Chain Core Developer Edition

Contents

[Overview 2](#_Toc484622677)

[About Chain Core 3](#_Toc484622678)

[About Blockchain 3](#_Toc484622679)

[Solution Architecture Diagrams 4](#_Toc484622680)

[Single Generator, Multi Signer, Single Region 4](#_Toc484622681)

[Description 4](#_Toc484622682)

[Workflow 5](#_Toc484622683)

[Single Generator, Multi Signer, Multi Region 5](#_Toc484622684)

[Description 5](#_Toc484622685)

[Workflow 6](#_Toc484622686)

[Prerequisites 6](#_Toc484622687)

[Creating an AAD app registration (service principal) 6](#_Toc484622688)

[Create an Azure Active Directory (AAD) application 7](#_Toc484622689)

[Get Application ID and Authentication key 9](#_Toc484622690)

[Getting Started 11](#_Toc484622691)

[Post Deployment 21](#_Toc484622692)

[Getting access to the deployed key vault 21](#_Toc484622693)

[Accessing the values from the key vault 24](#_Toc484622694)

[Accessing the VM Dashboards 27](#_Toc484622695)

[Useful Links 34](#_Toc484622696)

# Overview

This template is capable of deploying single or multi region based Chain Core network based on user’s selection. It deploys virtual machines running the Ubuntu Linux operating system with Chain Core Docker container. This template also deploys the Azure Key vault, Managed Disks and VNET to VNET connectivity between regions to regions using VNET gateways.

# About Chain Core

**Chain Core** is software designed to operate and connect to highly scalable permissioned blockchain networks conforming to the Chain Protocol. Each network maintains a cryptographically-secured transaction log known as a “blockchain”, which allows participants to define, issue, and transfer digital assets on a multi-asset shared ledger. Digital assets share a common, interoperable format and can represent any units of value that are guaranteed by a trusted issuer — such as currencies, bonds, securities, IOUs, or loyalty points. Each Chain Core holds a copy of the ledger and independently validates each update, or “block,” while a federation of block signers ensures global consistency of the ledger.

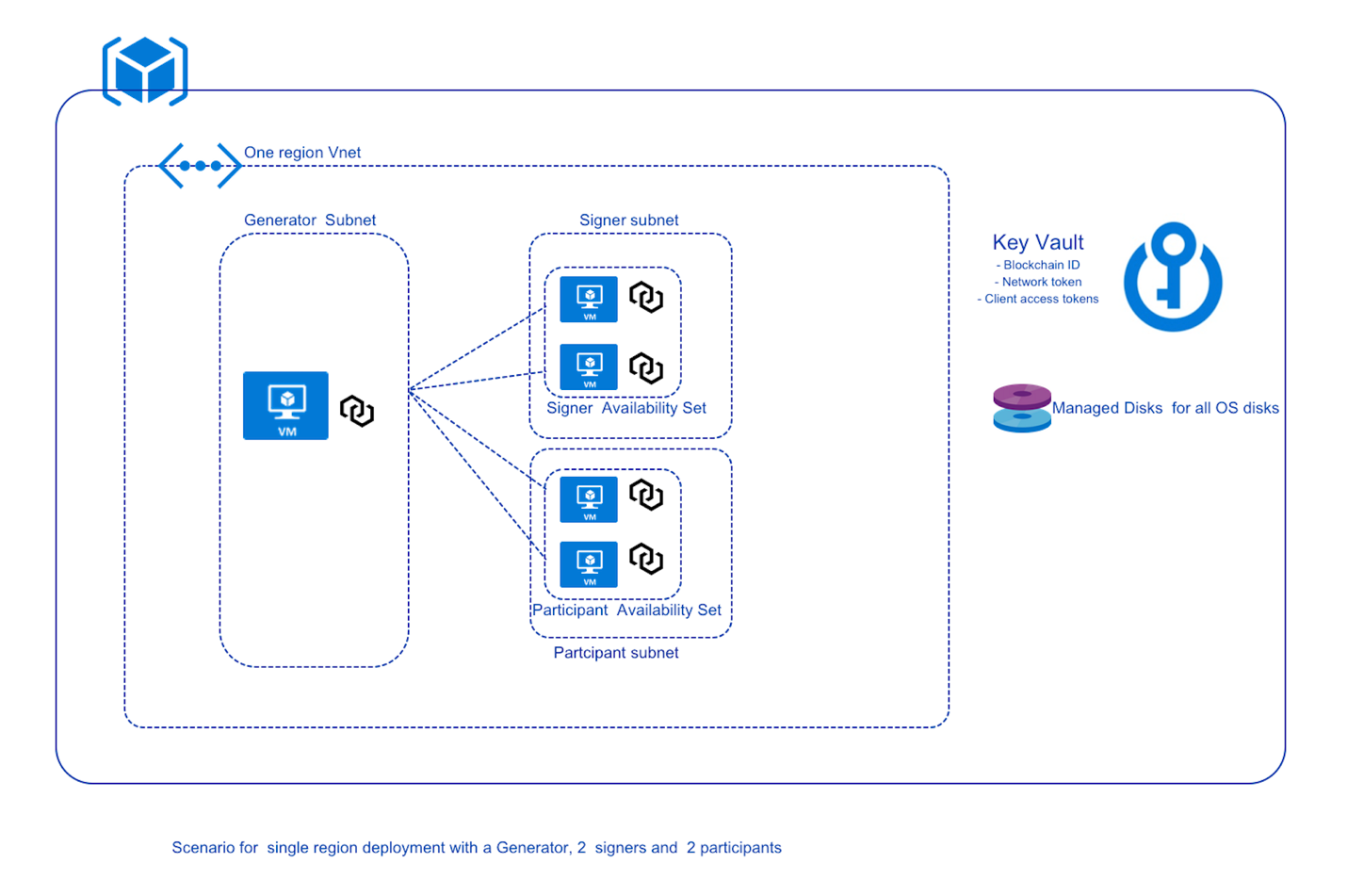
**Chain Core Developer Edition** is a free, downloadable version of Chain Core that is open source and licensed under the AGPL. Individuals and organizations use Chain Core Developer Edition to learn, experiment, and build prototypes.

# About Blockchain

Blockchain is the underlying technology behind Bitcoin; however, it is much more than just a virtual currency. It is a composite of existing database, distributed system, and cryptographic technologies that enables secure multi-party computation with guarantees around immutability, verifiability, auditability, and resiliency to attack. Different implementations employ different mechanisms to provide these attributes.

# Solution Architecture Diagrams

# Single Generator, Multi Signer, Single Region



## Description

All instances below are launched via a docker container and run on a linux virtual machine (Ubuntu Canonical 16.04) that is provisioned by the deployment.

1. A Chain Core block generator instance that reside in a block generator subnet for appropriate network isolation.
2. One to three Chain Core signer instances that on individual virtual machines and collectively reside in a subnet dedicated to signer instances.

One to three Chain Core participant instances that on individual virtual machines and collectively reside in a subnet dedicated to signer instances.

1. A key vault to store the blockchain ID, network and client tokens of all the chain core instances. Key vault is used to allow distribution of keys during deployment needed by signers and participants to connect to the block generator as well as for user to get client tokens needed to access the dashboard of each instance.
2. Virtual machines in this solution use Azure Managed disks to automate OS disk management.

