MICROSOFT AI LAB – Code DEfect Ai Server Setup

Contents

[1. Azure/Infrastructure Setup 3](#_Toc22734673)

[1.1 Virtual Network 3](#_Toc22734674)

[1.2 Virtual Machines 3](#_Toc22734675)

[1.3 Create an NSG rule for the VMs 5](#_Toc22734676)

[1.4 VPN gateway 7](#_Toc22734677)

[1.4.1 Create Virtual Network Gateway 7](#_Toc22734678)

[1.4.2 Add a Gateway subnet 7](#_Toc22734679)

[1.4.3 Generate certificates for Point to Site (P2S) connection 8](#_Toc22734680)

[1.4.4 Create a self-signed root certificate 8](#_Toc22734681)

[1.4.5 Export the root certificate public key 9](#_Toc22734682)

[1.4.6 Export the client certificate 12](#_Toc22734683)

[1.4.7 Configure gateway settings to create Point to Site (P2S) connection 15](#_Toc22734684)

[1.5 Prerequisites 18](#_Toc22734685)

[1.6 Database setup 18](#_Toc22734686)

[2. Deployment 21](#_Toc22734687)

[2.1. UI Code Deployment 21](#_Toc22734688)

[2.2. Backend - prediction code setup 23](#_Toc22734689)

[2.3. API Code Deployment 26](#_Toc22734690)

[2.4. Data Cleanup 28](#_Toc22734691)

[2.4.1. Data Cleanup Process 28](#_Toc22734692)

[2.4.2. Steps to create an event 28](#_Toc22734693)

[3. Appendix 30](#_Toc22734694)

[3.1. Infrastructure Details 30](#_Toc22734695)

[3.2. Configurations used 31](#_Toc22734696)

[3.3. Network Topology Diagram 32](#_Toc22734697)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 2019/09/30 | V1.0 | CodeDefectAI installation and setup guide. | Altran |

Overview

Code Defect AI is a machine learning classifier that predicts committed source code files carrying a higher risk of a bug based on historical commits. Developers are presented with explanation behind the prediction and factors used in making the specific prediction.

This document contains the details of the setup for Code Defect AI System. The infrastructure setup and architecture diagram has been captured in the forthcoming sections.

# Azure/Infrastructure Setup

This section details out the initial Azure cloud setup which includes instantiating virtual network, vpn gateway, VM’s, security groups and certificates for accessing the machines via the vpn gateway.

The setup is done using the Azure cloud after logging in to the Azure portal.

# Virtual Network

1. Navigate to create a resource > networking > virtual network.
2. In the create virtual network pane, type or select these values:
   1. Name: Type AILab-CodeDefectAI-VirtualNetwork.
   2. Resource Group: Drop down Select existing and select required resource group (same resource group in which load balancer is present).
   3. Subnet > Name: default.
3. Select create.

# Virtual Machines

1. Navigate to create a resource > compute > ubuntu server 18.04 LTS
2. On the basics tab enter or select required details
   1. Select Subscription
   2. Select Resource
   3. Enter Virtual Machine Name
   4. Select Region
   5. Select Size
   6. Select ssh public Key for Authentication Type
   7. Enter Username
   8. Enter ssh public key which was generated using putty key generator (Puttygen) as in Figure 1

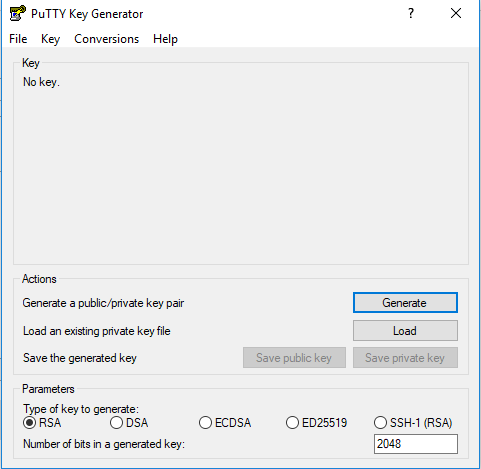


Figure 1 : PuttyGen for Key Generation

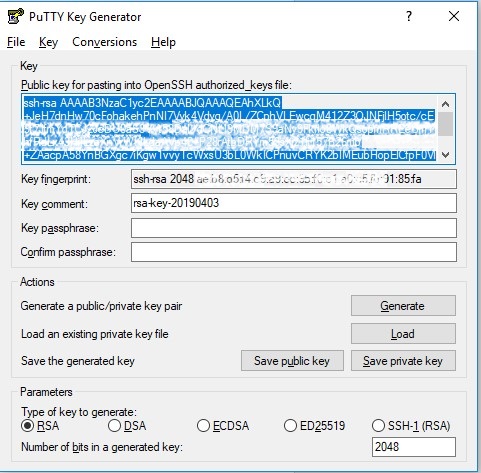


Figure 2 : Create Key

* 1. Copy & paste the generated key into the ssh public key field
  2. Click on next: disks

1. Select required disk size to be attached and click on next: networking
2. Select existing virtual network if exists or click on create new and give required name
3. Select subnet configuration
4. Provide new public ip
5. Under network security group, select advanced to create a new network security group (NSG), a type of firewall.
   1. In the configure network security group field, select create new.
   2. Enter required name for the NSG and select OK.

# Create an NSG rule for the VMs

Create a network security group (NSG) rule for the VMs to allow inbound internet (HTTP) connections.

1. Select All resources on the left menu. From the resource list, select NSG which was created while creating VM.
2. Under settings, select inbound security rules, and then select Add.
3. In the add inbound security rule dialog, type or select the following:

* Source: Select Service Tag.
* Source service tag: Select Internet.
* Destination port ranges: Type 80.
* Protocol: Select TCP.
* Action: Select Allow.
* Priority: Type 100.
* Name: Type AllowHTTP.
* Description: Type Allow HTTP.

