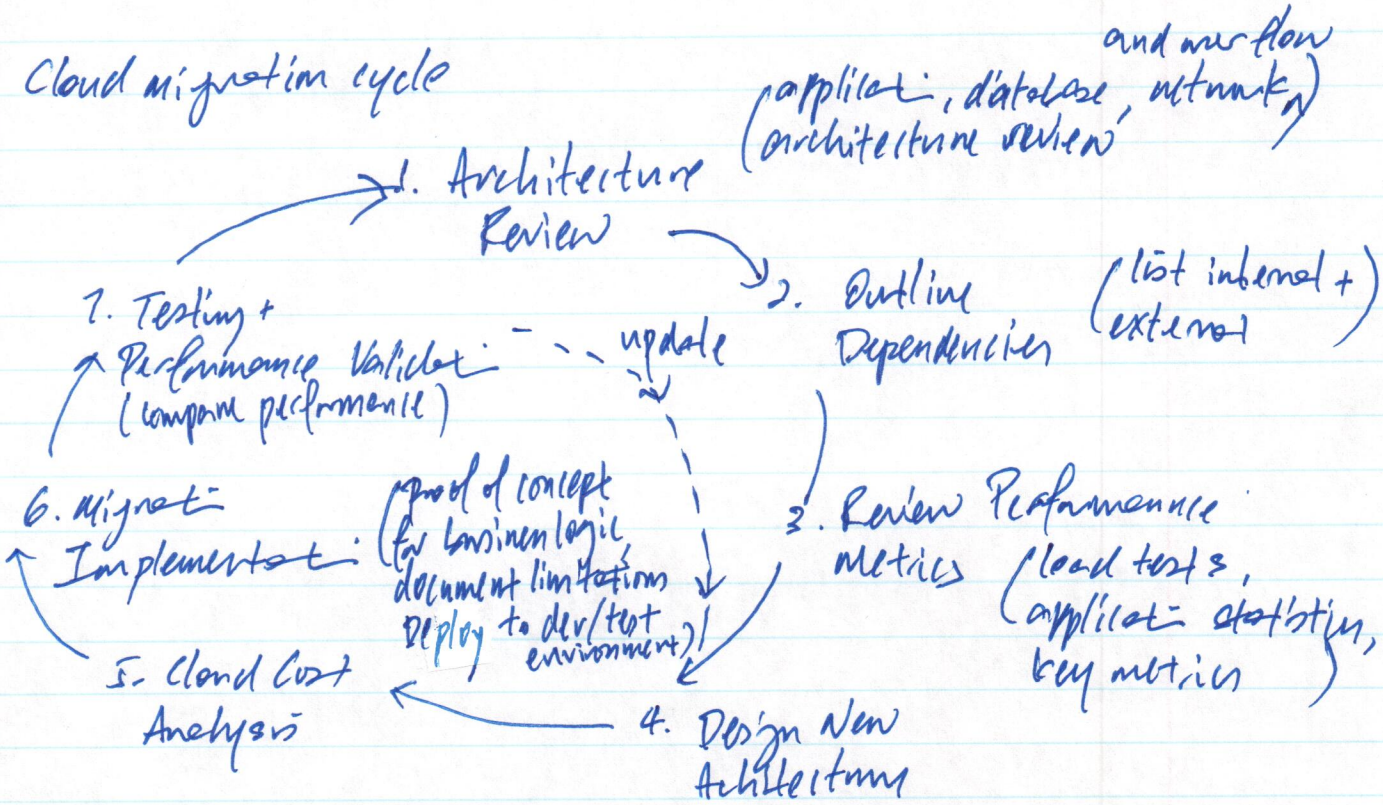
Types of migration:

1. Complete cloud migration
   - all workloads from on-premise to cloud.
   - options:
     a. refactor all workloads + move to Azure.
        time consuming due to planning, implementation, testing.
     b. lift + shift workloads to Azure with minor changes.

2. Hybrid cloud migration
   - migration executed in phases, parts are moved one by one.

3. migration within cloud.
   - move from one cloud service to another.
     e.g. from VM to Web App.

Cloud migration cycle




## Lesson 2: Migrating Web Applications

Options:
1. PaaS - Azure Web Apps
2. Lift + Shift - VMs (IaaS)

PaaS - Azure Web Apps

Pros
- easy + quick setup
- cheap with free options
- infrastructure managed for you
  - OS upgrades
  - security patches

Cons
- unable to install desired sw.
- limited access to host server.
- pay regardless of app running

Lift + shift

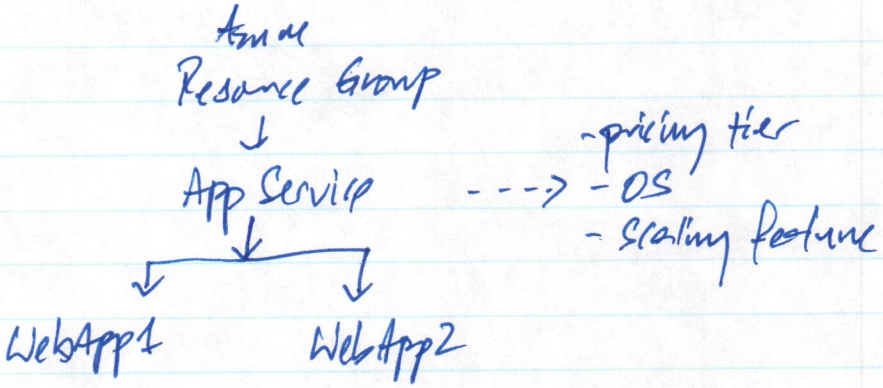
Pro
- full access to host server + underlying OS.
- ability to install + maintain specific software.
- full network configuration

Con
- require more time + labor to manage + scale.
- more expensive.
- require more resources + configurations - e.g. networking + load balancers.