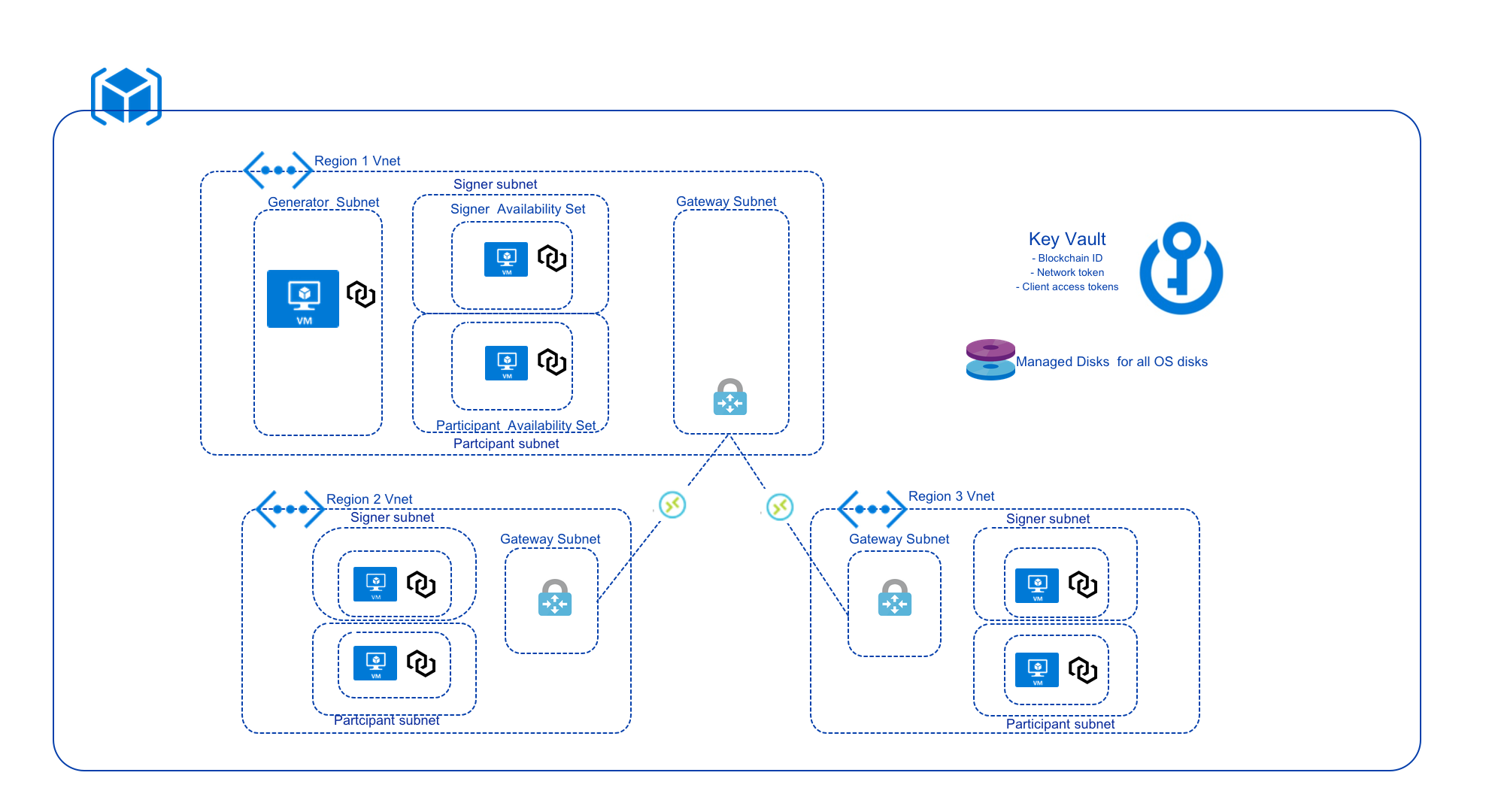
## Workflow

The blockchain ID and network token for signers are stored in key vault from generator chain core instance.

The Signer and participant chain core instances retrieve blockchain ID and network token from the key vault and configures block signer and block participant VMs.

The Signer and participant chain core instances store their respective client tokens in key vault.

# Single Generator, Multi Signer, Multi Region



## Description

1. A Generator VM (Ubuntu Canonical 16.04) which launches Chain Core in port 1999 through Chain Core Docker container.
   1. This VM is configured as a Block generator using a shell script as an extension to this VM.
   2. This VM is provisioned in a separate subnet i.e. generator subnet.
   3. It is deployed only in the first region.
2. Next is the Signer VM (Ubuntu Canonical 16.04) which launches Chain Core in port 1999 through Chain Core Docker container.
   1. User has the option to provision up to 3 block signer VMs in the first region and 1 block signer VM in the second and third regions.
   2. These VMs are configured as Block signers using a shell script as an extension to the VM.
   3. These VMs are provisioned in a separate subnet i.e. signer and participant subnet.
   4. It is deployed in 3 regions based on the number of regions selected by User.

3 Next is the Participant VM (Ubuntu Canonical 16.04) which launches Chain Core in port 1999 through Chain Core Docker container.

* 1. User has the option to provision up to 3 participant VMs in the first region and 1 participant VM the in second and third regions.
  2. These VMs are configured as a Block participants using a shell script as an extension to the VM.
  3. These VMs are provisioned in a separate subnet i.e. signer and participant subnet.
  4. It is deployed in 3 regions based on the number of regions selected by User.

1. A key vault to store the blockchain ID, network token, and client tokens of all the chain core instances.
2. Managed disks for provisioning the OS disks of all the chain core instances.
3. Deploys VNET gateways for each VNET in each region.
4. These VNETs communicate with each other using VNET2VNET connections. It makes use of a shared passphrase at both the VNET gateways so it can communicate with each other.

## Workflow

The blockchain ID and network token for signers are stored in the Key vault from the generator chain core instance.

The Signer and participant chain core instances retrieve the blockchain ID and network token from the Key vault and configure the block signer and block participant VMs.

The Signer and participant chain core instances present in all three regions store their respective client tokens in the Key vault.

# Prerequisites

The following ARM template deployments require few mandatory prerequisites.

1. An Azure Active Directory (AAD) app registration (a.k.a. service principal) under the same AAD tenant as the subscription that the deployment will be done under. If an existing service principal is not present, create one by following steps below

## Creating an AAD app registration (service principal)

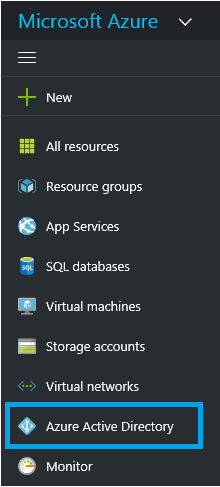
Sufficient permissions are required to register an application with Azure AD tenant and assign the application to a role in the Azure subscription. Make sure to secure proper permissions to perform these steps.

The following permissions are required:

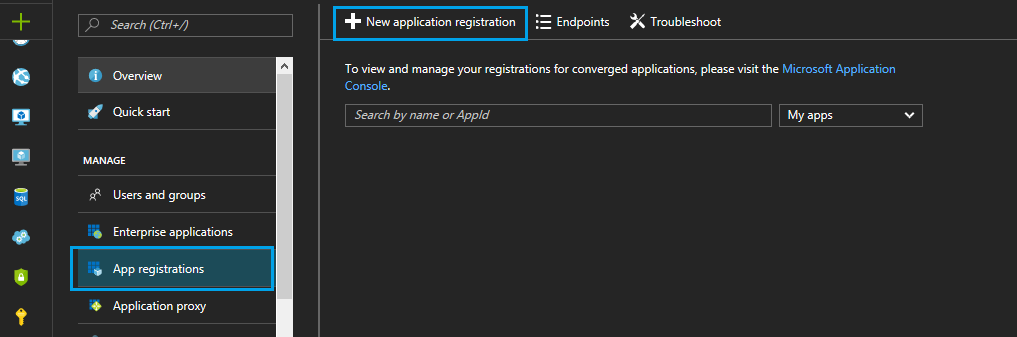
* An Azure user account that has admin access on the AAD tenant under which the deployment will be done.
* **Under User settings of the AAD section of the Azure portal, If App registrations** setting is set to **Yes,** any user in the Azure AD tenant can register an app.

## Create an Azure Active Directory (AAD) application

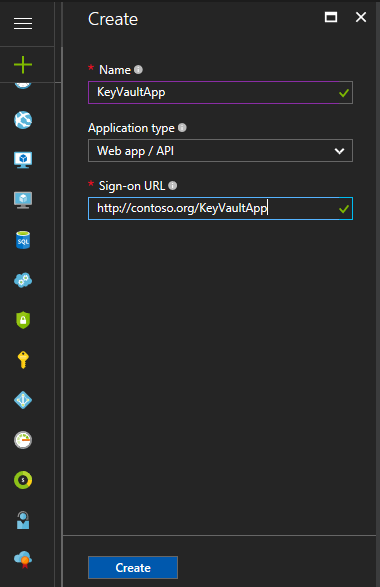
1. Log in to the [Azure portal](https://portal.azure.com/).
2. Select **Azure Active Directory**. Note if you do not see the option for Azure Active Directory in the left navigation pane, click on More Services at the bottom and search for Azure Active Directory.