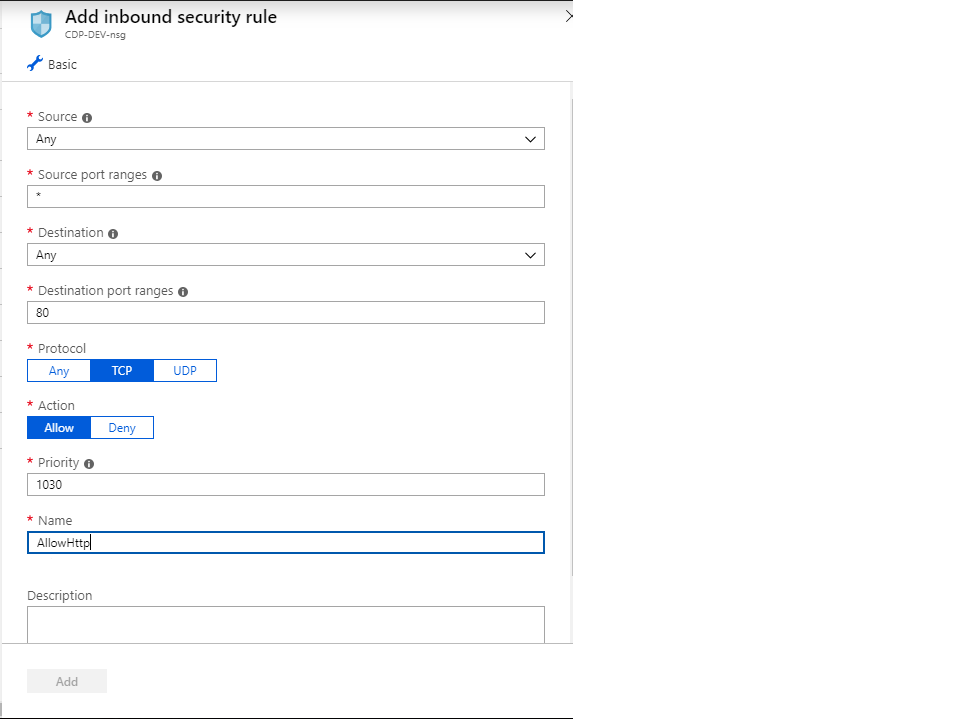
1. Select Add as in Figure 3

Figure 3: Create inbound security rule

# VPN gateway

A VPN gateway is a specific type of virtual network gateway that is used to send encrypted traffic between an Azure virtual network and an on-premises location over the public Internet. You can also use a VPN gateway to send encrypted traffic between Azure virtual networks over the Microsoft network.

* + 1. Create Virtual Network Gateway

In the current setup, VPN gateway is used for connecting to VMs in private network. To create virtual network gateway in Azure from a browser, navigate to the Azure portal

1. Click create a resource. In the search the marketplace field, type 'virtual network'. Locate virtual network from the returned list and click to open the virtual network page.
2. Near the bottom of the virtual network page, from the select a deployment model list, verify that Resource Manager is selected from the dropdown, and then click Create. This opens the Create virtual network page.
3. On the create virtual network page, configure the VNet settings. When you fill in the fields, the red exclamation mark becomes a green check mark when the characters entered in the field are valid.

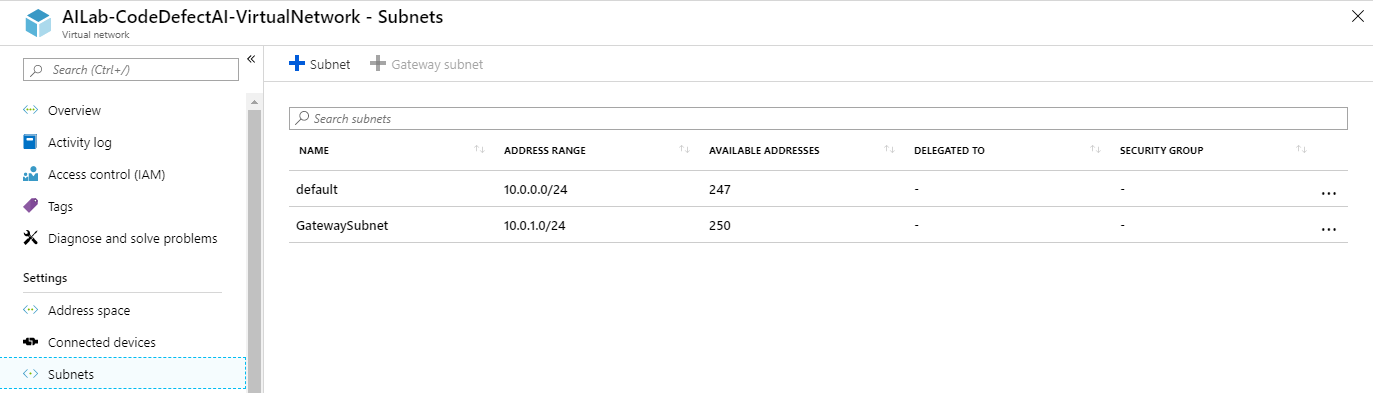
* Name: AILab-CodeDefectAI-VirtualNetworkGateway
* Address space: 10.0.1.0/24
* Subscription: Microsoft AI Lab
* Resource group: AILab-CodeDefectAI
* Location: West US
* Subnet: Frontend
* Address range: 10.1.0.0/24

1. Click create.
   * 1. Add a Gateway subnet

The gateway subnet contains the reserved IP addresses that the virtual network gateway services use. create a gateway subnet

Configure gateway settings:

1. In the portal, navigate to the virtual network for which you want to create a virtual network gateway.
2. On your virtual network page, click Subnets to expand VNet1 - subnets page.
3. Click gateway subnet at the top to open the add subnet page.
4. The name for your subnet is automatically filled in with the required value 'GatewaySubnet'.
5. To create the gateway subnet, click OK at the bottom of the page.



**Figure 4: Gateway Subnet creation**

* + 1. Generate certificates for Point to Site (P2S) connection

Certificates are used by Azure to authenticate clients connecting to a VNet over a Point-to-Site VPN connection. Once you obtain a root certificate, you upload the public key information to Azure. The root certificate is then considered 'trusted' by Azure for connection over P2S to the virtual network. You also generate client certificates from the trusted root certificate, and then install them on each client computer. The client certificate is used to authenticate the client when it initiates a connection to the VNet. We can generate these certificates through PowerShell as follows:

* + 1. Create a self-signed root certificate

Use the New-SelfSignedCertificate cmdlet to create a self-signed root certificate.

1. From a computer running Windows 10 or Windows Server 2016, open a Windows PowerShell console with elevated privileges. These examples do not work in the Azure Cloud Shell "Try It". You must run these examples locally.
2. The command below creates a self-signed root certificate named 'P2SRootCert' that is automatically installed in 'Certificates-Current User\Personal\Certificates'. You can view the certificate by opening *certmgr.msc* or *Manage User Certificates*.