Migrate Use Cases:
Azure Web App hardware limitations:
- max. 14 GB memory
- max. 4 vCPU cores
- limited language supported - C#, Java, Python, PHP, Node.js, Ruby

General guideline:
1. resource < 14 GB RAM ~ < 4 CPUs
   → Azure Web Apps

2. else → Lift + shift.



App Service Pricing Tier:

1. Shared Compute
   - no scaling due to shared resources with other users.
   - Windows OS - free tier and shared tier
   - Linux OS - free tier only

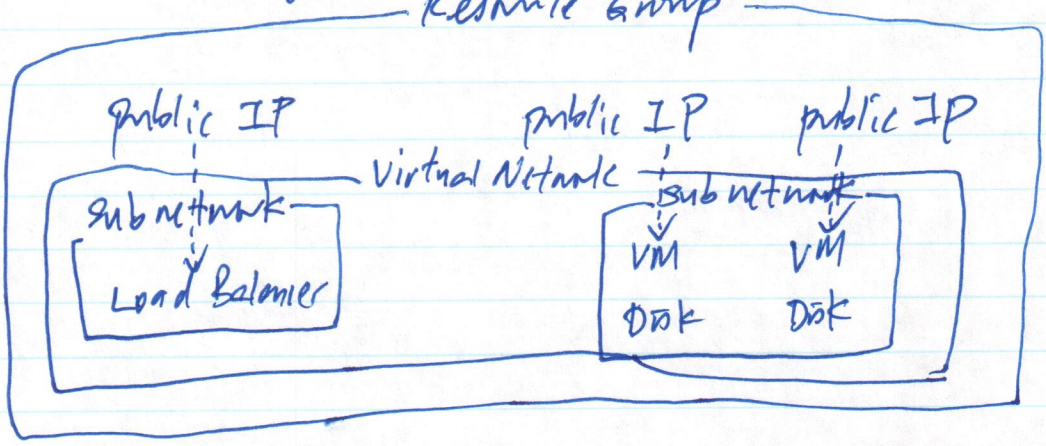
2. Dedicated Compute
   - 4 pricing tiers - Basic, Standard, Premium, Premium V2
   - Dedicated Azure VMs
   - Scalability options for manual & automatic, depending on pricing tier.

3. Isolated
   - same features as Dedicated compute
   - run on dedicated Azure Virtual Network for applications to run privately & inaccessible from public internet.

Web App Scaling
- scaling occurs at App Service, not Web App. When App Service is scaled, all applications in app service are scaled.

1. Horizontal Scaling
   - modifying number of VM instances that app service has running.
   - cost is based on number of instances running per hour.
   - Scaling out - increases } instances
     Scaling in - decrease }
2. Vertical Scaling
   - change App Service plan & pricing tier to increase & decrease resources (CPU, RAM, storage space).
   - Scaling up - more } resources
     scaling down - less }

Lift + Shift Migration




Things to consider:

1. VM sizing
   - find the right type of VM with specific size (memory, vCPU, storage).

2. Cost Prediction
   - stop incurring costs by deallocating VM.
   - charged hourly for every VM running.

3. Scaling
   - Autoscaling with VM scale sets.

Virtual Machine Scale Sets (VMSS)
- management + scaling of group of identical load-balanced VMs in order to host high availability + resilient apps.