1. Select **App registrations**



1. Click on + **New application registration**
2. Provide a Name (any random name) and Sign-on URL (any random URL, this can be changed later) for the application. Select either **Web app / API** or **Native** for the Application type. After setting the values, click on **Create**.

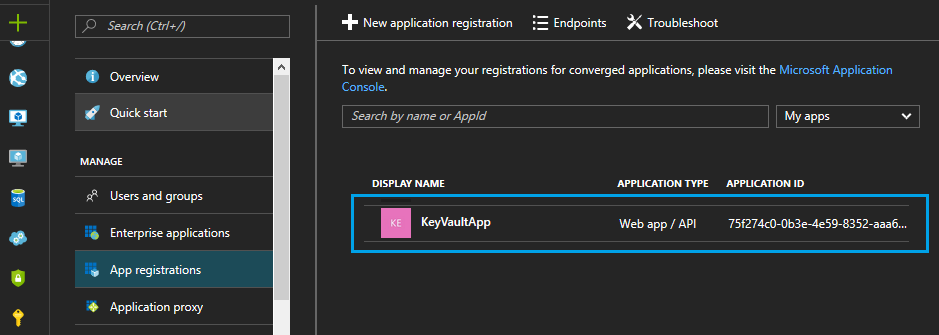


The application gets created.

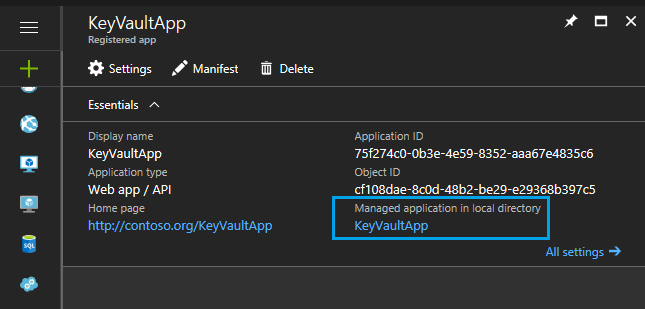
## Get Application ID and Authentication key

When programmatically logging in, use the ID for the application and an authentication key. To get those values, use the following steps:

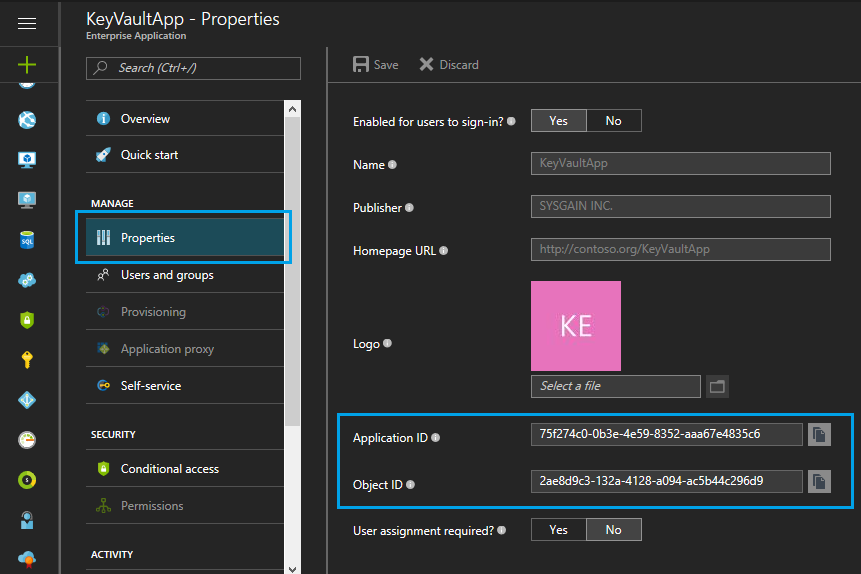
1. From **App registrations** in Azure Active Directory, select the new application (search for the application created above in the list of applications shown in the right side)



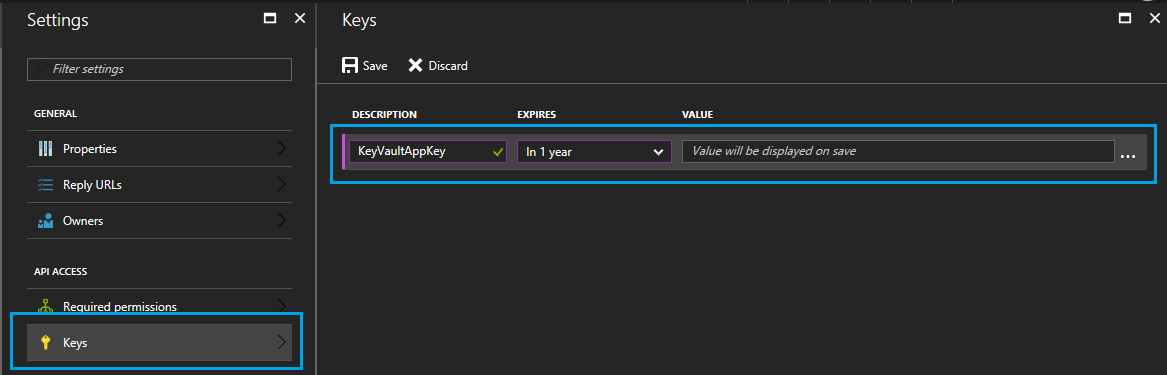
1. To get the Service principal’s Application ID[1] and Object ID[2], click on the application name (keyVaultApp in *blue color link*)



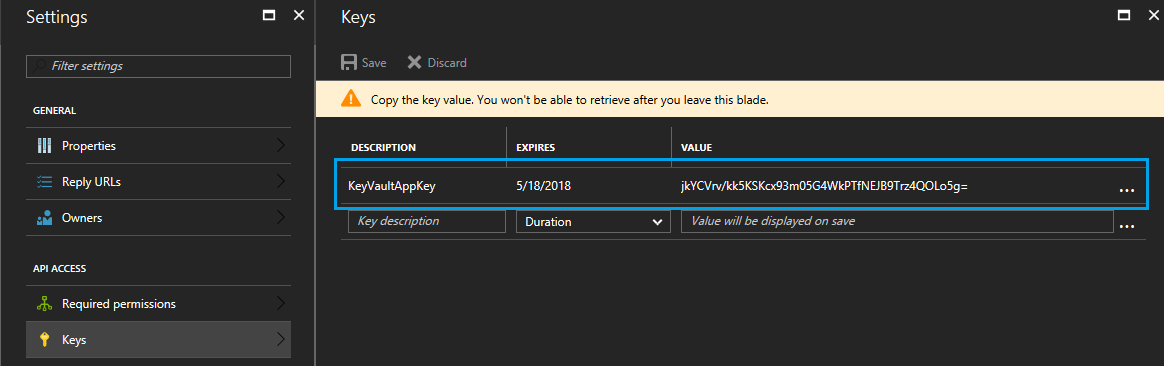
1. From the “Enterprise Application” view, navigate to the “Properties” section (highlighted in the blue box top left) as shown in the screenshot below. The Application ID and Object ID are highlighted in the blue box to the bottom right.
2. To generate an authentication key, go back to the **App registrations**, click on the app created above, click on settings then select **Keys.**



1. Provide a description of the key (any random string), and a duration for the key. When done, select **Save.**



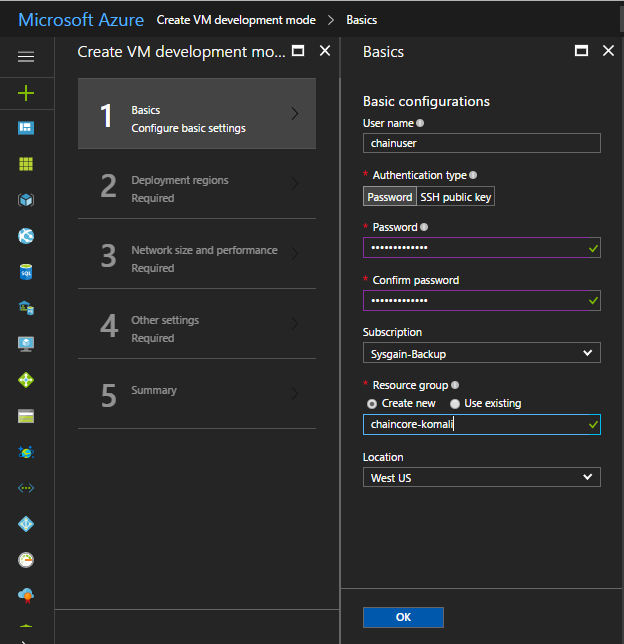
After saving the key, the value of the key is displayed. Copy this value – it cannot be retrieved later. Provide the key value with the application ID to log in as the application.