$cert = New-SelfSignedCertificate -Type Custom -KeySpec Signature -Subject "CN=AILabRootCertificate" -KeyExportPolicy Exportable -HashAlgorithm sha256 -KeyLength 2048 -CertStoreLocation "Cert:\CurrentUser\My" -KeyUsageProperty Sign -KeyUsage CertSign

1. To create client certificate run following command in PowerShell after generating root certificate:

New-SelfSignedCertificate -Type Custom -DnsName AILabClientCertificate -KeySpec Signature `

-Subject "CN= AILabClientCertificate " -KeyExportPolicy Exportable `

-HashAlgorithm sha256 -KeyLength 2048 `

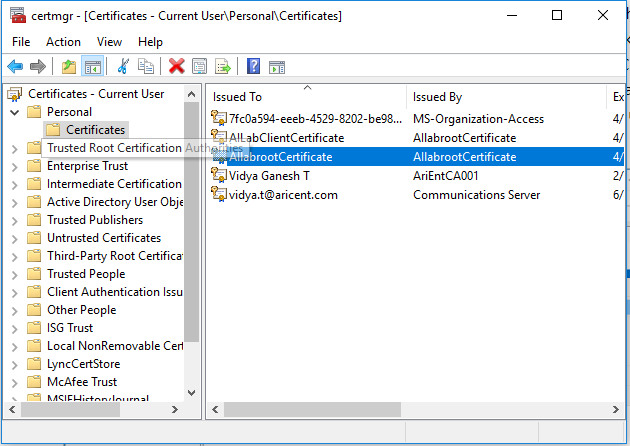
-CertStoreLocation "Cert:\CurrentUser\My" `

-Signer $cert -TextExtension @("2.5.29.37={text}1.3.6.1.5.5.7.3.2")

* + 1. Export the root certificate public key

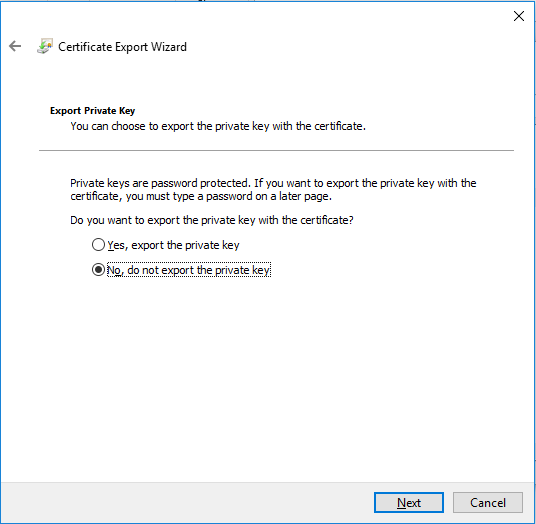
After creating a self-signed root certificate, export the root certificate public key .cer file (not the private key). You will later upload this file to Azure. The following steps help you export the .cer file for your self-signed root certificate:

1. To obtain .cer file from the certificate, open manage user certificates. Locate the self-signed root certificate, typically in 'Certificates - Current User\Personal\Certificates', and right-click. Click all tasks, and then click Export. This opens the Certificate Export Wizard.
2. If you can't find the certificate under Current User\Personal\Certificates, you may have accidentally opened "Certificates - Local Computer", rather than "Certificates - Current User"). If you want to open Certificate Manager in current user scope using PowerShell, you type certmgr in the console window.



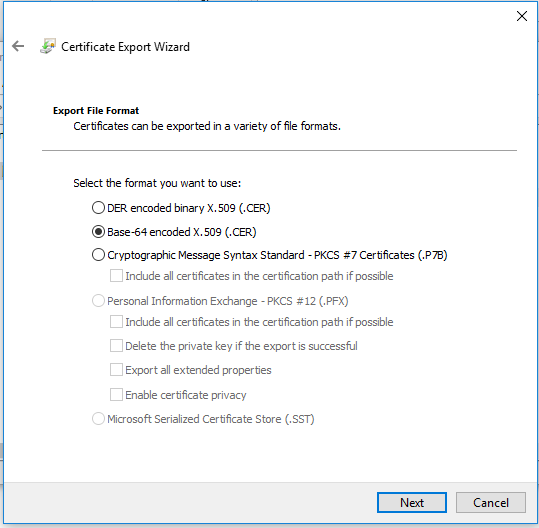
**Figure 5: Certificate Manager window**

1. In the Wizard, click Next as in **Figure 6**
2. Select No, do not export the private key, and then click Next.



**Figure 6: Export Private Key Screen**

1. On the Export File Format page as in **Figure 7** , select Base-64 encoded X.509 (.CER)., and then click Next.



**Figure 7: Certificate Export Format**

1. For file to export, browse to the location to which you want to export the certificate. For File name, name the certificate file. Then, click Next.
2. Click Finish to export the certificate and the certificate will be successfully exported.
3. If you open the exported certificate using Notepad as in **Figure 8**, the content of the file will look similar as the following file content. The section in blue contains the information that is uploaded to Azure. If you open your certificate with Notepad and it does not look like the following content, typically this means you did not export it using the Base-64 encoded X.509(.CER) format. Additionally, if you want to use a different text editor, understand that some editors can introduce unintended formatting in the background. This can create problems when uploaded the text from this certificate to Azure.

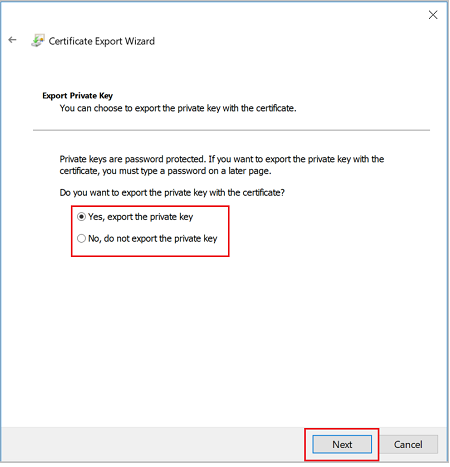


**Figure 8: Key**

* + 1. Export the client certificate

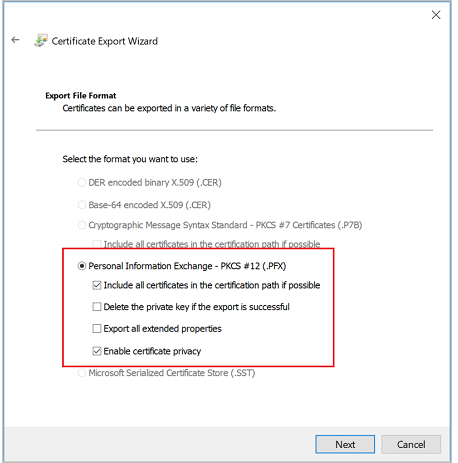
When you generate a client certificate, it's automatically installed on the computer that you used to generate it. If you want to install the client certificate on another client computer, you need to export the client certificate that you generated.

1. To export a client certificate, open Manage user certificates as in **Figure 5**. The client certificates that you generated are, by default, located in 'Certificates - Current User\Personal\Certificates'. Right-click the client certificate that you want to export, click all tasks, and then click Export to open the Certificate Export Wizard.
2. In the Certificate Export Wizard, click Next to continue.
3. Select Yes, export the private key, and then click Next as in **Figure 9**



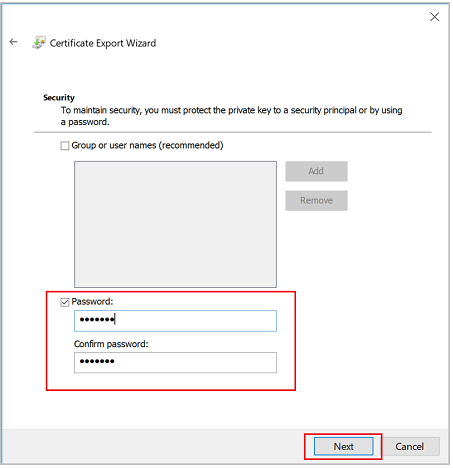
**Figure 9: Certificate Export Wizard**

1. On the Export File Format page as in **Figure 10**, leave the defaults selected. Make sure that Include all certificates in the certification path if possible is selected. This setting additionally exports the root certificate information that is required for successful client authentication. Without it, client authentication fails because the client doesn't have the trusted root certificate. Then, click Next.



