# Practice M6: All Together

For this module we will need a PC running recent version of **Windows**, **Linux**, or **macOS**. It can be either physical or virtual machine

## Problem (Variant A)

Implement three-tier architecture to host a PHP web application with SQL server database. In general, it must look like:

A screenshot of a cell phone

Description automatically generated

### Tasks

#### Infrastructure - 5 tasks, 15 pts

* (T101, 1 pts) Create a resource group named **RG-Solution**
* (T102, 2 pts) Create an artifact (availability set or virtual machine scale set) that provides high availability for virtual machines in the front-end group and name it **AS-FE**
* (T103, 2 pts) Create an artifact (availability set or virtual machine scale set) that provides high availability for virtual machines in the back-end group and name it **AS-BE**
* (T104, 5 pts) Create a set of **two** **Ubuntu 18.04** virtual machines in the **front-end group** each with a password set as an authentication method. If created in an availability set, name them **VM-FE-x**, where **x** is a sequence number
* (T105, 5 pts) Create a set of **two Ubuntu 18.04** virtual machines in the **back-end group** each with a password set as an authentication method. If created in an availability set, name them **VM-BE-x**, where **x** is a sequence number

#### Networking - 7 tasks, 19 pts

* (T201, 1 pts) Create a virtual network named **NET** with address space **10.0.0.0/16**
* (T202, 2 pts) Create a subnet named **NET-Sub-Front** with address space **10.0.1.0/24**
* (T203, 2 pts) Create a subnet named **NET-Sub-Back** with address space **10.0.2.0/24**
* (T204, 3 pts) Create a network security group **SG-Front**, attach it to the **NET-Sub-Front** subnet, and create two **inbound** rules – one to allow communication on port **22/tcp** and a second one to allow communication on port **80/tcp**
* (T205, 3 pts) Create a network security group **SG-Back**, attach it to the **NET-Sub-Back** subnet, and create two **inbound** rules – one to allow communication on port **22/tcp** and a second one to allow communication on port **9000/tcp**
* (T206, 4 pts) Create an external load balancer named **LBP** with the corresponding set of backend pool, health probe, and load balancing rule that maps external port **80/tcp** to internal port **80/tcp**
* (T207, 4 pts) Create an internal load balancer named **LBI** with the corresponding set of backend pool, health probe, and load balancing rule that maps external port **9000/tcp** to internal port **9000/tcp**

#### Databases - 3 tasks, 9 pts

* (T301, 3 pts) Create SQL Server and a database
* (T302, 3 pts) Configure connectivity to the server
* (T303, 3 pts) Initialize the database with the help of the **load-data.sql** file part of the supporting files set

#### Software Deployment - 5 tasks, 17 pts

* (T501, 2 pts) Install **NGINX** on all **front-end** servers. For the configuration use (you are free to modify it or use your own) the **nginx-sample.conf** file part of the supporting files set
* (T502, 3 pts) Install **PHP-FPM** on all **back-end** servers. Configure it to listen on port **9000**
* (T503, 4 pts) Install **all supplementary software** on all **back-end** servers to allow them to communicate with the SQL Server database
* (T504, 5 pts) Deploy and configure (add connection string) all **php files** (part of the supporting files set) to all back-end servers
* (T505, 3 pts) Have a fully working web application

## Solution # 1(Azure portal)

### General plan

In general, we will follow this plan:

1. Create a resource group
2. Take care of the network infrastructure (network + 2 subnets + 2 security groups with rules)
3. Create one public (external) and one internal load balancers
4. Create both front-end and back-end VMs, each in a separate availability set
5. Create and set up the database
6. Connect all together

### Actual steps

Navigate to <https://portal.azure.com>

Enter your credentials

#### Resource group

Search for **Resource groups** in the main search bar and hit **Enter**

Click on **+ Create**

Check the **Subscription**

For the **Resource group** enter **RG-Solution**

Select **West Europe** for **Region**

Click on **Review + create** and then on **Create**

Once it is created, click on the **Go to the resource group**

#### Network security group (front-end)

Click on **+ Create**

Search for **Network security group**

Click on **Create**

Check the values for the **Subscription** and **Resource group**

Enter **SG-Front**

Change the **Region** to **West Europe**

Click **Review + create**

Click **Create**

#### Security rules (front-end)

We will need to communicate over **SSH** and **HTTP**, so we must create two rules

Navigate to the security group **SG-Front**

Go to the **Inbound security rules** option under **Settings**

##### Add a rule for the SSH communication

Click on **+ Add**

Change **Destination port ranges** to **22**

Select the **TCP** option under **Protocol**

Enter **Port\_22** for **Name**

Click on **Add**

##### Add a rule for the HTTP communication

Being in the **Inbound security rules** option under **Settings**

Click on **+ Add**

Change **Destination port ranges** to **80**

Select the **TCP** option under **Protocol**

Enter **Port\_80** for **Name**

Click on **Add**

#### Network security group (back-end)

Return to the resource group

Click on **+ Create**

Search for **Network security group**

Click on **Create**

Check the values for the **Subscription** and **Resource group**

Enter **SG-Back**

Change the **Region** to **West Europe**

Click **Review + create**

Click **Create**

#### Security rules (back-end)

We will need to communicate over **port 9000/tcp** and for debugging purposes over **SSH**, so we can create the rules