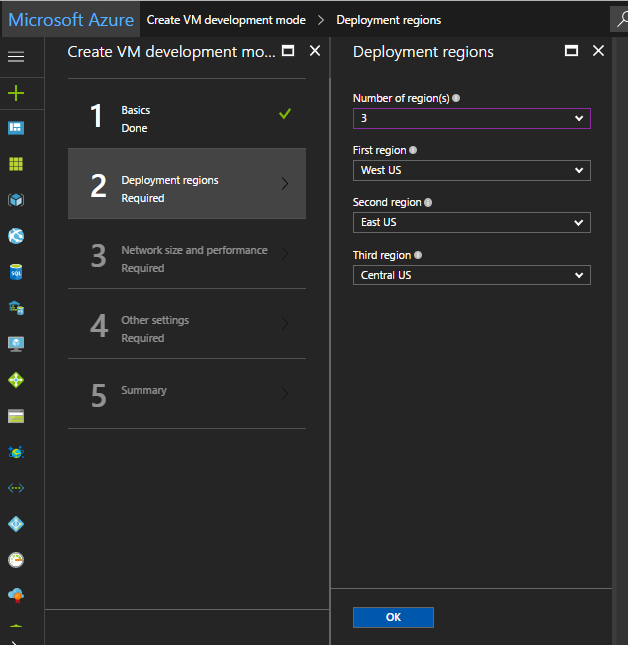


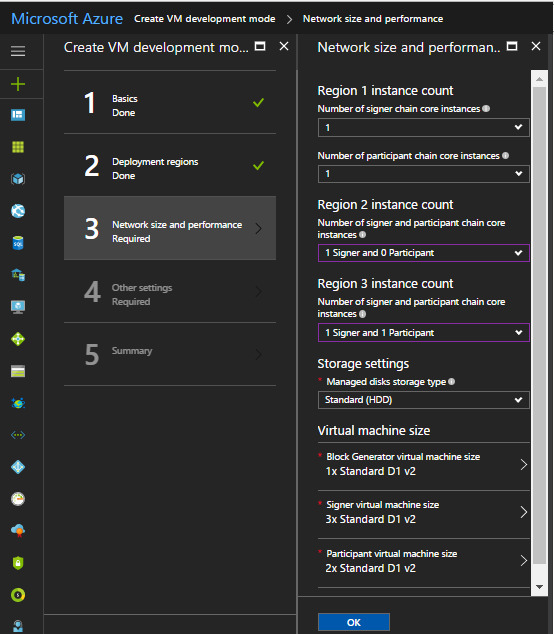
## Getting Started

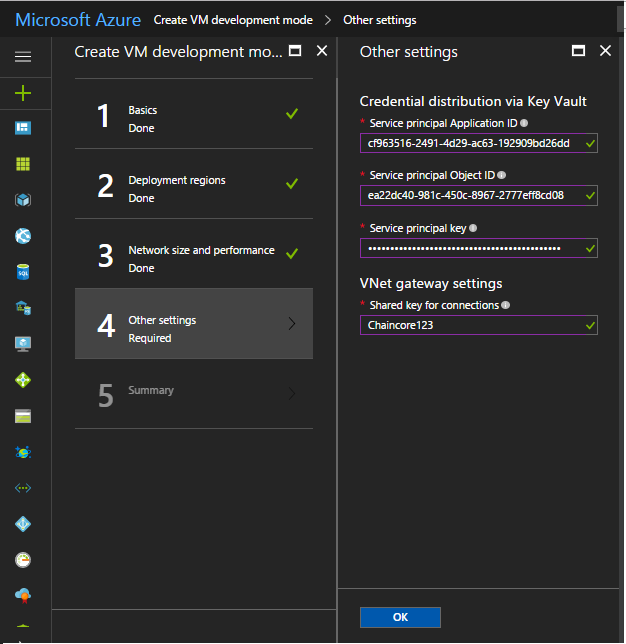
This process requires an Azure subscription that can support deploying several virtual machines and standard storage accounts. If necessary, [create a free Azure account](https://azure.microsoft.com/en-us/free/) to begin.

Once a subscription is secured, go to Azure portal. Select ‘+’, Marketplace (‘See all’), and search for ‘Chain Core Single Member Network Developer Edition’.

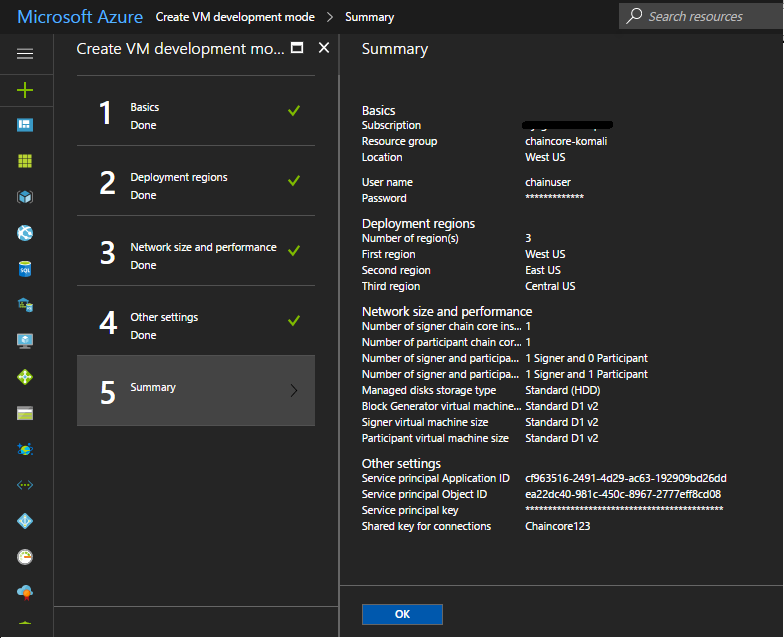


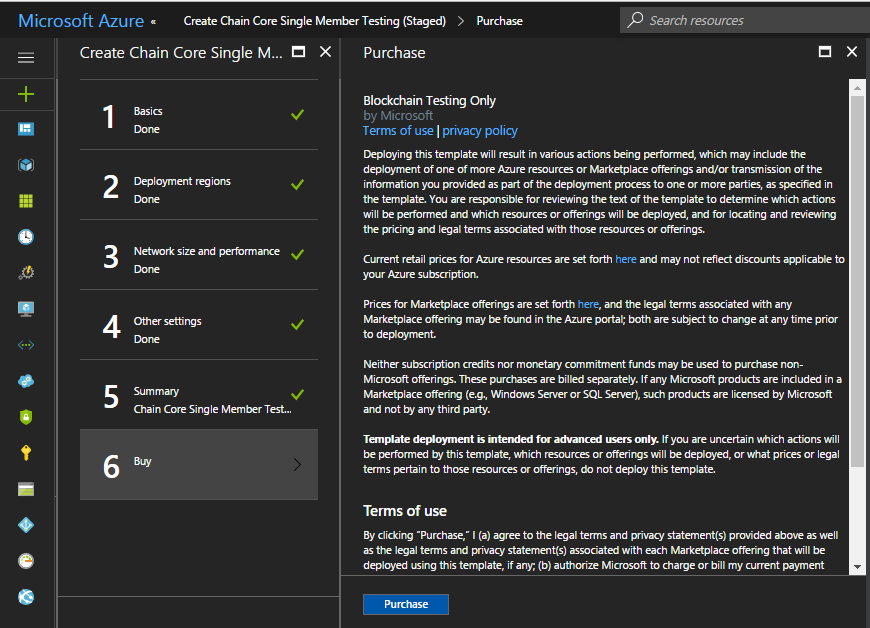






***Note****: To get the Application ID and Object ID refer the* ***Get Application ID and Authentication key*** *section from above.*