**Figure 10: Export File Format Screen**

1. On the Security page as in **Figure 11**, you must protect the private key. If you select to use a password, make sure to record or remember the password that you set for this certificate. Then, click Next.

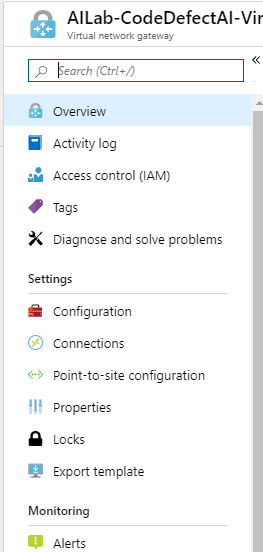


**Figure 11: Security Screen**

1. On the File to export, browse to the location to which you want to export the certificate. For file name, name the certificate file. Then, click Next.
2. Click finish to export the certificate.
3. Install the client certificate in the system which needs to be connected to VPN.
   * 1. Configure gateway settings to create Point to Site (P2S) connection

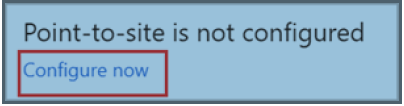
The client address pool is a range of private ip addresses that you specify. The clients that connect over a Point to Site VPN dynamically receive an ip address from this range. Use a private ip address range that does not overlap with the on-premises location that you connect from, or the VNet that you want to connect to.

1. Once the virtual network gateway has been created, navigate to the Settings section of the virtual network gateway page. In the Settings section, click Point-to-site configuration as in **Figure 12**



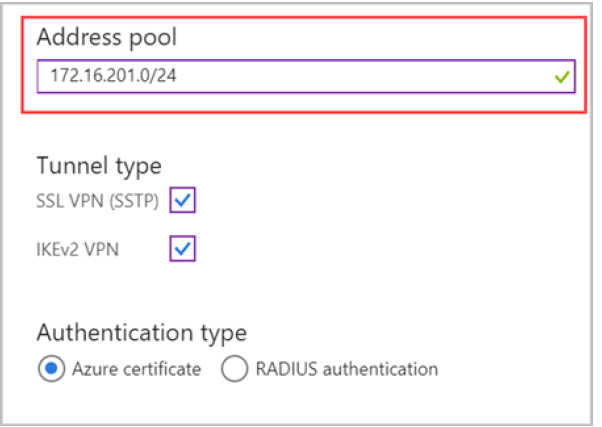
**Figure 12: Point to Site Configuration Settings**

1. Click Configure now to open the configuration page.



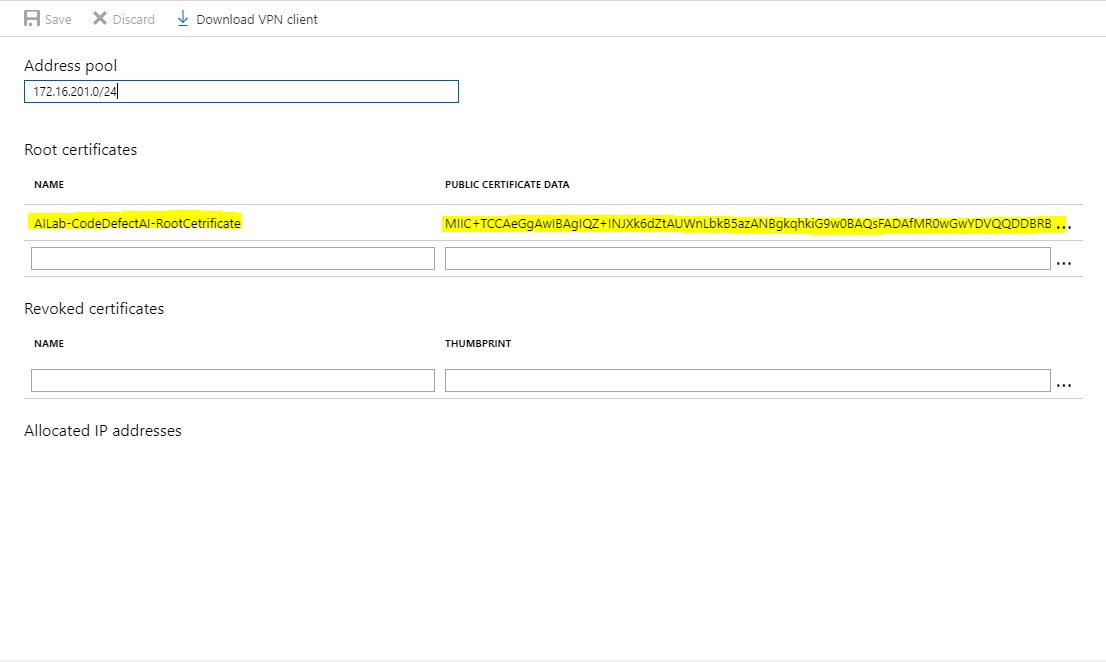
**Figure 13: Configure Point to Site**

1. On the Point-to-site configuration page, in the address pool box, add the private ip address range that you want to use as in **Figure 14**. VPN clients dynamically receive an ip address from the range that you specify. Click save to validate and save the setting.



**Figure 14: Address Pool – IP setting**

1. Upload the root certificate public certificate data by entering the certificate data from AILabRootCertificate.cer file as shown below.



**Figure 15: Details of Certificate**

1. Click on save & then click on download VPN client and install once its downloaded.
2. Now we will be able to connect to the private network through the VPN client which uses client certificate for authentication.

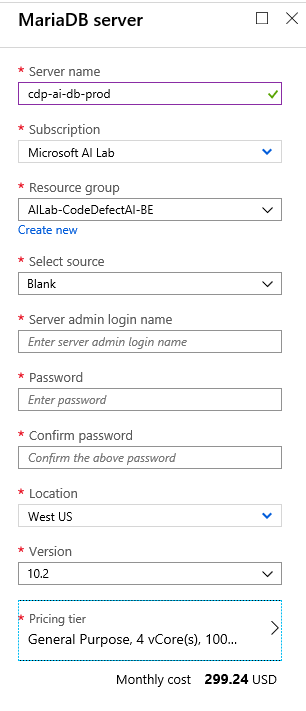
Environment Setup

# Prerequisites

Virtual Machines should have Python 3 version installed which can be obtained as part of Anaconda setup.