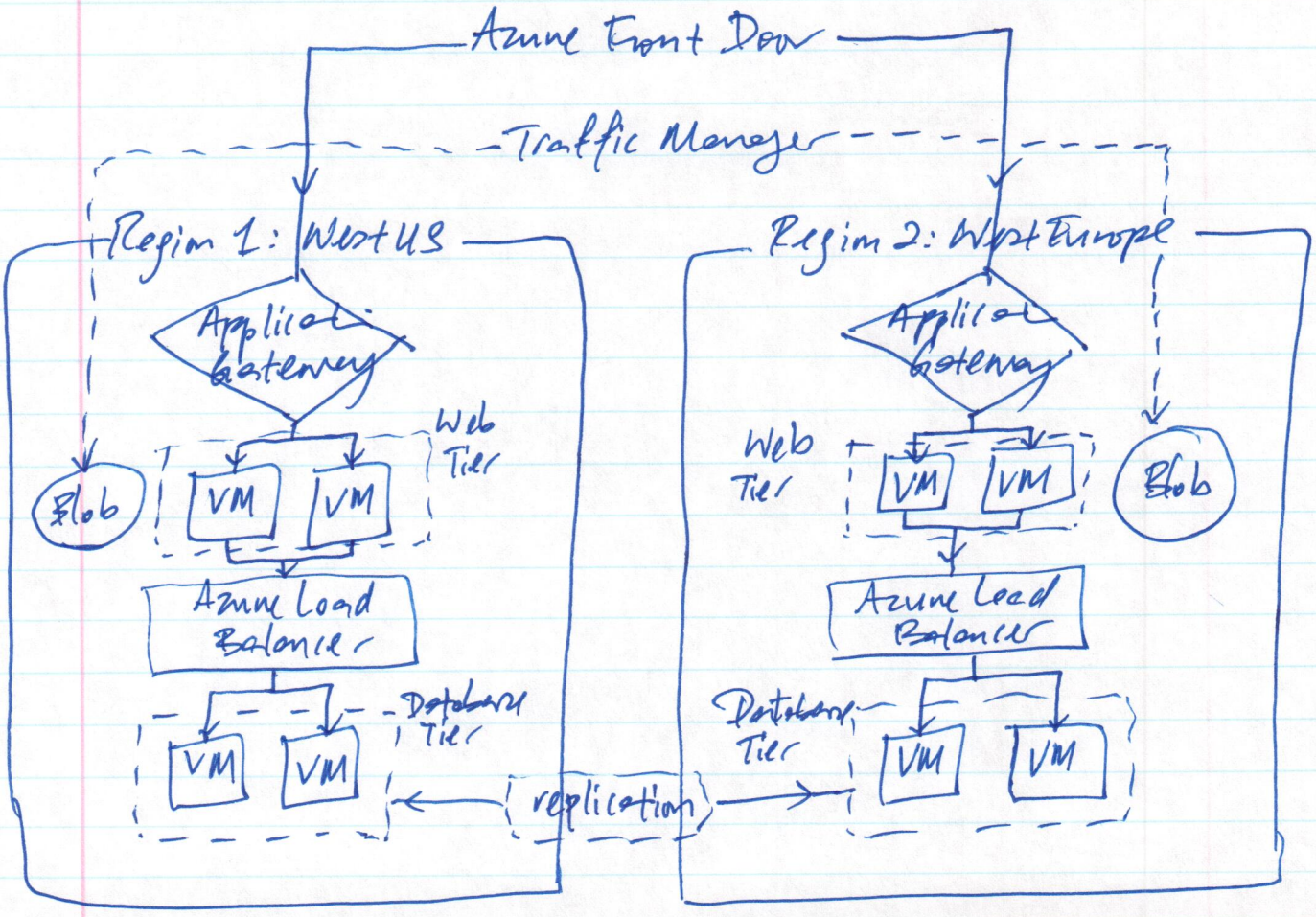
1. cloud init script
   - definition file when provisioning VMSS
   - containing details - packages to install
     - files to create on each VM.

2. custom image - image of OS with dependencies + application.

- Autoscaling with VMSS - horizontally based on key metrics e.g. CPU utilization, memory, disks, service bus queue size.

Load Balancer
- manually create VM, deploy code to each VM, create load balancer to direct traffic between VMs.

| Load Balancers | Global | Regional | Web (TCP) | Non-web (UDP) |
|---|---|---|---|---|
| Traffic Manager | ✓ | | ✓ | ✓ |
| Azure Front Door | ✓ | | ✓ | |
| Azure Load Balancer | | ✓ | ✓ | ✓ |
| Application Gateway | | ✓ | ✓ | |



Create VMSS:

Azure Portal → Create VM Scale Set → Subscripti
- Basic
  - resource group
  - name
  - region
  - image - ubuntu
  - size - B1ls
  - authentication - password
- Network
  - load balancer - yes
    - Azure load balancer
    - Create new
    - backend
- Scaling - instance count
  - scaling policy - manual or custom
    - min, max VMs
    - scale in: CPU threshold %, # VMs to decrease by
    - scale out: CPU threshold %, duration in minutes, # VMs to increase by
- management - monitoring - boot diagnostic - on
  - Azure storage
- health - monitor applicati health
- advanced - custom data - paste cloud_init
- Created - Networking - Add inbound security rule
  - destination port range - 80
  - name
- Instances - list VMs instances running.

Create Load Balancer:

1. Azure Portal → Create Virtual Network
   - Subscription
   - resource group
   - name
   - region
   - IPv4 address space 10.0.0.0/16
   - Subnet
     vm-subnet 10.1.1.0/24
     ag-subnet 10.1.2.0/24

2. → Create virtual machine

| | | 1st VM | 2nd VM |
|---|---|---|---|
| basics | - Subscription<br>- resource group<br>- VM name 1<br>- region<br>- image<br>- size - B1ls<br>- username / password<br>- inbound ports - SSH-22 | | - VM name 2 |
| networking | - virtual network<br>- subnet - vm-subnet<br>- public ip<br>- inbound ports - SSH-22 | | - vm-subnet |
| management | - boot diagnostics - off<br>- os guest " - off | | |
| advanced | - custom data - cloud init | | |

3. VS Code - terminal
   - scp -r ./SimpleWeb/ azureuser@ IP1: /home/azureuser/SimpleWeb
   " " IP2 "

4. Azure Portal → Create Application Gateway

basics
- subscription
- resource group
- name
- region
- virtual network
- subnet - ag-subnet