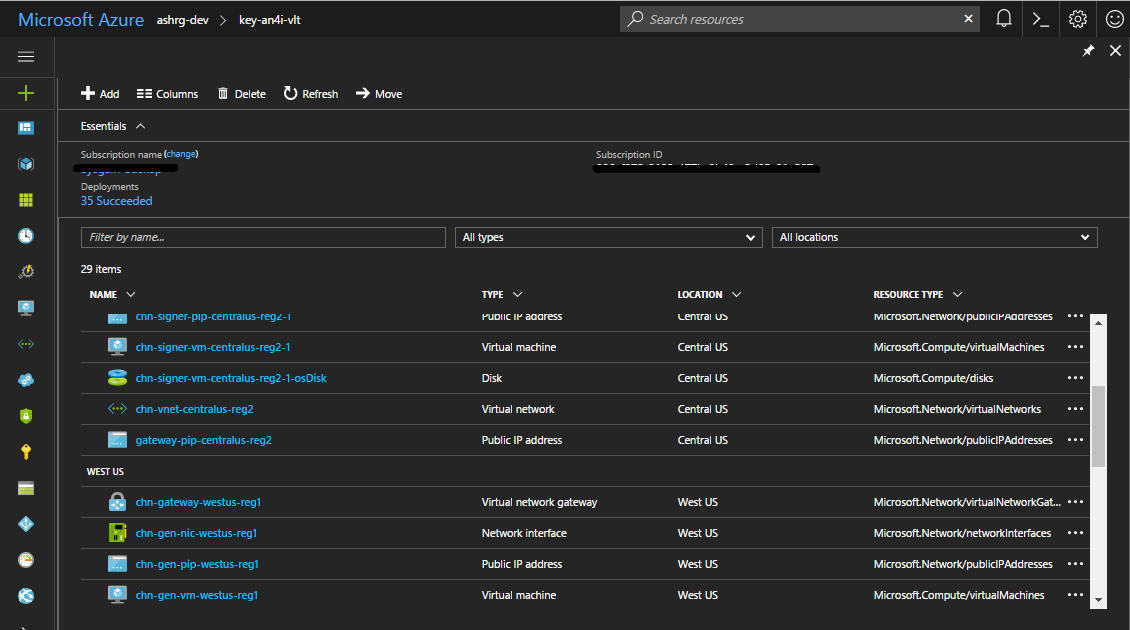




A detailed description of each parameter follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter Name** | **Description** | **Allowed Values** | **Default Value** |
| **Location (location)** | Resource group deployment location | Any location supported by Azure |  |
| **Number of region(s) (regionCount)** | Number of regions where the signers and participants get deployed | 1 to 3 | 1 |
| **First region (location\_1)** | Region where the Generator VM and corresponding signer and participant VM’s get deployed | Any location supported by Azure | West US |
| **Second region (location\_2)** | Region where signers and Participants are deployed other than the first region | Any location supported by Azure | Central US |
| **Third region (location\_3)** | Region where signers and Participants are deployed other than the first region and second region | Any location supported by Azure | East US |
| **Authentication Type (authenticationType)** | Authentication type to access the virtual machines | Password or SSH Public Key | Password |
| **User name (adminUserName)** | Admin username for all the deployed virtual machines | Any string | chainuser |
| **Password (adminPassword)** | Password to authenticate virtual machine | Password must be 12 characters and have 3 of the following 1 lower case character, 1 number, and 1 special character |  |
| **SSH Public key (sshPublicKey)** | SSH public key to authenticate virtual machine | SSH key type |  |
| **Block Generator virtual machine size (generatorVmSize)** | Size of the Generator virtual machine | "Standard\_A1",  "Standard\_A2",  "Standard\_A3",  "Standard\_A4",  "Standard\_A5",  "Standard\_A6",  "Standard\_A7",  "Standard\_A1\_v2",  "Standard\_A2\_v2",  "Standard\_A4\_v2",  "Standard\_A8\_v2",  "Standard\_A2m\_v2",  "Standard\_A4m\_v2",  "Standard\_A8m\_v2",  "Standard\_D1",  "Standard\_D2",  "Standard\_D3",  "Standard\_D4",  "Standard\_D1\_v2",  "Standard\_D2\_v2",  "Standard\_D3\_v2",  "Standard\_D4\_v2",  "Standard\_D5\_v2",  "Standard\_DS1",  "Standard\_DS2",  "Standard\_DS3",  "Standard\_DS4",  "Standard\_DS11",  "Standard\_DS12",  "Standard\_DS13",  "Standard\_DS14",  "Standard\_DS1\_v2",  "Standard\_DS2\_v2",  "Standard\_DS3\_v2",  "Standard\_DS4\_v2",  "Standard\_DS5\_v2",  "Standard\_DS11\_v2",  "Standard\_DS12\_v2",  "Standard\_DS13\_v2",  "Standard\_DS14\_v2",  "Standard\_DS15\_v2" | Standard\_DS1\_v2 (for Premium storage type) and Standard\_D1\_v2 (for Standard storage type) |
| **Signer virtual machine size (signerVmSize)** | Size of the Signer virtual machine | All the allowed values as Generator VM | Standard\_DS1\_v2 (for Premium storage type) and Standard\_D1\_v2 (for Standard storage type) |
| **Participant virtual machine size (participantVmSize)** | Size of Participant virtual machine | All the allowed values as Generator VM | Standard\_DS1\_v2 (for Premium storage type) and Standard\_D1\_v2 (for Standard storage type) |
| **Number of signer chain core instances- region 1 (signerNodeCountReg1)** | Number of signer instances in region 1 | 0 to 3 | 1 |
| **Number of participant chain core instances-region 1 (participantNodeCountReg1)** | Number of participant instances in region 1 | 0 to 3 | 1 |
| **Number of signer & participant chain core instances- region 2 (signerNodeCountReg2 & participantNodeCountReg2)** | Number of signer and participant instances in region 2 | “0 Signer and 1 Participant”, “1 Signer and 0 Participant”, and “1 Signer and 1 Participant” | “0 Signer and 1 Participant” |
| **Number of signer & participant chain core instances- region 3 (signerNodeCountReg3 & participantNodeCountReg3)** | Number of signer and participant instances in region 3 | “0 Signer and 1 Participant”, “1 Signer and 0 Participant”, and “1 Signer and 1 Participant” | “0 Signer and 1 Participant” |
| **Managed disks storage type (storageAccountType)** | Selection to use managed disk of type HDD or SSD | Standard or Premium | Standard |
| **Shared key for connections (VPNSharedKey)** | Shared key is used to establish encryption for connection | Must be 5 characters or more with a minimum of 1 lower case, 1 upper case and one number (Double quotes are not allowed) |  |
| **Service principal’s Object ID (objectId)** | This is the unique ID of service principal that will perform credential distribution across instances | GUID |  |
| **Service principal Application ID (servicePrincipal)** | The application ID of a service principal equivalent to a user name and is used to login as the service principal | GUID |  |
| **Service principal key (secretKey)** | The key of service principal is equivalent to its password |  |  |
| **Base URL (baseUrl)** | Deployment artifacts base URL |  |  |