# Database setup

1. Go to the respective resource group under which database instance needs to be created under resource Group blade
2. Click on the resource group (“AILab-CodeDefectAI-BE”)
3. Under the resource group click add, it will take you to market place, search “Azure Database for MariaDB” as in **Figure 16**
4. After Selecting Azure Database for MariaDB, click on create.
5. Provide appropriate name, server admin login name, password, location and select database version as 10.3
6. Select appropriate pricing tier based on number of cores and disk size.



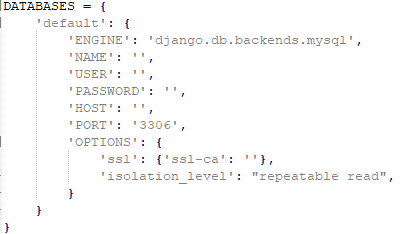
**Figure 16: RDS Service Creation - MariaDB**

1. After database is created. Go to the database resource.
2. Go to replication blade under the database. Add a replication server by providing the replication database name.

Note: After the server is created, check the connectivity using MySQL client and by adding your local IP to rule under connection security blade.

After Maria DB server is created and before creating the Code Defect AI database, follow the following steps to create Django API Authentication token.

1. Download the SSL certificate for connecting to the MariaDB database through Django API and copy to the appropriate directory and update the same in settings.py file
2. Connect to the database using mysql client from any machine and create the Code Defect AI Database.
3. Update the following database details in Django API settings.py file. Update ssl-ca path as */cdpweb/BaltimoreCyberTrustRoot.crt.pem* (this is because once the image is created this is where certificate will be placed.)



**Figure 17: DB settings in Django**

1. Update SECRET\_KEY in settings.py. SECRET\_KEY usually auto-populated when a new Django project is created but since in this case we will be creating image from this code itself, we’ll need to update it manually.



**Figure 18: Secret key format**

1. Update application insights instrumentation key under APPLICATION\_INSIGHTS details.



**Figure 19: Application insights instrumentation key format**

1. Download the database connectivity certificate from <https://www.digicert.com/CACerts/BaltimoreCyberTrustRoot.crt.pem> *(Refer Docs URL:* [*https://docs.microsoft.com/en-us/azure/mariadb/howto-configure-ssl*](https://docs.microsoft.com/en-us/azure/mariadb/howto-configure-ssl) *for details*.*)* and replace the placeholder certificate file with this certificate in following two locations:

*core/certificate/BaltimoreCyberTrustRoot.crt.pem*

*restservices/api/BaltimoreCyberTrustRoot.crt.pem*

1. Run following commands

* python manage.py makemigrations
* python manage.py migrate
* python manage.py createsuperuser --username <username> --email <email>
* python manage.py drf\_create\_token <username>

Once authentication token is generated execute CDP AI database creation script and initial data insert script for project name, project summary and scheduler table. You can find these scripts inside the code.

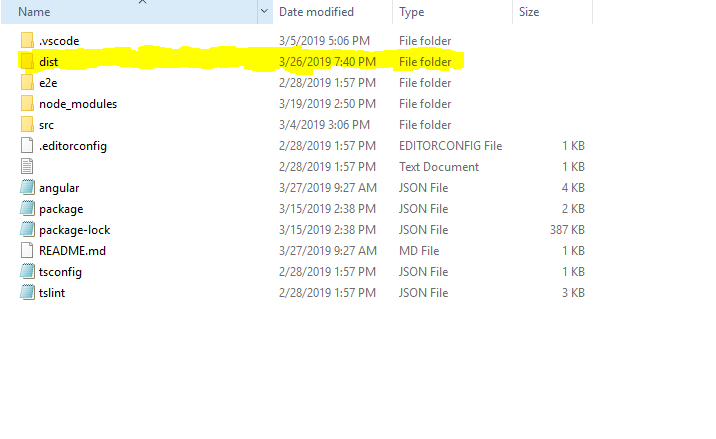
# Deployment

## UI Code Deployment

For Creating UI Docker Image

1. Open the UI Code using Visual Studio Code
2. Update authentication token generated while setting up DB in app/Helper/token-interceptor.ts
3. Open the Visual Studio code terminal and run the following command to generate deployable bits.

ng build –prod

1. In the file structure a folder called “dist” will be generated as in **Figure 18** which will contain the deployable bits

**Figure 20: UI code directory**

1. Copy folder of the generated bits inside ‘dist’ folder and move to the Docker VM server using any FTP Client tool like WinSCP

Directory structure



1. Prepare Docker file as

# First stage image labelled as node-angular-cli

1. FROM nginx:latest
2. LABEL authors="Altran"
3. # Copying dist folder
4. COPY dist/ /var/www/
5. RUN apt-get update -y
6. RUN apt-get upgrade -y
7. COPY default /etc/nginx/conf.d/default.conf
8. EXPOSE 80
9. CMD ["nginx", "-g", "daemon off;"]

You can also find dockerfile inside the code.

4. Build the Image as following

sudo docker build -t ‘<Azure Container Registry>/< Image Repository Name >/<Image Name>’ <dockerfile path. “.” If dockerfile is in same directory>

e.g., sudo docker build -t aicdpdockerprod.azurecr.io/codedefectai/cdpui .

4. Login to Azure Container Service using following command

sudo docker login <Azure container Registry>

eg., sudo docker login aicdpdockerprod.azurecr.io

5. User name and password of the container are present under Azure container registry, access keys azure blade.

6. push the docker image to Azure container registry using following command

sudo docker push <Image Name>

**For creating webapp for deploying UI (from local machine powershell)**

**# Login to Azure Tenet**

*az login --tenant microsoft.onmicrosoft.com*

**# Create an App Service plan**

*az appservice plan create --name <app service plan name> --resource-group <resource group name> --sku S2 --is-linux -l westus*

*az webapp create --resource-group <resource group name> --plan <app service plan nama> --name codedefectai --deployment-container-image-name <UI code image name>*

**# Configuring Azure container username and password for pulling docker image from azure. User name and password can be found under Azure container registry, access keys azure blade.**

*az webapp config container set --name codedefectai --resource-group <resource group name> --docker-custom-image-name <UI docker image name > --docker-registry-server-url < Azure container registry server URL> --docker-registry-server-user <Azure container registry user name>--docker-registry-server-password <Azure container registry password>*

**# Set port to value specified in container app**

*az webapp config appsettings set --resource-group <resource group name> --name codedefectai --settings WEBSITES\_PORT=80*

## Backend - prediction code setup

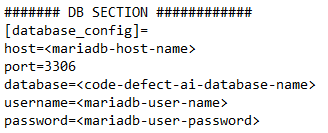
1. Copy the backend code to Docker VM using WinSCP or any other FTP tool.
2. Unzip TransientData.zip folder in the same directory using the following command.