frontends
- public IP
- firewall public IP

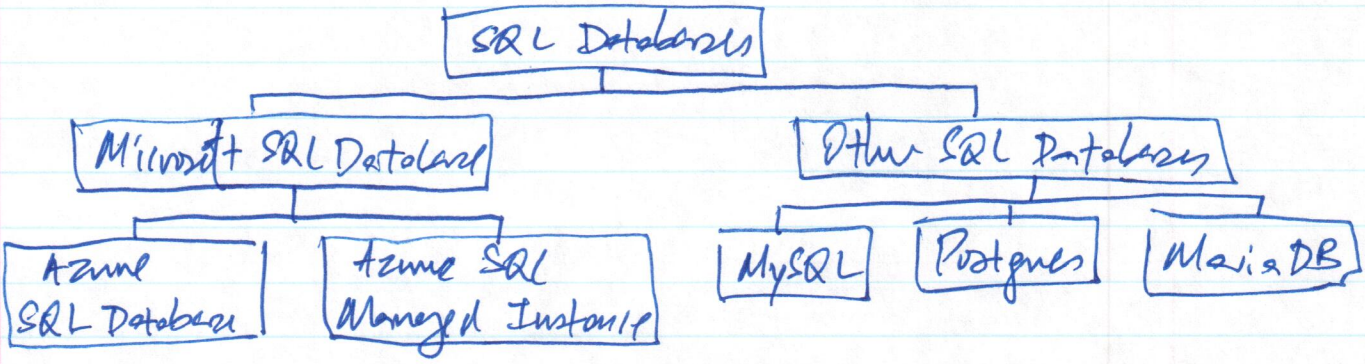
backend
- add backend pool - name
  - VM → vm1
  - VM → vm2

configure
- frontends ..... routing rules ..... backends
  - public IP
  - add rule
    - name
    - listener name
    - frontend - name
      port - 80
    - backend - name (create new)
      http setting - name
      backend port - 80

5. go to application gateway public IP

Lesson 3: Migrating Database Workloads

| Azure Databases | Managed DB Service | Database in VM |
|---|---|---|
| Configure | setup/configured for you | manually setup/config |
| Scalability | flexible | VM resizing |
| Networking | limited configuration | full control |
| Cost | storage + pricing tier | VM size per hour |




Azure SQL Database:

Pro

- only pay for what you use with flexibility to scale.

Con:

- networking configuration is limited.
- may require code changes to ensure successful migration.

Azure SQL Managed Instance:

Pro

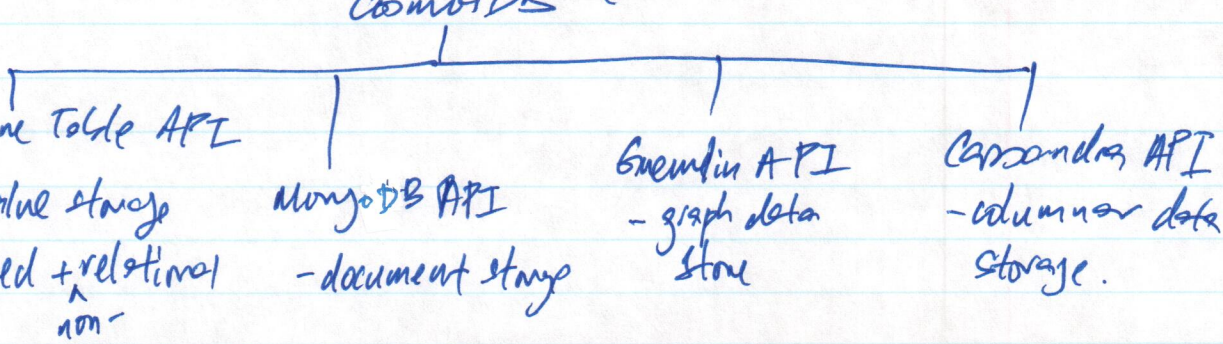
- ~100% compatible with on-premise SQL.
- little to no change in code.
- database is connected to virtual network, which allows access from private network.

Con

- can be expensive.

No SQL Database:

(multi-model (managed service, scales globally))



Create Azure SQL
Azure Portal → SQL database → Subscription
resource group
database name
Server → create new
compute + storage → name
admin login
password
region

→ server-name
→ firewalls + virtual networks
→ client IP
rule name - start IP - end IP
0.0.0.0 255.255.255.255

VS Code → Extension → SQL Server (mssql) install
connections +
→ hostname / server-name
database name
authentication type - user
- password
- profile name

→ right click on user profile
SQL query scripts

Backup + Restore:
Backup - extracting current database structure with & w/o data.
- Backup file w backup script.

Restore - bring backup database online.
- potential compatibility issues.

SQL Server
- Microsoft SQL server management studio (SSMS)
- manage SQL servers + databases.

- mssql for VS code
  - connect to databases + execute queries.
- sqlcmd utility
  - connect, create a backup, restore database using command line interface.

Postgres
- pgAdmin
  - administrative database UI tools to manage databases.
- PostgreSQL for VS code
  - connect to PostgreSQL database, run queries, execute scripts.
- pg_dump, pg_restore
  - command-line utilities for creating, restoring Postgres database backups.

MySQL
- MySQL workbench
  - create, restore backups using import, export features.
- MySql for VS code
  - manage a MySql database in VS Code.
- mysqldump, mysqlpump
  - creating, restoring MySQL database backups.

create backup postgres tar
'pg_dump -U postgres -W -F t -d <database_name> > <path/filename.tar>'

restore backup postgres tar
'pg_restore -h <server> p 5432 --no-tablespaces -W -O -F t -x -d <database_name> -U <username>@<server_name> <path/filename.tar>'

Azure Database Migration Service
- managed service to migrate database workloads to Azure.
- assessments + recommendations
- offline migration - source database not accessible during migration.
  online migration - continuous sync between sync + target database.
- requirements
  - network connectivity between databases
  - ports opened - 443, 53, 9354, 445, 1200, 1433 (for SQL)
  - enable TCP/IP protocol for access to source database.