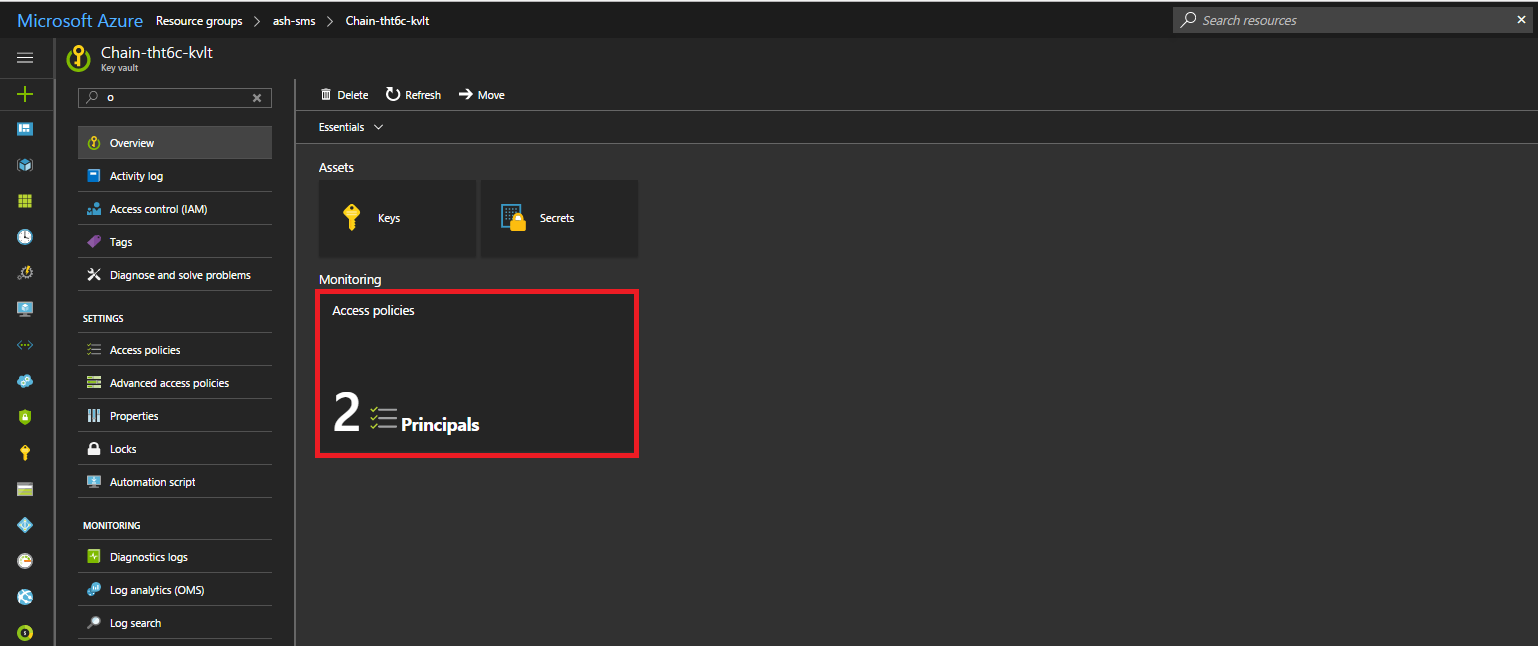
Once the template deploys successfully, it will show all the resources as shown below.



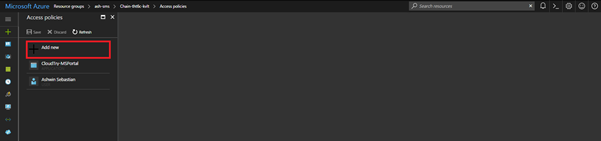
# Post Deployment

## Getting access to the deployed key vault

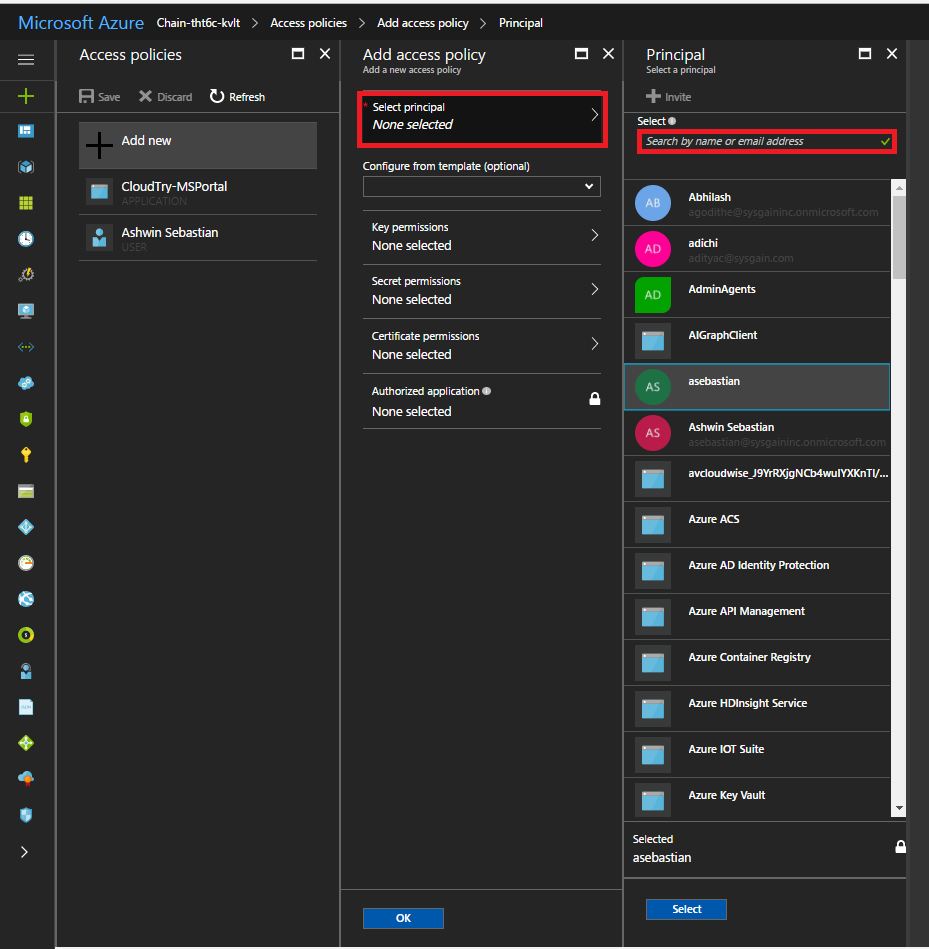
1. Open the ***key vault*** resource from the resource group.



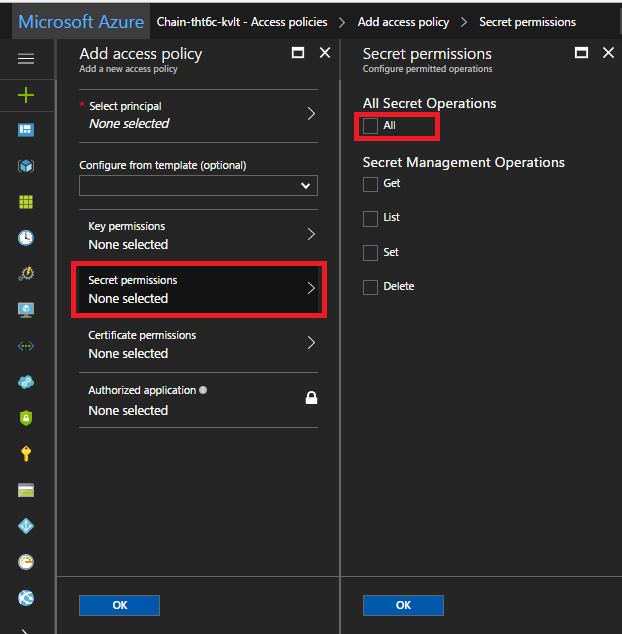
1. Click on the Access policies then click on ***Add new***, a new pane opens up called Add access policy