*unzip TransientData.zip*

1. Update cronfile in case timing of DailyTaskScheduler needs to be changed. Entries in cronfile are in linux crontab format.

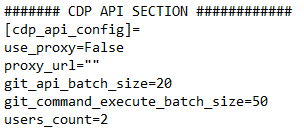


1. Update database host/connection IP, Port, database name, username and password in Config/cdp.ini file



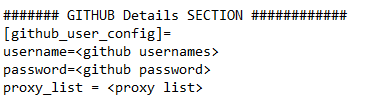
**Figure 21: DB settings**

1. Update API batch size git\_command\_execute\_batch\_size and proxy url, if any, for internet access in Config/cdp.ini. Our code uses Asyncio to provide concurrency for faster processing. Git API batch size refers to number of requests that will be made to github APIs simultaneously, and git\_command\_execute\_batch\_size refers to number of git commands that will be run simultaneously on local git repo to fetch data, using asyncio.



**Figure 22: connection settings**

1. Update github username, password and proxy list, if any. Enter github usernames separated by comma, without any spaces. E.g., user1,user2,user3 etc .Update github password for those users. Currently system is configured to use common password for all the users. Please note that for normal user, github has an API limit of 5000 requests per hour.



**Figure 23: github user settings**

1. Create Image from dockerfile that you’ll find inside code directory.

*sudo docker build -t ‘<Azure Container Registry>/< Image Repository Name >/<Image Name>’ <dockerfile path. “.” If dockerfile is in same directory>*

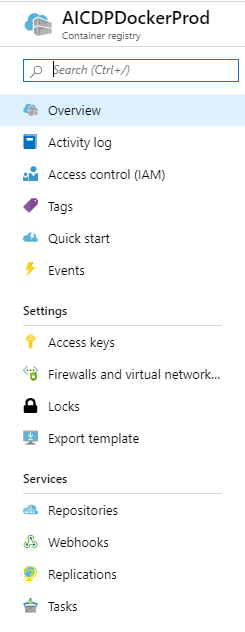
1. Login to Azure Container Service using following command

*sudo docker login <Azure container registry name>*

1. Push the image to Azure Container registry

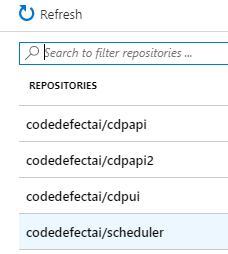
*sudo docker push <image-name>*

1. Once you have the image in container registry. Create a container from the image using the following steps: Open Container registry and click on Repositories under services heading as in **Figure 21**



**Figure 24: Container Registry**

* 1. Click on the image name that was pushed.



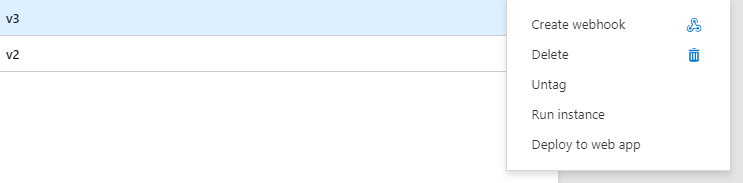
**Figure 25: Images In Docker Registry**

* 1. Click on the context menu with respect to tag of the pushed image as in **Figure 23**



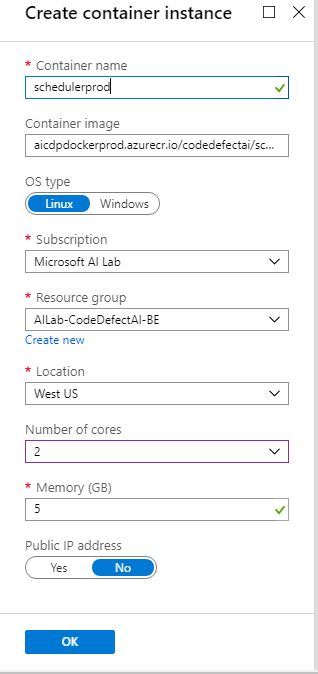
**Figure 26: Specific Image Tags**

* 1. Click on run instance.



**Figure 27: Run Specific Image on Container**

* 1. Create container with the details as in



**Figure 28: Create Container Instance**

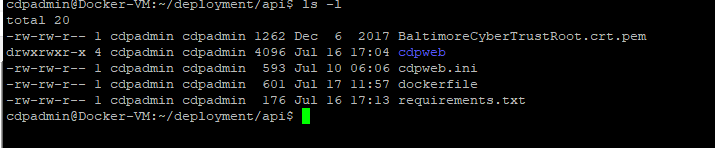
* 1. Once the container is created. You can see it in selected resource group. It will run the scheduler process automatically as per the timings specified in cronfile file.

## API Code Deployment

For Creating Django API Docker Image

1. Create Deployment directory in Docker VM which is present in the Test Resource Group

2. Copy the API Code, requirement.txt file and create the directory structure as in



**Figure 29: API code structure**

3. Update ALLOWED\_HOSTS in restservices\api\cdpweb\cdpweb\settings.py file with URL of the UI webapp.



**Figure 30: Allowed Hosts Field**

4. Prepare Docker file as

1. FROM ubuntu:18.04
2. RUN apt-get update -y
3. RUN apt-get install -y apt-utils
4. RUN apt-get install -y python3-pip
5. RUN apt-get install -y python3-dev
6. RUN apt-get install -y python3-mysqldb
7. RUN apt-get install -y libmysqlclient-dev
8. RUN mkdir /cdpweb
9. ADD . /cdpweb/
10. COPY ./requirements.txt /cdpweb/requirements.txt
11. COPY ./BaltimoreCyberTrustRoot.crt.pem /cdpweb/BaltimoreCyberTrustRoot.crt.pem
12. COPY . /cdpweb
13. WORKDIR /cdpweb
14. RUN pip3 install -r requirements.txt
15. EXPOSE 5000
16. CMD [ "python3", "cdpweb/manage.py", "migrate" ]
17. ENTRYPOINT [ "python3" ]
18. CMD [ "cdpweb/manage.py", "runserver", "0.0.0.0:5000" ]

5. Build the Image as following

*sudo docker build -t ‘<Azure Container Registry>/< Image Repository Name >/<Image Name>’ <dockerfile path. “.” If dockerfile is in same directory>*

6. Login to Azure Container Service using following command

*docker login <Azure container registry name>*

1. push the docker image to Azure container registry using following command

*docker push <Azure Container Registry>/<Image Repository Name>/<Image Name>*

**For Deploying API (from local machine powershell)**

**# Login to Azure Tenant**

*az login --tenant microsoft.onmicrosoft.com*

**# Create an App Service plan**

*az appservice plan create --name <app service plan name> --resource-group <resource group name> --sku P2V2 --is-linux -l westus*

**# Login to Docker Registry**

*az acr login --name <Azure container Registry user name>*

**# Create a web app**

*az webapp create --resource-group <resource group name> --plan <app service plan name> --name codedefectaiapi --deployment-container-image-name <deployment-container-API-image-name >*

**# Configuring Azure container username and password for pulling docker image from azure.** **User name and password can be found under Azure container registry, access keys azure blade.**

*az webapp config container set --name codedefectaiapi --resource-group <resource group name> --docker-custom-image-name <docker-custom-API - image-name > --docker-registry-server-url <docker-registry-server-url >--docker-registry-server-user <Azure container registry user name> --docker-registry-server-password <Azure container registry password>*

**# Set port to value specified in container app**

*az webapp config appsettings set --resource-group <resource group name> --name codedefectaiapi--settings WEBSITES\_PORT=5000*

## Data Cleanup

As part of data cleanup process, to delete old data from database, a database event has been created. The event is a weekly job that runs as per the time setup during event creation.