- pricing tiers -
  - standard tier
    - free
    - small to medium workloads
    - only offline migration

  - premium tier
    - per hour
    - large workloads
    - both online, offline migration

SQL migration tools:
1. Microsoft assessment + planning tool (MAP)
   - automated tool that provides assessment report with hardware + software details + actionable steps + recommendations.

2. data migration assistant
   - tool for upgrading SQL database to newer SQL version or migrate to Azure SQL Database by detecting compatibility issues + moving schema, data, objects to target server.

Migrate MS SQL database to Azure
Azure Portal → Create Database Migrat Service

basics
- subscript
- resource group
- name
- locat
- service mode - Azure
- pricing tier - standard, premium

network - virtual network

→ new migrat project — name
source server type - SQL server
target server type - Azure SQL database
choose type of activity - schema only ✓
offline
online
project only

1. - source SQL server name
   - authentica - username, password
2. - target server name
   - authenticate - username, password
3. - map to target databases
   select source, target databases, schema source
4. - summary
   activity name

migrat project → new-activity — offline data migration
1. select sourc
2. target
3. database map
4. configure migrat settings - which tables
5. Summary

52

## Lesson 4: Migrating Background Jobs

Background Jobs
- any tasks that can be run in another process outside of main UI application.
- benefit from faster response times, scalable architecture, better user experience.
- move logic to background jobs when:
  1. HTTP requests take more than 30 seconds.
  2. business logic + transactions use more than 40-50% of server resources.

- Categories
  1. HTTP listener -
     - single-page application that can offload any database or business logic operations to a Web API for scaling of API.

  2. Timer (scheduled task)
     - run + execute logic at specific time recurrently.

  3. Queue
     - transaction does not need to be immediately executed.
     - Azure Service Bus or Azure Storage Queue

Create Service Bus
Azure Portal → Create Service Bus → Subscription
resource group
name
location
pricing tier

→ + Queue → name
max queue size
message time to live / lock duration / enable partitioning

Create Storage Queues:
Azure Portal → Create storage account → Subscription
resource group
Storage account name
location
performance
account kind - storage V2

→ Queues → + Queues
name

Azure WebJobs:
- runs executable or script on Azure Web App. - Windows only.
- types
  1. Continuous
     - runs on all instances of web app.
     - always running. automatically restarts if script ends.

  2. Triggered
     - manually started or on schedule.
     - runs on a single instance of the web app.

- has SDK in C#
- webjobs are hosted within Azure Web App.
  Functions run directly in App Service, which allows scalability.

Azure Functions:
- code that can be executed stateless for short time.
- supported on Linux, Windows
- languages - C#, Javascript, Java, Python.
- components
  1. Trigger - kicks of functi
     - e.g. when new message is queued - Service Bus Queue.
       when file uploaded in Blob container.

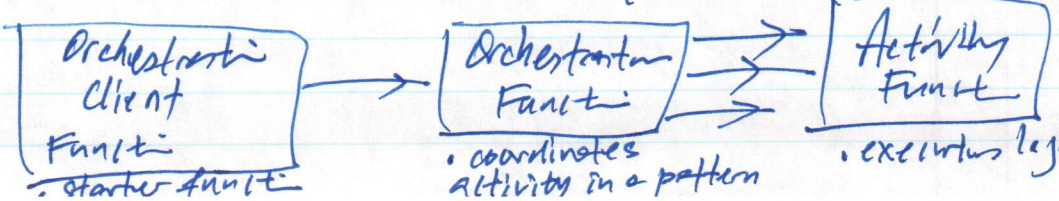
2. Bindings - input + output
- automatically connects function to other Azure resources without need to manually instantiate connections.

- hosting options
  - Consumption plan
    - automatically allocated to handle scale of requests.
    - only pay for number of executions, time of execution + memory used.

  - premium plan
    - Azure Functions instances run on preselected Azure VMs.
    - " " prewarmed to avoid cold start
    - virtual network connectivity is available with unlimited execution duration.
    - cost calculated on pricing tier selected based on CPU core and memory.

  - app service plan
    - Azure Functions hosted on same App Service plan as web app.
    - cost effective since pay for one App Service plan.

Azure Durable Functions:
- execution of stateful, long-running orchestrator functions.
- relies on Azure Storage to keep track of states during execution with a built-in autoscale orchestrator based on number of queue messages.
- supports Linux, Windows.
- supports C#, F#, Javascript only.



- Azure Durable Functions patterns

1. Function Chaining
   - break long running processes into smaller functions that can be run in sequence.

2. Fan-Out/Fan-In
   - run activity functions in parallel + at scale.

3. HTTP Long-running API
   - triggered by HTTP call, kicks off a long-running process + allows user to get status of request - Accepted, Created, Submitting, Finalizing, Complete.

Create webjob:

Azure Portal → Create a Web App → Subscription
resource group
name
publish
runtime stack
OS
region
sku + size

Basics

networking – insights

→ webjobs under settings → + Add
name
upload zip of exe
type – continuous or triggered
scheduled
CRON expression – 0/5 * * * * *
min hr month day of week year

5/6 → select webjob, click on Logs to see history.