Navigate to the security group **SG-Back**

Go to the **Inbound security rules** option under **Settings**

##### Add a rule for the SSH communication

Click on **+ Add**

Change **Destination port ranges** to **22**

Select the **TCP** option under **Protocol**

Enter **Port\_22** for **Name**

Click on **Add**

##### Add a rule for the communication over port 9000/tcp

Being in the **Inbound security rules** option under **Settings**

Click on **+ Add**

Change **Destination port ranges** to **9000**

Select the **TCP** option under **Protocol**

Enter **Port\_9000** for **Name**

Click on **Add**

#### Virtual Network and Subnet 1 (front-end)

Return to the resource group

Click on **+ Create**

Search for **Virtual Network**

Click on **Create**

For **Name** enter **NET**

Check the values in **Subscription**, **Resource group** and **Location** fields

Click **Next: IP Addresses**

Check that the **Address space** field contains **10.0.0.0/16**

In the **Subnet** section click on the label **default**

Change **Subnet name** to **NET-SUB-Front**

Adjust the **Address range** to be **10.0.1.0/24**

Click on **Save**

Click on the **Review + create** button and then on **Create**

#### Subnet 2 (back-end)

Click on **Go to resource** or navigate to the **NET** virtual network created earlier

Select **Subnets** under **Settings**

Click on **+ Subnet**

For **Name** enter **NET-SUB-Back**

In the **Address range (CIDR block)** enter **10.0.2.0/24**

In the **Network security group** drop-down select **SG-Back**

Click **Save**

#### Link the front-end subnet to a security group

Navigate to the **NET-SUB-Front** subnet

In the **Network security group** drop-down select **SG-Front**

Click **Save**

#### Public Load Balancer

Return to the resource group

Click on **+ Create**

Search for **Load Balancer** in the main search bar and hit **Enter**

Click **Create**

Check the **Subscription**

For **Resource group** select **RG-Solution** created earlier

For **Name** enter **LBP**

Change **Region** to **West Europe** (or the one you are using so far)

Change **SKU** to **Basic**

Click **Next: Frontend IP configuration >** button

Click the **Add a frontnd IP configuration** button

Enter **LBP-FE** in the **Name** field

Then click **Create new** under **Public IP address**

In the **Name** fieldenter **LBP-IP**

Click **OK** to confirm the IP address creation

Then click **Add** to confirm the frontend configuration creation

Click **Review + create**

Click **Create**

#### Internal Load Balancer

Return to the resource group

Click on **+ Create**

Search for **Load Balancer** in the main search bar and hit **Enter**

Click on **Create**

Check the **Subscription**

For **Resource group** select **RG-Solution** created earlier

For **Name** enter **LBI**

Change **Region** to **West Europe** (or the one you are using)

Change **SKU** to **Basic**

Change the **Type** to **Internal**

Click **Next: Frontend IP configuration**

Click the **Add a frontnd IP configuration** button

Enter **LBI-FE** in the **Name** field

In the **Subnet** drop-down list select **NET-SUB-Front** item

Change the **Assignment** to **Static**

Enter **10.0.1.254** in the **IP address** field

Then click **Add** to confirm the frontend configuration creation

Click **Review + create**

Click **Create**

#### Availability set (front-end)

Return to the resource group

Click on **+ Create**

Search for **Availability Set**

Click on **Create**

Set the **Name** to **AS-FE**

Change the **Region** to **West Europe** (or the one you are using)

Set **Update domains** to **2**

Click **Review + create**

Click **Create**

#### Availability set (back-end)

Return to the resource group

Click on **+ Create**

Search for **Availability Set**

Click on **Create**

Set the **Name** to **AS-BE**

Change the **Region** to **West Europe** (or the one you are using)

Set **Update domains** to **2**

Click **Review + create**

Click **Create**

#### Virtual machine 1 (front-end #1)

Return to the resource group

Click on **+ Create**

Click on **Ubuntu Server 18.04 LTS** in the **Popular** resources list

Check the values for **Subscription** and **Resource group**

For **Virtual machine name** enter **VM-FE-1**

Select **West Europe** for **Region**

Select **Availability set** in the **Availability options** drop-down list

Select the **AS-FE** option under the **Availability set** drop-down

Click on **Change size** if necessary

Select **B1s** and click **Select**

Change **Authentication type** to **Password**

Enter **demouser** for **Username**

Enter **ExamPrepPa66word** for **Password**

Switch the **Public inbound ports** option to **None**

Click on **Next : Disks >**

Leave everything as it is

Click on **Next: Networking >**

In the **Subnet** drop-down select **NET-SUB-Front**

For **Public IP** select **None** or for debugging purposes allow it to create one (you can use the serial console instead)

Click on **Next: Management >**

Ensure that the **Boot diagnostics** is set to **On** especially if you chose to NOT have a public IP address and you plan to use the serial console

Accept the default values

Click on **Next: Advanced >**

Paste the contents of the **fe-cloud-init.yaml** file in the **Custom data** text area

Click **Review + create**

Click **Create**

#### Virtual machine 2 (front-end #2)

Repeat the procedure once more

Return to the resource group

Click on **+ Create**

Click on **Ubuntu Server 18.04 LTS** in the **Popular** resources list

Check the values for **Subscription** and **Resource group**

For **Virtual machine name** enter **VM-FE-2**

Select **West Europe** for **Region** (or the one you are using so far)

Select **Availability set** in the **Availability options** drop-down list

Select the **AS-FE** option under the **Availability set** drop-down

Click on **Change size**

Select **B1s** and click **Select**

Change **Authentication type** to **Password**

Enter **demouser** for **Username**

Enter **ExamPrepPa66word** for **Password**

Switch the **Public inbound ports** option to **None**

Click on **Next: Disks >**

Leave everything as it is

Click on **Next: Networking >**

In the **Subnet** drop-down select **NET-SUB-Front**

For **Public IP** select **None** or for debugging purposes allow it to create one (you can use the serial console instead)