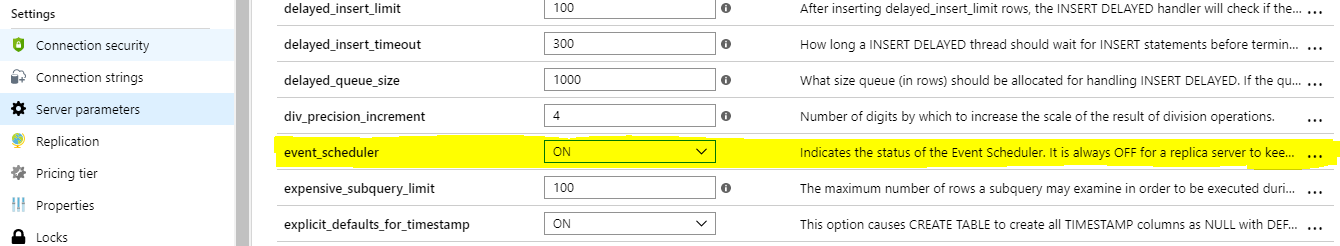
### Data Cleanup Process

All the tables that are to be cleaned, have a column called ‘archived’ which is set as 0 by default. For the data that is older than 15 Days in those tables, ‘archived’ is set to 1 and data older than 45 days and having ‘archived’ set as 1, is permanently deleted.

The number of days (15 and 45) are not fixed and can be changed while creating the event.

### Steps to create an event

1. Set event\_scheduler as ON under in Server Parameters in MariaDb Settings blade.



**Figure 31: Azure Maria DB Event Scheduler Setting**

2. Log onto Maria db

**#Using WinSCP or any other tool, copy the *datacleanup.sql* script from code to server.**

**#Connect to mariadb instance**.

*sudo mysql -h <mariadb server name> -u <server admin login name> --ssl-ca=<path-to-Baltimore-certi/BaltimoreCyberTrustRoot.crt.pem> -p<password>*

***#*Change database to CodeDefectAI database**

*Use <database name>*

***#*Run datacleanup.sql script with the following command:**

*source </path/to/script>/datacleanup.sql*

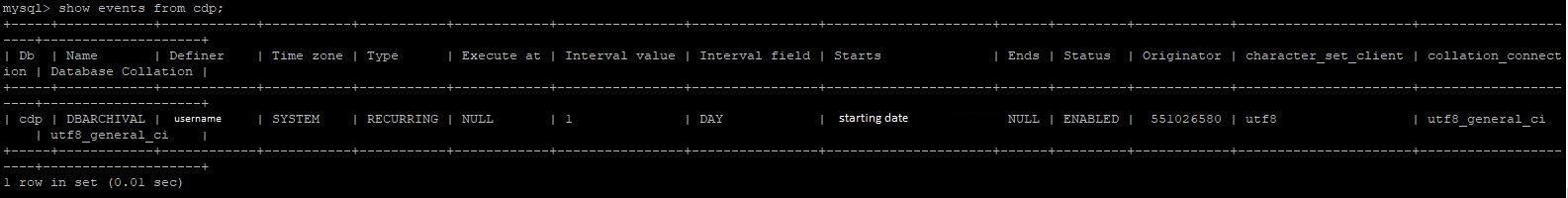
**#It should give you an output like:**

Query OK, 0 rows affected (0.03 sec)

3. Once event has been created, you can check the event using below command.

*show events from <database name>*

It will give output like:



**Figure 32: show events command output**

# Appendix

## Infrastructure Details

There are two separate Azure resource groups, one for User interface and another for API and backend.

1. AILab-CodeDefectAI
2. AILab-CodeDefectAI-BE

After moving to WebApp Services, we are using the following resources:

|  |  |  |  |
| --- | --- | --- | --- |
| Resource Name | Resource  Type | Description | Resource Group |
| ailabcodedefectaidiag | Storage account | UI Resource group storage account | [AILab-CodeDefectAI](https://ms.portal.azure.com/#@72f988bf-86f1-41af-91ab-2d7cd011db47/resource/subscriptions/9d010d9f-ac48-4bc2-b5ec-bd0481011ed7/resourceGroups/AILab-CodeDefectAI/overview) |
| AILab-CodeDefectAI-VirtualNetwork | Virtual network | To connect to Azure resources | [AILab-CodeDefectAI](https://ms.portal.azure.com/#@72f988bf-86f1-41af-91ab-2d7cd011db47/resource/subscriptions/9d010d9f-ac48-4bc2-b5ec-bd0481011ed7/resourceGroups/AILab-CodeDefectAI/overview) |
| AILab-CodeDefectAI-VirtualNetworkGateway | Virtual network gateway | To connect to Azure virtual network | [AILab-CodeDefectAI](https://ms.portal.azure.com/#@72f988bf-86f1-41af-91ab-2d7cd011db47/resource/subscriptions/9d010d9f-ac48-4bc2-b5ec-bd0481011ed7/resourceGroups/AILab-CodeDefectAI/overview) |
| AILab-CodeDefectAI-VPN-Gateway-Ip | Public IP address | Public IP attached to VPN Gateway | [AILab-CodeDefectAI](https://ms.portal.azure.com/#@72f988bf-86f1-41af-91ab-2d7cd011db47/resource/subscriptions/9d010d9f-ac48-4bc2-b5ec-bd0481011ed7/resourceGroups/AILab-CodeDefectAI/overview) |
| codedefectai | App Service plan | UI web app service plan | [AILab-CodeDefectAI](https://ms.portal.azure.com/#@72f988bf-86f1-41af-91ab-2d7cd011db47/resource/subscriptions/9d010d9f-ac48-4bc2-b5ec-bd0481011ed7/resourceGroups/AILab-CodeDefectAI/overview) |
| codedefectai | App Service | UI web app service | [AILab-CodeDefectAI](https://ms.portal.azure.com/#@72f988bf-86f1-41af-91ab-2d7cd011db47/resource/subscriptions/9d010d9f-ac48-4bc2-b5ec-bd0481011ed7/resourceGroups/AILab-CodeDefectAI/overview) |
| virtual\_network\_gateway\_nsg | Network security group | Virtual network security group to filter network traffic to and from Azure gateway | AILab-CodeDefectAI |
| AICDPDockerProd | Container registry | Container Registry to store all images | AILab-CodeDefectAI-BE |
| webappcodedefectai (AICDPDockerProd/webappcodedefectai) | Container registry webhook | UI repository webhook | AILab-CodeDefectAI-BE |
| webappcodedefectaiapi (AICDPDockerProd/webappcodedefectaiapi) | Container registry webhook | API repository webhook | AILab-CodeDefectAI-BE |
| ailabcodedefectaibediag | Storage account | Backend Resource group storage account | AILab-CodeDefectAI-BE |
| cdp-ai-db-prod | Azure Database for MariaDB server | Production database | AILab-CodeDefectAI-BE |
| codedefectaiapi | App Service plan | Django API App service plan | AILab-CodeDefectAI-BE |
| codedefectaiapi | App Service | Django API App service | AILab-CodeDefectAI-BE |
| Code-Defect-AI-App-Insights | Application Insights | analytics tools to help analyse webapp data, diagnose issues and to understand user interaction with the app | AILab-CodeDefectAI-BE |
| schedulerprod-v3 | Container instances | Container where Scheduler code is running for prediction | AILab-CodeDefectAI-BE |