Create Azure Function in VS Code:

VS Code → Functions → Create New Project
- name
- language - python
- trigger - HTTP trigger
  - name
  - authorization - Anonymous, Admin

- Function Name Folder
  - __init__.py
  - function.json
- local.settings.json
- host.json
- requirements.txt

logging HTTP Post / Get

→ Run - start debugging to run locally
http://localhost:7071/api/FunctionName

Postman → Post / Get an existing

Function triggered by Service Bus Queue in VS Code:

VS Code → Functions → Create Function
- trigger - Azure Service Bus Topic Trigger
  - name
  - create new app setting
- Subscription
- select service bus namespace
  select queue

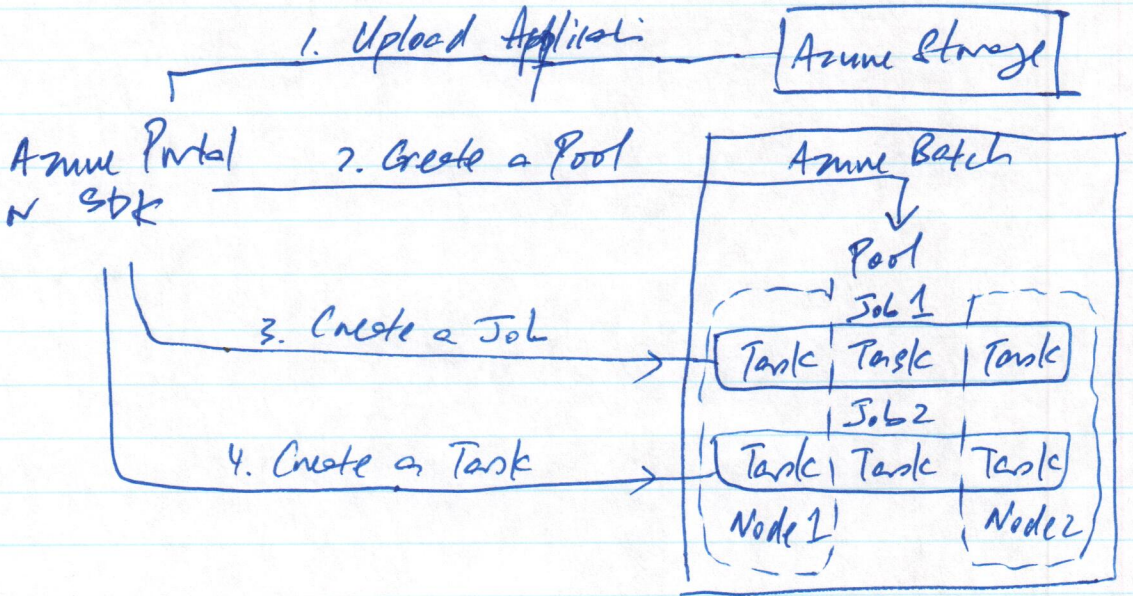
ServiceBusFunction Name - local.settings.json - bus connection string to existing service bus.

→ Deploy Function App - Subscription
- Create New Function App (Advanced)
  - name

- python
- uninstall - pay per executing
- resource group
- storage account
- applicat. insight
- region
→ Upload settings to deploy local.settings.json.

- check in Azure Portal

Azure Batch:
- run long-running background jobs in parallel in provisioned virtual machines.
- when need a background tasks that needs > 14 GB RAM, or > 4 vCPU.
- configure with:
  1. Azure Portal
  2. Azure CLI
  3. SDK (C#, Python only)




Linux: /bin/sh -c $AZ_BATCH_APP_PACKAGE_appName_1_0_0/appExe
Windows: cmd /c $AZ_BATCH_APP_PACKAGE_appName#1.0.0\appExe

Create Azure batch:
Azure Portal → Create Batch Service → Subscription
Resource Group
Name
Location
Storage Account - name
- Storage V1
- replication

Under Features → Applications → + Add - name
- version - 1.0.0
- upload package - zip ~~of~~ exe.

→ Pools → + Add - Pool ID (name)
- Image Type - marketplace.
- publisher - canonical
- offer - ubuntuserver
- sku - 18.0.4-lts
- VM size - A1 (cheapest)
- target dedicated node - 1 (minimum)
- applicat packages - select uploaded version

→ Jobs → + Add - Job ID (name)
- Pool (select)
select Job → + Add - Task ID (name)
- command line
'/bin/sh -c $AZ_BATCH_APP_PACKAGE_appName_1.0.0/appExe'

- applicat package - select uploaded version.

→ Pools to check status.
→ Jobs to check status.
→ Task to check status.

view stderr.txt & stdout.txt from app.exe.

5. Azure Performance

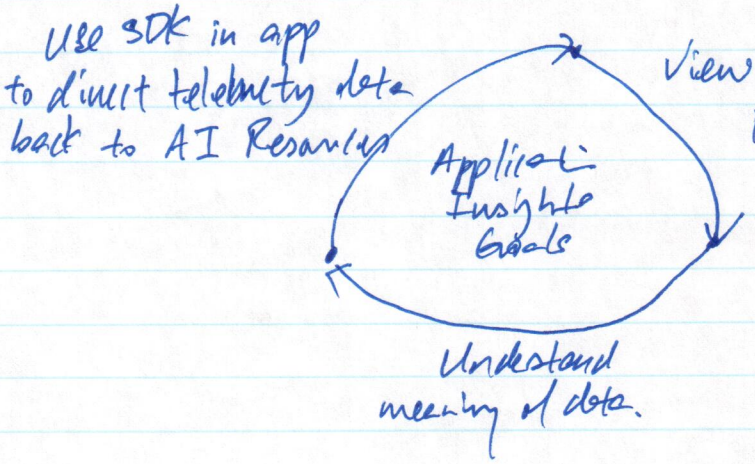
Lesson 2: ~~Application Insights~~

Application Performance Management (APM) Tools:
- monitor health of applic.
- detect + diagnose potential issues
- gather + display data - e.g. error count, performance metric

Applic Insights:
- notify potential problems & anomalies with analytics.
- reduce downtime + outages that can be costly + bad.
- problems that can cause downtime:
  1. high resource utilizat
  2. full disks
  3. high network traffic.

notify via - email, SMS, Azure push notificats, voice call.