Click on **Next: Management >**

Ensure that the **Boot diagnostics** is set to **On** especially if you chose to NOT have a public IP address and you plan to use the serial console

Accept the default values

Click on **Next: Advanced >**

Paste the contents of the **fe-cloud-init.yaml** file in the **Cloud data** text area

Click **Review + create**

Click **Create**

#### Virtual machine 3 (back-end #1)

Return to the resource group

Click on **+ Create**

Click on **Ubuntu Server 18.04 LTS** in the **Popular** resources list

Check the values for the **Subscription** and **Resource group**

For **Virtual machine name** enter **VM-BE-1**

Select **West Europe** for **Region** (or the one you are using so far)

Select **Availability set** in the **Availability options** drop-down list

Select the **AS-BE** option under the **Availability set** drop-down

Click on **Change size**

Select **B1s** and click **Select**

Change **Authentication type** to **Password**

Enter **demouser** for **Username**

Enter **ExamPrepPa66word** for **Password**

Switch the **Public inbound ports** option to **None**

Click on **Next: Disks >**

Leave everything as it is

Click on **Next: Networking >**

In the **Subnet** drop-down select **NET-SUB-Back**

Accept the creation of a public IP address (for debugging purposes and easier testing)

Click on **Next: Management >**

Ensure that the **Boot diagnostics** is set to **On** especially if you chose to NOT have a public IP address and you plan to use the serial console

Accept the default values

Click on **Next: Advanced >**

Paste the contents of the **be-cloud-init.yaml** file in the **Cloud data** text area

Click **Review + create**

Click **Create**

#### Virtual machine 4 (back-end #2)

Repeat the procedure once more

Return to the resource group

Click on **+ Create**

Click on **Ubuntu Server 18.04 LTS** in the **Popular** resources list

Check the values for the **Subscription** and **Resource group**

For **Virtual machine name** enter **VM-BE-2**

Select **West Europe** for **Region** (or the one you are using so far)

Select **Availability set** in the **Availability options** drop-down list

Select the **AS-BE** option under the **Availability set** drop-down

Click on **Change size**

Select **B1s** and click **Select**

Change **Authentication type** to **Password**

Enter **demouser** for **Username**

Enter **ExamPrepPa66word** for **Password**

Switch the **Public inbound ports** option to **None**

Click on **Next: Disks >**

Leave everything as it is

Click on **Next: Networking >**

In the **Subnet** drop-down select **NET-SUB-Back**

Accept the creation of a public IP address (for debugging purposes and easier testing)

Click on **Next: Management >**

Ensure that the **Boot diagnostics** is set to **On** especially if you chose to NOT have a public IP address and you plan to use the serial console

Accept the default values

Click on **Next: Advanced >**

Paste the contents of the **be-cloud-init.yaml** file in the **Cloud data** text area

Click **Review + create**

Click **Create**

#### Database and database server

Return to the resource group

Click on **+ Create**

Click on **SQL Database** in the **Popular** resources list

Click **Create**

Check the values for **Subscription** and **Resource group**

Enter **azesqldb** in the **Database name** field

Click on **Create new** link under the **Server** field

In the **New server** windows enter the following values:

* For **Server name** enter **azesql**
* In **Server admin login** enter **demosa**
* Use **ExamPrepPa66word** for **Password**
* Select **West Europe** for **Location** (or another one you prefer)

Click on **OK**

Click on **Configure database** link under **Compute + storage**

Select **Basic** plan (**5 DTU**) and click **Apply**

Change the **Backup storage redundancy** setting to **Locally-redundant backup storage**

Click on **Review + create**

Click on **Create**

#### Setup database connectivity

Navigate to the SQL server

Click on the **Firewalls and virtual networks** option under **Settings**

Select **Yes** in **Allow Azure services and resources to access this server**

Click on **+ Add client IP**

Click **Save**

Click **OK**

#### Load data

Navigate to the SQL database

Click on **Query editor (preview)**

Enter the credentials specified during the creation process (should be **demosa / ExamPrepPa66word**)

Click **OK**

Paste the code from **sql/load-data.sql** file

Click on **Run**

Check that the data is indeed loaded into the database

#### Setup external load balancer rules

Navigate to the front-end (public) load balancer (**LBP**)

##### Backend pools

Go to **Backend pools** under **Settings**

Click on **+ Add**

For **Name** enter **LBP-BP**

Under **Virtual network** select **NET**

Select **Virtual machines** under the **Associated to**

Click the **+ Add** button

Then select both **VM-FE-1** and **VM-FE-2** and click **Add**

Click once more on **Add**

##### Health probes

Go to **Health probes**

Click on the **+ Add** button

For **Name** enter **LBP-HP**

Accept the default values

Click on **Add**

Wait a while for the health probe to be created

##### Load balancing rules

Go to **Load balancing rules**

Click on the **+ Add** button

For **Name** enter **LBP-RULE**

Select the **LBP-FE** item in the **Frontend IP address** list

Select **LBP-BP** in the **Backend pool** list

Both for **Port** and **Backend port** enter **80**

Select **LBP-HP** in the **Health probe** list

Click **Add**

Wait a while for the load balancing rule to be created

##### Inbound NAT rules

If we want to be able to connect to a particular VM behind the load balancer, we can create one or more NAT rules