Temporary Resources:

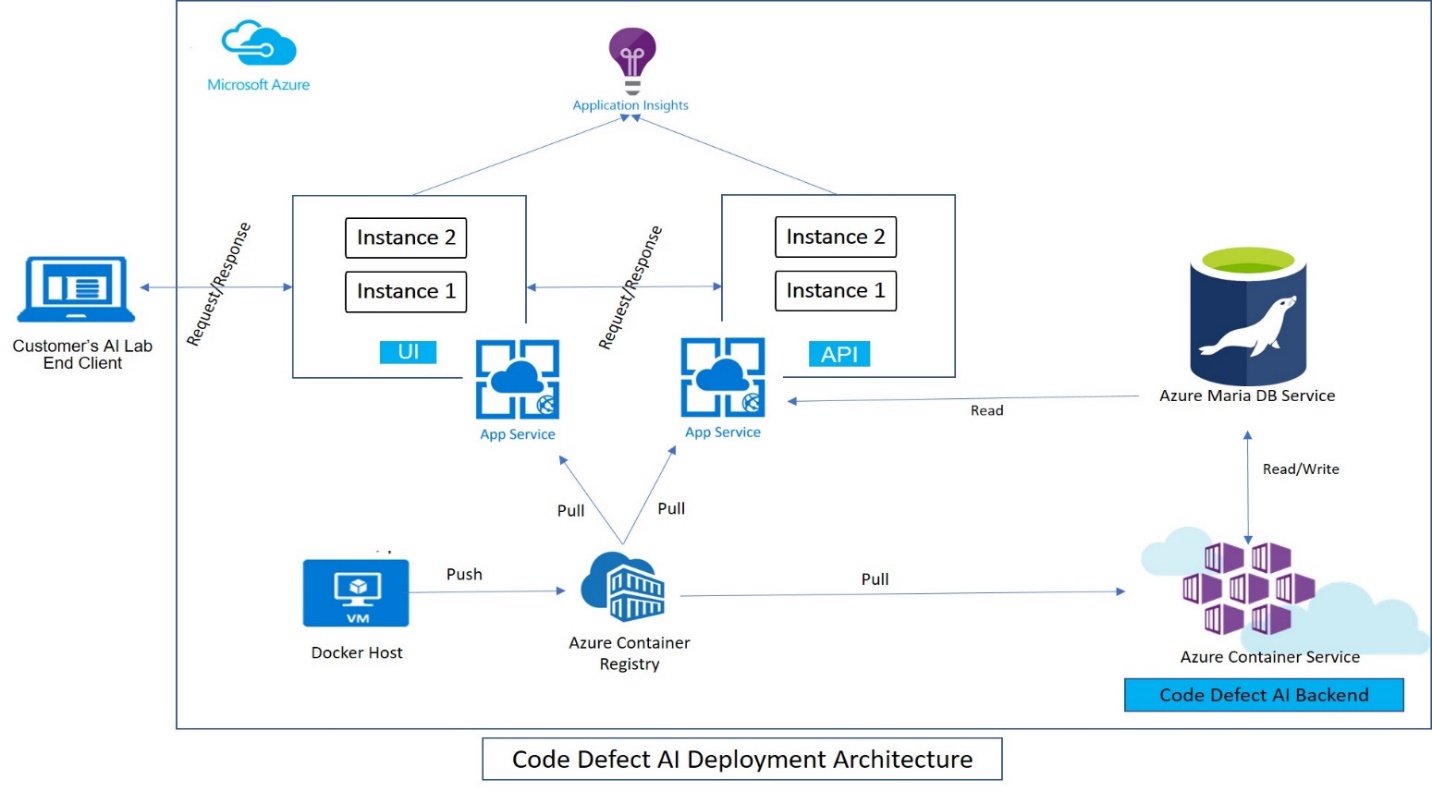
|  |  |  |  |
| --- | --- | --- | --- |
| Resource Name | Resource  Type | Description | Resource Group |
| Docker-VM | Virtual Machine | This resource is needed on a temporary basis to run Docker Scripts, create image and push images to container registry | AILab-CodeDefectAI-Test |

## Configurations used

For the deployment of CodeDefectAI following configurations were used for above mentioned infrastructure components.

|  |  |  |
| --- | --- | --- |
| Resource Name | Resource Type | Configuration Used |
| ailabcodedefectaidiag | Storage account | Storage (general purpose v1) with Locally-redundant storage (LRS) |
| ailabcodedefectaibediag | Storage account | Storage (general purpose v1) with Locally-redundant storage (LRS) |
| cdp-ai-db-prod | Azure Database for MariaDB server | General Purpose, 4 vCore(s), 60 GB |
| AICDPDockerProd | Container registry | SKU:Premium |
| codedefectai | App Service plan | P1v2: 2 (app service plan is configured to instantiate 2 P1V2 type Linux Server instances for the corresponding WebAPP ) |
| codedefectaiapi | App Service plan | P2V2:2 (app service plan is configured to instantiate 2 P2V2 type Linux Server instances for the corresponding WebAPP) |
| schedulerprod-v3 | Container Instance | 2 vCPUs, 5 GiB Memory |
| Docker-VM | Virtual Machine | Standard D2s v3 (2 vcpus, 8 GiB memory) |

## Network Topology Diagram



**Figure 33: Code Defect AI Deployment Architecture**