- works with .NET, Node.js, Java, Python
- enable Azure Resource on a VM to inform of application's CPU, RAM, Disk, Network,
- ingest log data.




Azure Log Analytics Workspace - environment for Azure Monitor log data.
- log/insight queries will select monitor/logging data.

AI provides out-of-box data points/measures
1. failure metrics
2. application performance metrics
3. end-to-end transaction metrics

Enrich metrics/events by
1. integrating AI SDK into codebase
2. send custom events + telemetry using AI SDK.

VM Scale Sets
- scaled under many different circumstances.
- catch-all for applications
- can run in non-native cloud applications.
- after host data implemented out-of-box.

Azure Portal → VM scale sets → Insights
- CPU utilization - healthy 35 to 75%
Available memory - higher the better
Bytes Sent Rate
Bytes Received Rate
Logical Disk Space Used - < 70%

Setup AI on VMSS:
'az login'
'chmod +x setup-script-sh'
'./setup-script-sh'
1. 'az group create --name
--location
--verbose'
2. 'az storage account create --name
--resource-group
--location
--sku Standard_LRS'

61

(network security group)

3. 'az network nsg create --resource-group
--name
--verbose'

4. 'az vmss create --resource-group
--name
--image
--vm-sku
--nsg
--subnet
--vnet-name
--backend-pool-name
--storage-sku
--load-balancer
--custom-data cloud_init.txt
--upgrade-policy-mode automatic
--admin-username
--generate-ssh-keys
--verbose'

5. 'az network vnet subnet update --resource-group
(associate nsg to subnet) --name
--vnet-name
--network-security-group
--verbose'

6. 'az network lb probe create --resource-group
(create health probe) --lb-name
--name
--protocol tcp
--port 80
--interval 5
--threshold 2
--verbose'

7. 'az network lb rule create --resource-group
(create network load balancer rule) --name
--lb-name
--probe-name
--backend-port-name
--backend-port 80
--frontend-ip-name loadBalancerFrontEnd
--frontend-port 80
--protocol tcp
--verbose'

8. 'az network nsg rule create --resource-group
(add port 80 to nsg) --nsg-name
--name Port_80
--destination-port-ranges 80
--direction Inbound
--priority 100
--verbose'

9. 'az network rule create --resource-group
(add port 22 to nsg) --nsg-name
--name Port_22
--destination-port-ranges 22
--direction Inbound
--priority 110
--verbose'

Deploy App to VMSS Instance:

1. 'az vmss list-instance-connection-info --resource-group
--name

→ output: {"instance 0": "ip:port"
"instance 1": "ip:port"
}
↳ IP of loadBalancer

2. 'ssh adminuser@ip -p port'

'sudo apt update'
'sudo apt install python3.7'
'sudo -H pip3 install --upgrade pip'

3. install + start Redis server - in-memory data structure store
'wget http://download.redis.io/...'
'tar xzf redis-6.2.4.tar.gz'
'cd redis..'
'make'
'redis-cli ping' → returns 'PONG'
'src/redis-server'

4. 'cd front-end'
'pip install -r requirements.txt'
'python3 main.py'

5. go to ip

6. Azure Portal → VMSS → Insights
- enable Log Analytics workspace

Applicat Insights in Azure Kubernetes Cluster:
- APM for Kubernetes
- collects data about containers, pods, nodes.
- displays data for easy analysis.
- remedy affected node by seeing terminated + unresponded pods
- keep informed by viewing pods, looking at container logs, monitoring resource utilizati.
triggering alerts.

Azure Portal → AKS → Insights
- Cluster, Nodes, Controllers, Deployments tabs
- Cluster - node CPU utilizat %
memory utilizat %
node count, active pods.

- nodes - pod status
  pod creation time
  labels
  containers requests + limits

Deploy app in AKS with AI:

1. Run locally
   - `docker-yaml up -d` → create image + run app
   - `docker images` → view images locally
   - `docker ps` → see 2 running containers
   - `docker-yaml down` → stop app.