Let’s create two rules for the SSH connectivity, one for each machine

Go to **Inbound NAT rules**

Click on the **+ Add** button

For **Name** enter **LBP-NAT-SSH-1**

Select **VM-FE-1** as **Target virtual machine**

Select **ipconfig1** for **Network IP configuration**

Select the **LBP-FE** item in the **Frontend IP address** list

Enter **10021** in the Frontend Port field

Leave **Service** set to **Custom**

Enter **22** in **the Backend Port** field

Set **Protocol** to **TCP**

Click **Add**

Repeat the procedure for the second VM as well, but change the name (**LBP-NAT-SSH-2**) and port (**10022**)

#### Configure the backend servers

##### VM-BE-1

*If you created a public IP address for the machine, then follow the instructions and if not, use the serial console*

Navigate to the **Overview** and copy the **Public IP address**

Open a terminal and type

**ssh demouser@<public ip>**

Modify the **PHP FPM** configuration to make it listen on port **9000**

**sudo vi /etc/php/7.2/fpm/pool.d/www.conf**

Save and close the file

Alternatively, you can execute the following command:

**sudo sed -i.bak 's@listen = /run/php/php7.2-fpm.sock@listen = 9000@g' /etc/php/7.2/fpm/pool.d/www.conf**

It will change the value in question and will make a backup copy of the original file

Restart the service and check its state:

**sudo systemctl restart php7.2-fpm.service**

**systemctl status php7.2-fpm.service**

Create a folder **/site** and change the ownership to **www-data** user and group:

**sudo mkdir /site**

**sudo chown -R www-data:www-data /site**

##### Test it

Create a simple **/site/index.php** file that contains **<?php** **phpinfo(); ?>** construction in it

Save and close the file

Alternatively, you can execute the following command to create the file:

**echo '<?php phpinfo(); ?>' | sudo tee /site/index.php**

Test it on the command line by executing:

**php /site/index.php**

You should see a dump of the PHP configuration

##### VM-BE-2

Repeat the procedure on **VM-BE-2**

#### Setup internal load balancer rules

Navigate to the back-end load balancer (**LBI**)

##### Backend pools

Go to **Backend pools** under **Settings**

Click on **+ Add**

For **Name** enter **LBI-BP**

Under **Virtual network,** you should see that the **NET** is already selected (and you can not change it)

Select **Virtual machines** under the **Associated to**

Click the **+ Add** button

Select both **VM-BE-1** and **VM-BE-2** machines

Click on **Add**

Once again, click on **Add**

##### Health probes

Go to **Health probes**

Click on the **+ Add** button

For **Name** enter **LBI-HP**

Change the **Port** to **9000**

Accept the default values for the rest of the parameters

Click on **Add**

##### Load balancing rules

Go to **Load balancing rules**

Click on the **+ Add** button

For **Name** enter **LBI-RULE**

Select **LoadBalancerFrontEnt (10.0.1.254)** item in the **Frontend IP address** list

Select **LBI-BP** in the **Backend pool** list

Change both **Port** and **Backend port** to **9000**

Select **LBI-HP** in the **Health probe** list

Click **Add**

#### Configure the front-end servers

##### VM-FE-1

Open a terminal and type

**ssh -p 10021 demouser@<public LB ip>**

Modify the **NGINX** configuration file by pasting the contents of the **conf/nginx-sample.conf** file over the default one

**sudo vi /etc/nginx/sites-available/default**

Save and close the file

Alternatively, you can copy the file over the existing one using the **scp** command

On your host machine, if you are in the folder where the source file is located, you can execute:

**scp -P 10021 nginx-sample.conf demouser@<public LB ip>:.**

This will copy the file to the home folder of the **demouser**

Then, to move the file to its final destination, you can execute:

**ssh -p 10021 demouser@<public LB ip> sudo mv nginx-sample.conf /etc/nginx/sites-available/default**

This will overwrite the existing (default) file

Restart and test that the **NGINX** service is working

**sudo systemctl restart nginx**

**systemctl status nginx**

Test the page can be open locally

**curl http://localhost/index.php**

##### VM-FE-2

Repeat the steps executed on **VM-FE-1** on this machine as well

##### Test from outside

Copy the **Public IP address** of the public load balancer

Open a browser tab and navigate to it (don’t forget to add **/index.php**)

You should see the same configuration information

#### Connect back-end servers to the database

We will set up each back-end server separately

##### VM-BE-1

*For this part, it would be better to wither have public IP addresses for the two backend VMs or at least set a NAT rules in the internal loadbalancer and use of the frontend machines to access them*

Open a session to **VM-BE-1**

Add the **Microsoft’s** repository:

**sudo bash -c "curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add -"**

**sudo bash -c "curl https://packages.microsoft.com/config/ubuntu/18.04/prod.list > /etc/apt/sources.list.d/mssql-release.list"**

Update package information

**sudo apt-get update**

Install the **Microsoft** tools:

**sudo ACCEPT\_EULA=Y apt-get install -y msodbcsql17 mssql-tools**

Install the **ODBC** driver:

**sudo apt-get install -y unixodbc-dev**

Install all necessary packages:

**sudo apt-get install -y gcc g++ make autoconf libc-dev pkg-config**

Install **PHP PEAR** and **PHP** development packages:

**sudo apt-get install -y php-pear php-dev**

Install SQL server extensions:

**sudo pecl install sqlsrv-5.8.1**

**sudo pecl install pdo\_sqlsrv-5.8.1**

Adjust the configuration:

**sudo bash -c "echo extension=sqlsrv.so > /etc/php/7.2/mods-available/sqlsrv.ini"**

**sudo bash -c "echo extension=pdo\_sqlsrv.so > /etc/php/7.2/mods-available/pdo\_sqlsrv.ini"**

Enable both modules:

**sudo phpenmod sqlsrv pdo\_sqlsrv**

Restart the service:

**sudo systemctl restart php7.2-fpm.service**

Open the **/site/index.php** file and paste the contents of the **web/index.php** file from the accompanying files

Return to the **Azure Portal** and go to the SQL database

In the **Settings** section click on **Connection strings**

Switch to **PHP** and copy the connection string for SQL Server Extension (the second one)

Paste it at the beginning of the file and enter the password for the database set during the creation process

Save and close the file

##### Local test

Test the site locally by executing:

**php /site/index.php**

##### VM-BE-2

Repeat all steps that you did on **VM-BE-1** on **VM-BE-2** as well

##### External test

Copy the **Public IP address** of the public load balancer

Open a browser tab and navigate to it but do not forget to add **/index.php** at the end

#### The end

By now, you should have a fully working solution

Of course, there are multiple ways to automate all or some parts of what has been done so far

## Solution #2 (Azure CLI)

Actions that must be executed are more or less the same, just their order and/or configuration is a little bit different

All steps that follow can be executed without any changes either in a bash session on a Linux machine or in bash mode of the Azure Cloud Shell

Should you want to execute them in a PowerShell session under Windows, then you must prefix the variables with a dollar sign ($) during their declaration

Let us log in first

**az login**

Then, we can set a few variables

**RES="RG-Solution-AZ"**

**LOC="westeurope"**

**SGF="SG-FE"**

**SGB="SG-BE"**

**NET="NET"**

**NSF="NET-SUB-Front"**

**NSB="NET-SUB-Back"**

Select a subscription if there is more than one

**az account set --subscription "<Subsription Name>"**

Now, let’s create the **resource group**

**az group create --name $RES --location $LOC**

We can create the front-end **network security group**

**az network nsg create --name $SGF --resource-group $RES**

**az network nsg rule create --name Port\_22 --nsg-name $SGF --resource-group $RES --access Allow --protocol tcp --direction inbound --priority 100 --destination-port-range 22**

**az network nsg rule create --name Port\_80 --nsg-name $SGF --resource-group $RES --access Allow --protocol tcp --direction inbound --priority 110 --destination-port-range 80**

And then the back-end **network security group**

**az network nsg create --name $SGB --resource-group $RES**

**az network nsg rule create --name Port\_22 --nsg-name $SGB --resource-group $RES --access Allow --protocol tcp --direction inbound --priority 100 --destination-port-range 22**

**az network nsg rule create --name Port\_9000 --nsg-name $SGB --resource-group $RES --access Allow --protocol tcp --direction inbound --priority 110 --destination-port-range 9000**

Then, we create the **virtual network** and the two **subnets** – one for the front-end and one for the back-end

**az network vnet create --name $NET --resource-group $RES**

**az network vnet subnet create --name $NSF --vnet-name $NET --resource-group $RES --address-prefix 10.0.1.0/24 --network-security-group $SGF**

**az network vnet subnet create --name $NSB --vnet-name $NET --resource-group $RES --address-prefix 10.0.2.0/24 --network-security-group $SGB**

The **public load balancer** with its rules and probes can be created with:

**az network public-ip create --name LBP-IP --resource-group $RES --allocation-method dynamic**

**az network lb create --name LBP --resource-group $RES --frontend-ip-name LBP-FE-IP --public-ip-address LBP-IP --backend-pool-name LBP-BP**

**az network lb probe create --name LBP-HP --lb-name LBP --resource-group $RES --protocol tcp --port 80**

**az network lb rule create --name LBP-RULE --lb-name LBP --resource-group $RES --protocol tcp --frontend-port 80 --backend-port 80 --frontend-ip-name LBP-FE-IP --backend-pool-name LBP-BP --probe-name LBP-HP**

Then, the **internal load balancer** can be created with:

**az network lb create --name LBI --resource-group $RES --frontend-ip-name LBI-FE-IP --private-ip-address 10.0.1.254 --backend-pool-name LBI-BP --vnet-name $NET --subnet $NSF**

**az network lb probe create --name LBI-HP --lb-name LBI --resource-group $RES --protocol tcp --port 9000**

**az network lb rule create --name LBI-RULE --lb-name LBI --resource-group $RES --protocol tcp --frontend-port 9000 --backend-port 9000 --frontend-ip-name LBI-FE-IP --backend-pool-name LBI-BP --probe-name LBI-HP**

Now, let’s create a pair of **network interface cards** for the front-end virtual machines:

**az network nic create --name NIC-FE-1 --resource-group $RES --vnet-name $NET --subnet $NSF --lb-name LBP --lb-address-pools LBP-BP**

**az network nic create --name NIC-FE-2 --resource-group $RES --vnet-name $NET --subnet $NSF --lb-name LBP --lb-address-pools LBP-BP**

And another one for the back-end virtual machines:

**az network nic create --name NIC-BE-1 --resource-group $RES --vnet-name $NET --subnet $NSB --lb-name LBI --lb-address-pools LBI-BP**

**az network nic create --name NIC-BE-2 --resource-group $RES --vnet-name $NET --subnet $NSB --lb-name LBI --lb-address-pools LBI-BP**

We can create the **SQL server** now:

**az sql server create --name AZESQLSRV --resource-group $RES --location $LOC --admin-user demosa --admin-password 'ExamPrepPa66word'**

Then, adjust the firewall settings to allow connections from **Azure Services** and our **Client IP address** (substitute x.x.x.x with the actual IP address):

**az sql server firewall-rule create --name AllowAzureServices --server azesqlsrv --resource-group $RES --start-ip-address '0.0.0.0' --end-ip-address '0.0.0.0'**