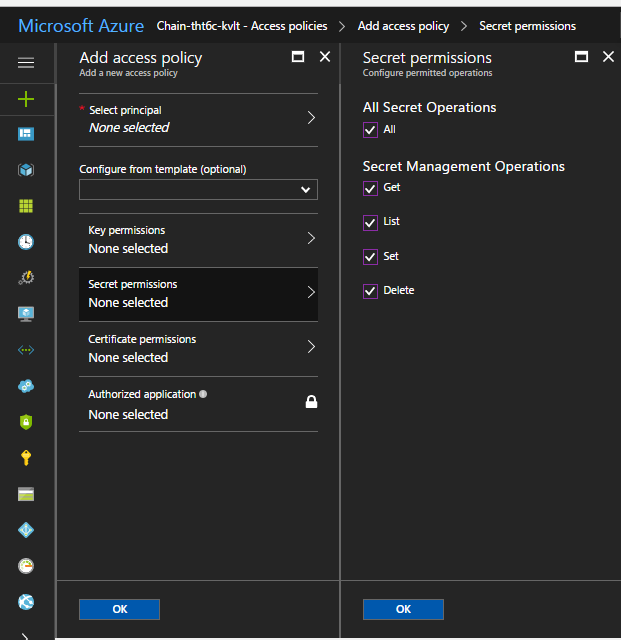


1. In ***Select principal section***, select the desired username/AppName and click select.

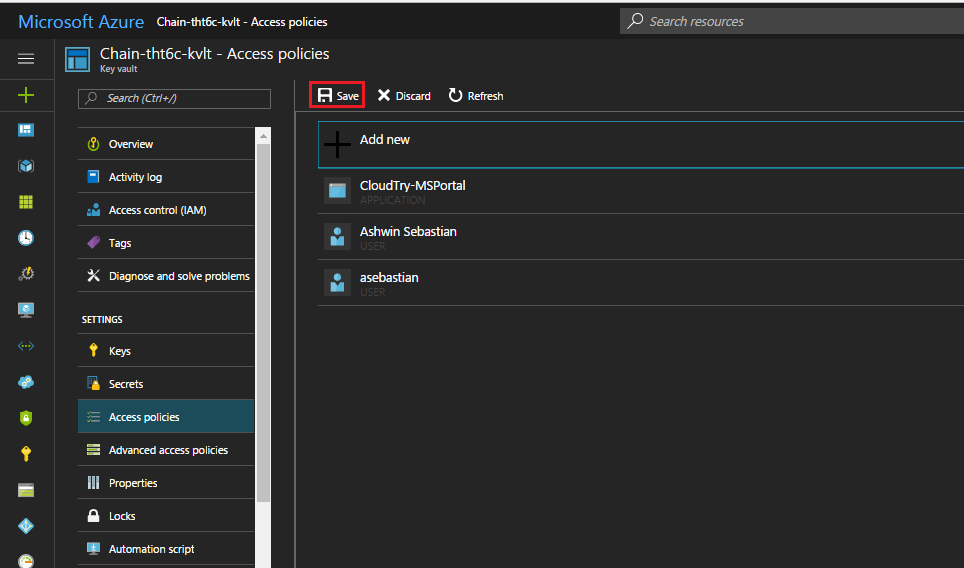


1. In ***Secret permissions section***, check the All checkbox to give all permissions to the concerned object ID and click **ok.**





1. The user name will be listed under Access polices. Now click on ***save***.



1. The Object ID has successfully been added.

## Accessing the values from the key vault

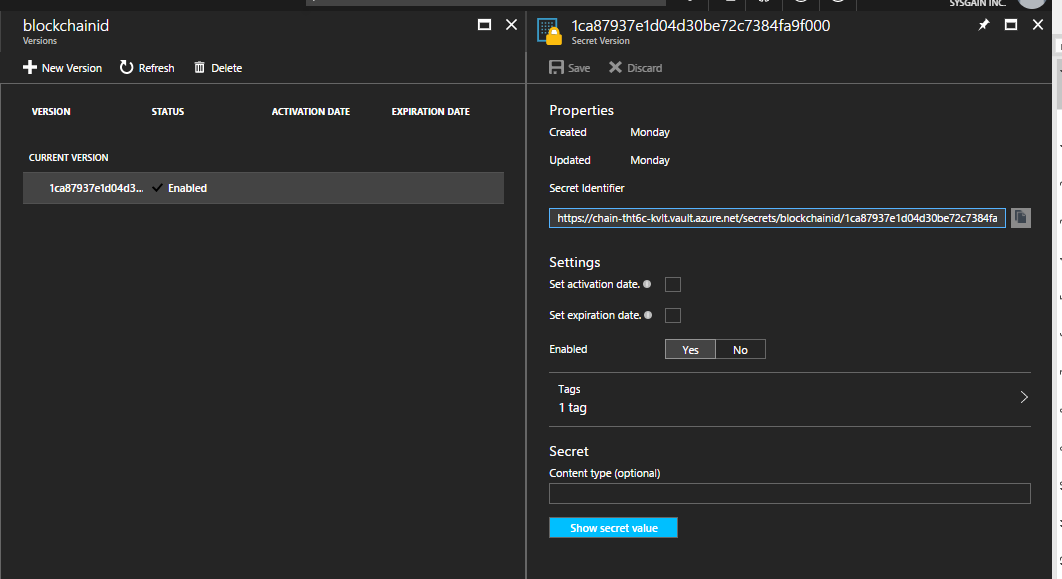
1. Open the ***key vault*** resource from the resource group.



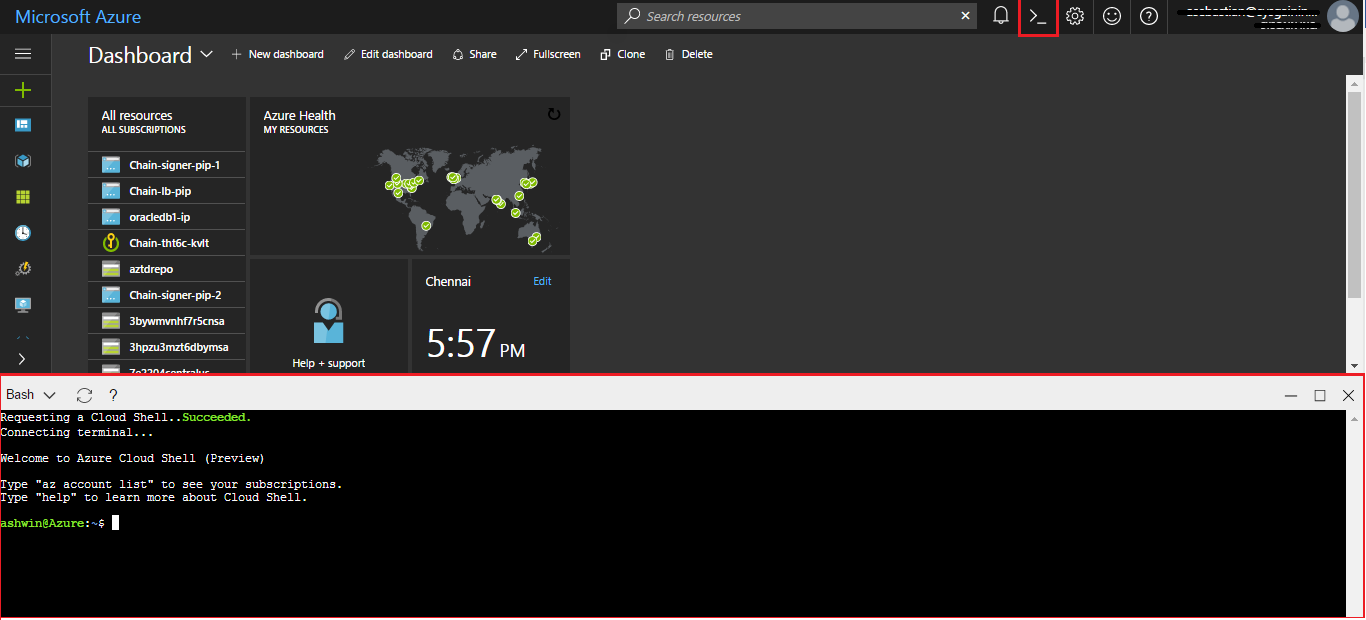
1. Click on ***Secrets*** to see the list of all the secret names.



1. Click on any secret and view its secret value by clicking on the current version of the secret name and click on ***Show secret value***.