2. Create container registry in Azure
   - `az group create --name --location`
   - `az acr create --resource-group --name --sku Basic`
   - `az acr login --name`
   - `az acr show --name --query loginServer --output table`
   - `docker tag id acrname.azurecr.io/app:v1`
   - `docker push acrname.azurecr.io/app:v1`
   - `az acr repository list --name acrname.azurecr.io --output table`

3. Create Kubernetes cluster
   `az aks create --name --resource-group --node-count --enable-addons monitoring --generate-ssh-keys --attach-acr acrname`

- 'az aks install-cli'
- 'az aks get-credentials --resource-group
--name'
- 'kubectl get nodes'

4. Deploy images to AKS
- 'az acr show --name
--query loginServer
--output table'
- 'kubectl apply -f app.yaml' → deploy app.
- 'kubectl get service app --watch' → test applicat.
- 'kubectl get service' → get external IP

5. Check
- 'kubectl get pods'
- 'az aks update --name
--resource-group
--attach-acr'
- 'kubectl set image deployment app app=acrname.azurecr.io/
app:v1'

→ redeploy
- go to Azure Portal → AKS → Insights

6. Autoscale Pods
- 'kubectl autoscale deployment app --cpu-percent=
--min=
--max='

- also in app.yaml

7. Create synthetic load
- 'kubectl run -it --rm load-generator --image=busybox /bin/sh'
→ creates container with busybox image.
- 'while true; do wget -q -O- [Public-IP]; done'
→ loop queries to cluster to increase load.

66

- 'kubectl get hpa' → check number pods
- go to Azure Portal → AKS → Insights
- 'kubectl delete hpa app' → delete horizontal autoscaling

Collecting Telemetry Data:
- telemetry data is data to collect, observe, analytize automatically
- Azure Portal → A.I → Connection String
→ Events
- # users, session, events, ...
- tracer = Tracer(
exporter = AzureExporter(
connection_string = ' '
sampler = ProbabilityScampler(1.0)
→ send all data
)
)
tracer.span(name = ' ')

tracer object → exporter ⇄ connection_string
sampler

1. Edit Dockerfile
- add dependencies
RUN pip install adal, opencensus, flask

2. Edit main.py
- add logger for custom events
logger = logging.getLogger(__name__)
handler = AzureLogHandler(connection_string = ' ')
handler.setFormatter(logging.Formatter('%(traceId)s
%(spanId)s
%(message)s'))

logger.addHandler(handler)
logger.addHandler(AzureEventHandler(connection_string = ' '))
logger.setLevel(logging.INFO)

- add metrics

```
exporter = metrics_exporter.new_metrics_exporter(
    enable_standard_metrics = True,
    connection_string = '    ')
view_manager.register_exporter(exporter)
```

- add tracer

```
tracer = Tracer(exporter = AzureExporter(
                    connection_string = '    '),
                sampler = ProbabilitySampler(1.0))
app = Flask(__name__)
```

- add requests

```
middleware = FlaskMiddleware(
    app,
    exporter = AzureExporter(connection_string = '    '),
    sampler = ProbabilitySampler(rate = 1.0))
```

- connect Redis

```
r = redis.Redis(redis_server)
```

- capture event

```
with tracer.span(name = "    ") as span:
    ....
    logger.info('    ', extra = properties)
```

3. redeploy app to VMSS.

4. check AI instance.

5. redeploy app to AKS cluster

```
docker-compose down
docker-compose up -d --build --force-recreate
```

'docker tag ID acrname.azurecr.io/app:v2'
'az acr login --name acrname'
'docker push acrname.azurecr.io/app:v2'
'kubectl set image deployment app app=acrname.azurecr.io/app:v2'

'kubectl get service app --watch'

- go to external IP

## Lesson 3: Applicat Analytics

Log Aggregator
- ingest log files, organize into tables
- searchable with queries, plot into chart
- use data to gain knowledge of how app behaves.
  - make decisions on app improvements.
  - gain awareness of how app is used.

Drive decisions with data. Applicat's analytics can help by:
1. assist in debugging code
2. offer root cause analysis
3. recreate issues
4. provide data about applicat
5. saves time
6. QA testing

Azure Log Analytics workspace
* provides one place for logs.
- sorts + organizes logs data into tables — using Kusto language.
- query to find informat efficiently, quickly + accurately ^
- Azure Portal → create Log Analytics workspace
  - resource group
  - name
  - region
  - pricing tier
- AI → logs

- use alerts on specific queries. e.g. send alert if error is found.

Collect log data:
Kusto query in Applicat Insights → Logs
```
requests
    where timestamp > ago(3d)
    limit 10
```

Display log data:
- display log data as chart - area, pie, time, column, scatter, doughnut
- slice + organize data - split, sum

Lesson 4: Azure Management

Azure Management
- create, manage, execute automated processes using scripts.
- run tasks 24/7
- kick off anytime, triggered by webhook or schedule
- provides results of each task.

Azure Automat Account
- Jobs - execution tasks + output results.
- schedules - when job should run.
- scripted runbooks - contain tasks to automate, written in Python or Powershell.
- runbooks gallery - library of runbooks.
- graphical runbooks - created using UI.
- update management - automatically handle OS updates for Linux + Windows VMs.

Azure Portal → Automat Accounts → Jobs
→ Runbooks → Create, Import, Browse Gallery

→ update management – update OS.

Runbook
- script written in Powershell, Python, or created using UI
- contains code to complete task for automation.
- scheduled, executed in automated fashion.

Azure Portal → Automat Accounts → Runbooks → Create Runbook
- name
- type – Python
- write code
- test pane → start
- publish
- start or schedule

Auto scaling Rules for VMSS
1. Schedule – scale up/down at specific times.
2. VM metrics – scale up/down based on metric (CPU, network, RAM utilization %)

Azure Portal → VM Scale Set → Scaling → Manual Scale
Custom Auto Scale
↳ Default delete
- add a scale condition

• Scale based on metrics
Scale based on specific instance count
↳ instance count
specify start/end dates
repeat specific days

→ Instances – see instances.