**az sql server firewall-rule create --name AllowClientIP --server azesqlsrv --resource-group $RES --start-ip-address 'x.x.x.x' --end-ip-address 'x.x.x.x'**

Finally, we can create the **database**:

**az sql db create --name AZESQLSRVDB --server AZESQLSRV --resource-group $RES --edition Basic --capacity 5**

We can now use the **load-data.sql** file to create the table and load it with data

For this, we can use a command similar to this one:

**sqlcmd -S azesqlsrv.database.windows.net,1433 -U demosa -P ExamPrepPa66word -d AZESQLSRVDB -i load-data.sql**

Next, we can create both **availability sets** for the front-end and back-end virtual machines:

**az vm availability-set create --name AS-FE --resource-group $RES --platform-fault-domain-count 2 --platform-update-domain-count 2**

**az vm availability-set create --name AS-BE --resource-group $RES --platform-fault-domain-count 2 --platform-update-domain-count 2**

Then, we can create the two front-end **virtual machines**. We will use the **azcli-fe-cloud-init.yaml** file provided with the resource for the module (do not forget to adjust names, ports, addresses, etc.):

**az vm create --name VM-FE-1 --resource-group $RES --availability-set AS-FE --nics NIC-FE-1 --image UbuntuLTS --size Standard\_B1s --authentication-type password --admin-username demouser --admin-password ExamPrepPa66word --custom-data azcli-fe-cloud-init.yaml**

**az vm create --name VM-FE-2 --resource-group $RES --availability-set AS-FE --nics NIC-FE-2 --image UbuntuLTS --size Standard\_B1s --authentication-type password --admin-username demouser --admin-password ExamPrepPa66word --custom-data azcli-fe-cloud-init.yaml**

And finally, we create the two back-end **virtual machines**. We will use the **azcli-be-cloud-init.yaml** file provided with the resource for the module (do not forget to adjust the connection string and any other things like names, ports, addresses, etc.):

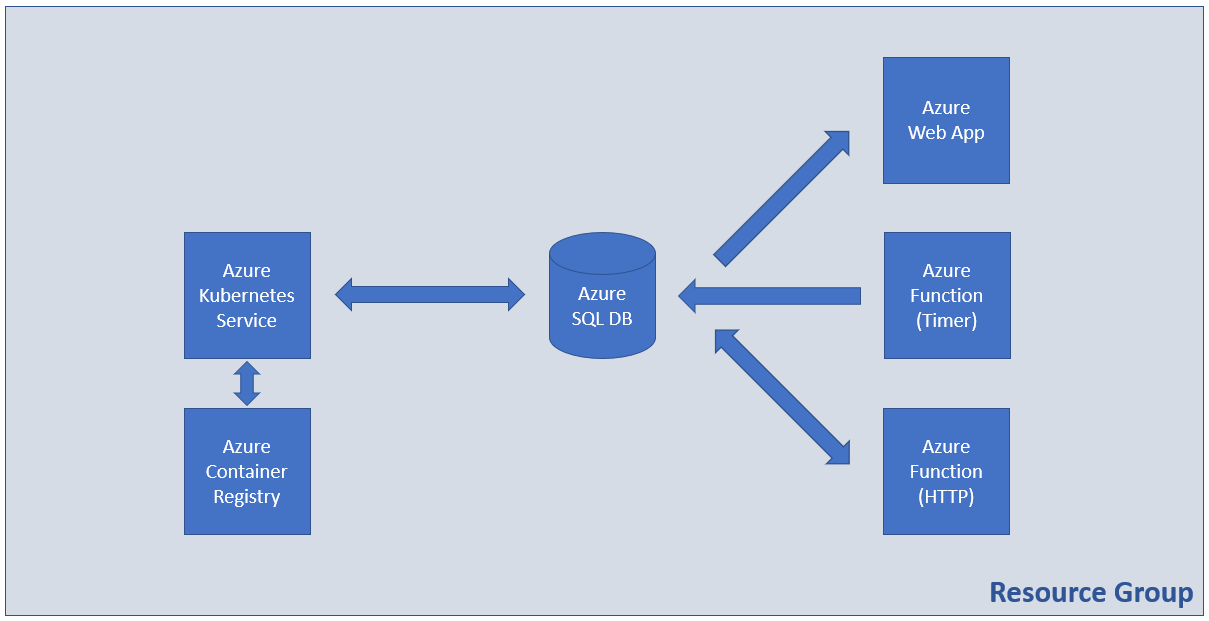
**az vm create --name VM-BE-1 --resource-group $RES --availability-set AS-BE --nics NIC-BE-1 --image UbuntuLTS --size Standard\_B1s --authentication-type password --admin-username demouser --admin-password ExamPrepPa66word --custom-data azcli-be-cloud-init.yaml**

**az vm create --name VM-BE-2 --resource-group $RES --availability-set AS-BE --nics NIC-BE-2 --image UbuntuLTS --size Standard\_B1s --authentication-type password --admin-username demouser --admin-password ExamPrepPa66word --custom-data azcli-be-cloud-init.yaml**

Now, we should have a fully working setup

## Problem (Variant B)

You are expected to create the following set of resources:



### Tasks

#### Infrastructure - 5 tasks, 13 pts

* (T101, 1 pts) Create a resource group named **RG-SolutionB**
* (T102, 3 pts) Create a container registry with **Basic** SKU
* (T103, 2 pts) Enable the Admin user
* (T104, 5 pts) Create an Azure Kubernetes Service resource with **one node** of size **B2s**
* (T105, 2 pts) **Link** the ACR to the AKS

#### Containers and Images - 7 tasks, 16 pts

* (T201, 2 pts) Add the SQL connection string to the **index.php** file in the **docker/web** folder
* (T202, 2 pts) Build the Docker image from the **Dockerfile** that is in the **docker** folder
* (T203, 2 pts) Tag the Docker image for the Azure Container Registry
* (T204, 2 pts) Publish the Docker image to the Azure Container Registry
* (T205, 3 pts) Adjust the **deployment.yaml** file in the **manifests** folder to point to the published Docker image
* (T206, 2 pts) Publish the manifests to the Kubernetes cluster (Azure Kubernetes Service)
* (T207, 3 pts) Make sure that the app is working and showing correct results

#### Databases - 3 tasks, 9 pts

* (T301, 3 pts) Create SQL Server and a database
* (T302, 3 pts) Configure connectivity to the server
* (T303, 3 pts) Initialize the database with the help of the **create-structures.sql** file part of the supporting files set

#### Web Apps and Functions - 8 tasks, 22 pts

* (T501, 3 pts) Create a PHP code-based (not container-based) web application (App Service) \*
* (T502, 2 pts) Add the SQL connection string to the **index.php** file in the **webapp** folder
* (T503, 2 pts) Deploy the web application code to Azure
* (T504, 2 pts) Make sure that the web app is working and showing correct results
* (T505, 3 pts) Create a code-based **Function App** with **.NET Core** as runtime \*
* (T506, 3 pts) Create a **Timer triggered** function. It must execute **every two minutes** and insert a row with **SubmittedName=TIMER** in the database (table **SubmittedItems**)
* (T507, 5 pts) Create a **HTTP triggered** function. When executed it must accept a single parameter (**name**), store the value in the database (table **SubmittedItems**), and return how many times the value has been inserted and when was the first time. The format should be **VALUE has N copies and the first one was inserted on TIME**. For example, if the function was called 5 times with the text **Exam**, and the first execution was on **17.10.2020 09:30**, then it should return **Exam has 5 copies and the first one was inserted on 17.10.2020 09:30**. Please note that the format of the time is not important and may not match the example
* (T508, 2 pts) Make sure that you have executed the HTTP triggered function successfully several times

*\* Note that you may need to create an additional resource group(s)*

## Solution (Azure portal)

### General plan

In general, we will follow this plan:

1. Create a resource group
2. Create container registry
3. Create Kubernetes service
4. Create and set up the database
5. Create and deploy the image
6. Create and deploy the web app
7. Create a function app and two functions

### Actual steps

Navigate to <https://portal.azure.com>

Enter your credentials

#### Resource group

Search for **Resource groups** in the main search bar and hit **Enter**

Click on **+ Create**

Check the **Subscription**

For the **Resource group** enter **RG-SolutionB**

Select **West Europe** for **Region** (or something else you prefer)

Click on **Review + create** and then on **Create**

Once it is created, click on the **Go to the resource group**

#### Container registry

Click on **+ Create**

Search for **Container registries**

Click on **Create container registry**

Make sure that the subscription, resource group, and region are all set

Enter an arbitrary name

Change the **SKU** to **Basic**

Click **Review + create** and then **Create**

Navigate to the resource

Go to **Access Keys** under **Settings** and enable the **Admin** user

#### Azure Kubernetes Service

Return to the resource group

Click on **+ Create**

Search for **Kubernetes services**

Click on **+ Create > Create a Kubernetes Cluster**

Make sure that the subscription, resource group, and region are all set

Enter an arbitrary name

Change the size of the nodes to **Standard B2s**

Set the number of nodes to **1**

Switch to **Integrations** tab

Select the registry created earlier in the **Container registry** drop-down list

Click **Review + create** and then **Create**

#### Database and database server

Return to the resource group

Click on **+ Create**

Click on **SQL Database** in the **Popular** resources list

Check the values for the **Subscription** and **Resource group**

Enter an arbitrary name in the **Database name** field

Click on **Create new** link under the **Server** field

In the **New server** windows enter the following values:

* For **Server name** enter an arbitrary name
* In **Server admin login** enter **demosa**
* Use **ExamPrepPa66word** for **Password**
* Select **West Europe** for **Location**

Click on **OK**

Click on **Configure database** link under **Compute + storage**

Select **Basic** plan (**5 DTU**) and click **Apply**

Click on **Review + create**

Click on **Create**

#### Setup database connectivity

Navigate to the SQL server

Click on the **Firewalls and virtual networks** option under **Settings**

Select **Yes** in **Allow Azure services and resources to access this server**

Click on **+ Add client IP**

Click **Save**

Click **OK**

#### Load data

Navigate to the SQL database

Click on **Query editor (preview)**

Enter the credentials specified during the creation process (should be **demosa / ExamPrepPa66word**)

Click **OK**

Paste the code from **sql/create-structures.sql** file

Click on **Run**

Check that the data is indeed loaded into the database

#### Database connection string

Navigate to the database

Go to **Connection strings** under **Settings**

Switch to the **PHP** tab

Copy the lines related to the second connection type

#### Modify image configuration

Go to the folder where the files are extracted

Open the file **docker/web/index.php**

Paste the copied connection string in the appropriate section

Don’t forget to enter the actual password you used during the database creation

Save and close the file

#### Image build and test

Open a terminal session in the folder **docker/**

Execute the following command to build the image

**docker build -t examapp .**

List the available images

**docker images**

Run the app locally:

**docker run -d -p 8080:80 --name testexamapp examapp**

Open a browser window and navigate to <http://localhost:8080> to test the app

Once everything is considered working, we may delete the running container with

**docker container rm testexamapp --force**

#### Publish the image

While still in the terminal session, log in to Azure by executing

**az login**

Select a subscription if there is more than one

**az account set --subscription "<Subsription Name>"**

Tag the image against the new registry:

**docker tag examapp <registry-name>.azurecr.io/examapp:latest**

Next, log in to the registry:

**az acr login --name <registry-name>**

Then push the image to the registry

**docker push <registry-name>.azurecr.io/examapp:latest**

#### Modify and deploy application manifests

Navigate to the **manifests/** folder

Open for editing the **deployment.yaml** file

Put the actual image name in the appropriate placeholder

Save and close the file

To get and store the credentials needed for communication with the cluster, you must execute:

**az aks get-credentials --resource-group RG-SolutionB --name <cluster-name>**

Now, you can use the **kubectl** tool to manage the cluster. Test that it is working:

**kubectl cluster-info**

Deploy both the service and application simultaneously:

**kubectl apply -f service.yaml -f deployment.yaml**

We can check periodically how it is going:

**kubectl get svc,pod**

Get the load balancer IP address and test the application

#### App Service + Web App

Return to the portal and navigate to the resource group

Click on **+ Create**

Search for **App services**

Click on **Create app service**

Make sure that the subscription, resource group, and region are all set

Enter an arbitrary name

Make sure that **Code** is selected

Select **PHP 7.4**

For **SKU and size** select **F1**

Click **Review + create** and then **Create**

Copy the connection string used earlier to the **webapp/index.php** file

Save and close the file

Go to the **Deployment Center** under **Deployment** in the **Web App** created earlier

Switch to the **FTPS Credentials** tab

Start an FTP client, for example, **FileZilla**

Connect to the service and upload the **webapp/index.php** file

Return to the **Overview** mode

Copy the application **URL** and paste it into a browser tab

The application should be working

#### Function App + Timer Trigger

Return to the portal and navigate to the resource group

Click on **+ Create**

Search for **Function App**

Click on **Create Function App**

Make sure that the subscription, resource group, and region are all set

Enter an arbitrary name

Make sure that the **Code** option is selected

For **Runtime stack** select **.NET**

For **Version** select **3.1**

Click **Review + create** and then **Create**

Once done, navigate to the function application

Switch to **Functions** under **Functions**

Click **+ Add** to add a new functions

Select the correct template

Enter name and schedule and click **Add**

Navigate to the code of the function and paste the following:

using System;

using System.Data.SqlClient;

public static void Run(TimerInfo myTimer, ILogger log)

{

string constr = "Server=tcp:<sql-server>,1433;Initial Catalog=<sql-db>;Persist Security Info=False;User ID=demosa;Password=<password>;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;";

string sqltext;

log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");

//

using (SqlConnection conn = new SqlConnection(constr))

{

conn.Open();

// Insert a row

sqltext = "INSERT INTO SubmittedItems (SubmittedName) VALUES ('TIMER')";

using (SqlCommand cmd = new SqlCommand(sqltext, conn))

{

cmd.ExecuteNonQuery();

}

}

}

Don’t forget to change the values of the database name, server, and the password

Click **Save**

Test the function and see if the web application reflects the changes

#### Function App + HTTP Trigger

Navigate to the function application

Switch to **Functions** under **Functions**

Click **+ Add** to add a new functions

Select the correct template

Enter name and click **Add**

Navigate to the code of the function and paste the following:

#r "Newtonsoft.Json"

using System.Net;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Primitives;

using Newtonsoft.Json;

using System.Data.SqlClient;

public static async Task<IActionResult> Run(HttpRequest req, ILogger log)

{

log.LogInformation("C# HTTP trigger function processed a request.");

string constr = "Server=tcp:<sql-server>,1433;Initial Catalog=<sql-db>;Persist Security Info=False;User ID=demosa;Password=<password>;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;";

string sqltext;

string name = req.Query["name"];

string requestBody = await new StreamReader(req.Body).ReadToEndAsync();

dynamic data = JsonConvert.DeserializeObject(requestBody);

name = name ?? data?.name;

if (name != null)

{

using (SqlConnection conn = new SqlConnection(constr))

{

conn.Open();

// Insert a row

sqltext = "INSERT INTO SubmittedItems (SubmittedName) VALUES ('" + name + "')";

using (SqlCommand cmd = new SqlCommand(sqltext, conn))

{

await cmd.ExecuteNonQueryAsync();

}

// Query the database

sqltext = "SELECT SubmittedName, MIN(SubmissionTime), COUNT(SubmittedName) FROM SubmittedItems WHERE SubmittedName='" + name + "' GROUP BY SubmittedName";

using (SqlCommand cmd = new SqlCommand(sqltext, conn))

{

SqlDataReader reader = cmd.ExecuteReader();

if (reader.HasRows)

{

reader.Read();

return (ActionResult)new OkObjectResult(String.Format("{0} has {2} copies and the first one was inserted on {1}", reader[0], reader[1], reader[2]));

}

else

return (ActionResult)new OkObjectResult("Nothing found for " + name);

}

}

}

else

return new BadRequestObjectResult("Please pass a name on the query string or in the request body");

}

Don’t forget to change the values of the database name, server, and the password

Click **Save**

Test the function and see if the web application reflects the changes

## Clean up

Don’t forget to stop and delete all resources that you won’t need

You can delete a resource group with all referenced resources directly

If you have any doubts, visit the **All resources** option in **Azure Portal** to check if there are any unnecessary resources left and delete them manually