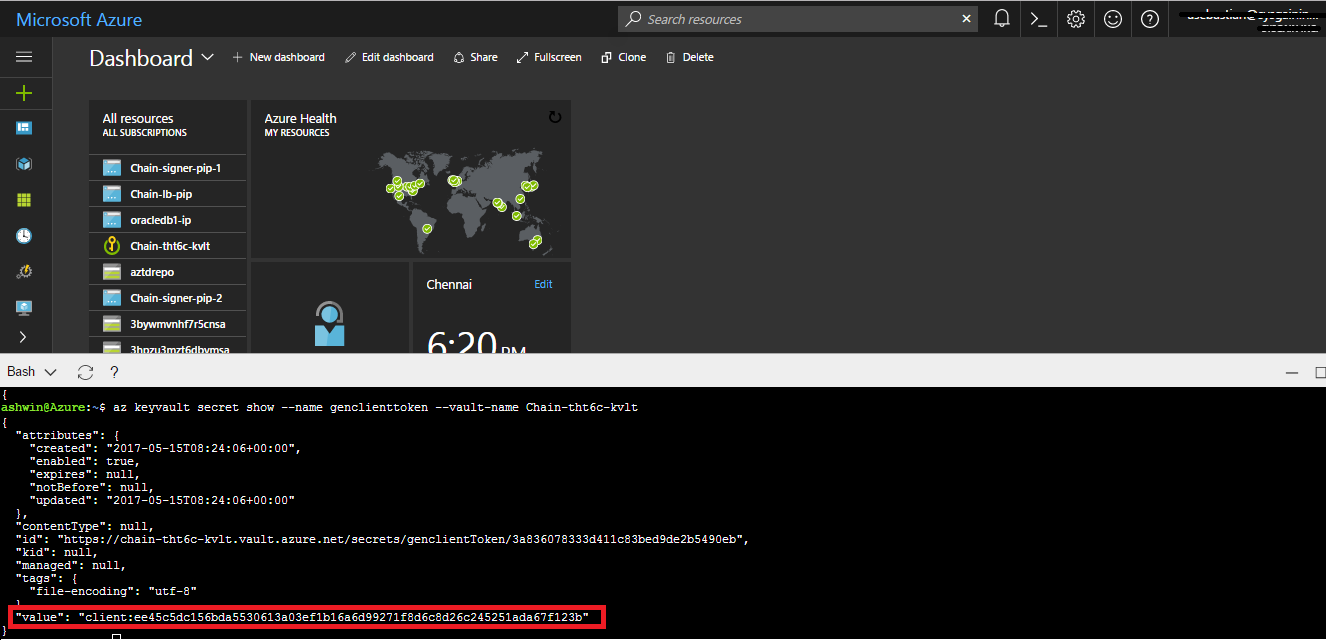
Secret values can also be viewed from the key vault using the Cloud Shell feature of the Azure portal.



Use the following commands to access the key vault and view the secret name.

***Note:*** These commands only work when the principal (i.e. object) ID is present in the access policies of the key vault.

az keyvault secret show --name <secret-name> --vault-name <keyvault-name>

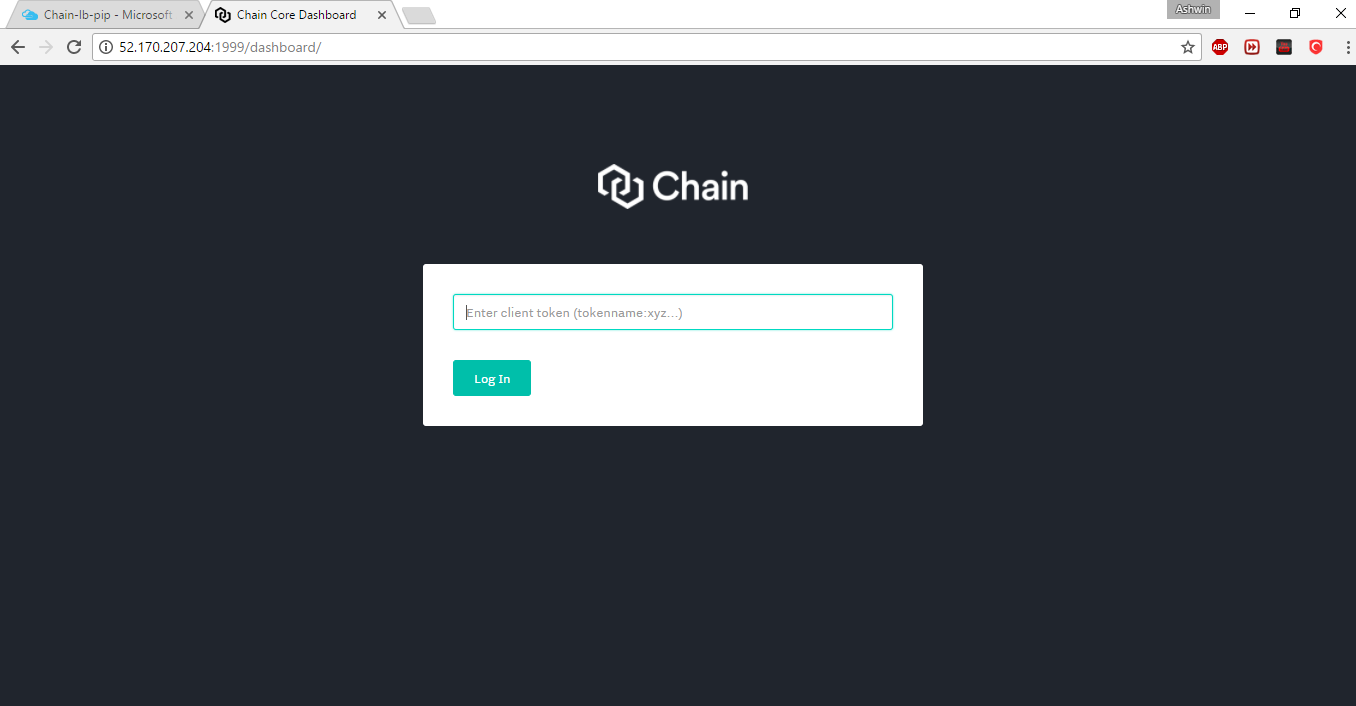


## Accessing the VM Dashboards

After the template deployment has completed, access the generator dashboard by accessing generator VM’s IP address at port 1999. You can get the public IP address by looking for Generator dashboard url listed in the outputs section of Microsoft.Template deployment. The output looks like

*http://generator-sxcv.westus.cloudapp.azure.com:1999*

http://<generator IP\_address>:1999

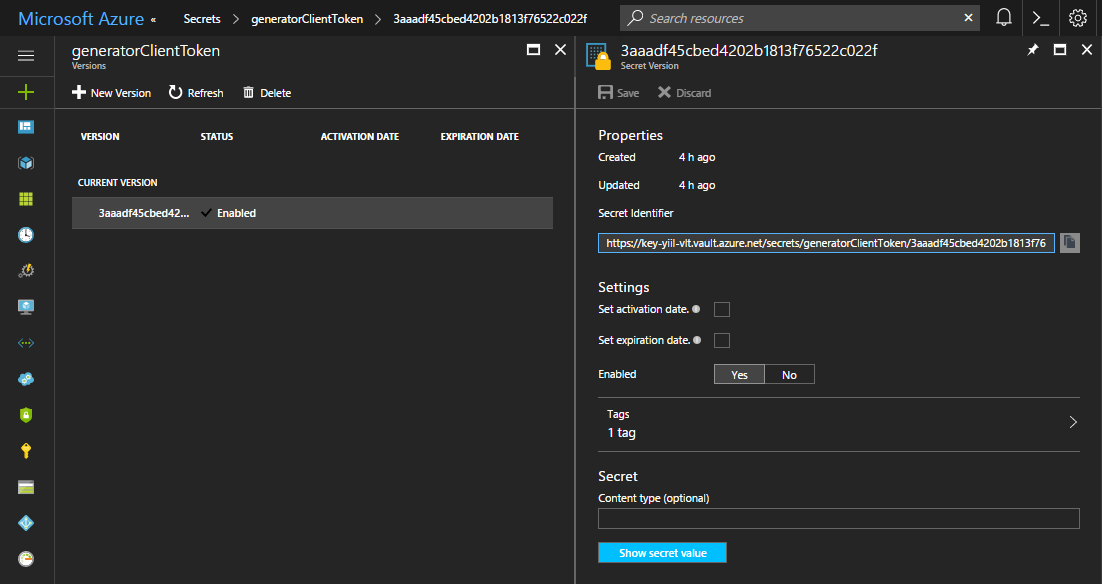


Get the client token from the key vault secrets

The secret name of the generator client access token is ***generatorClientToken.*** Click on the secret name and then click on ***show secret value***.

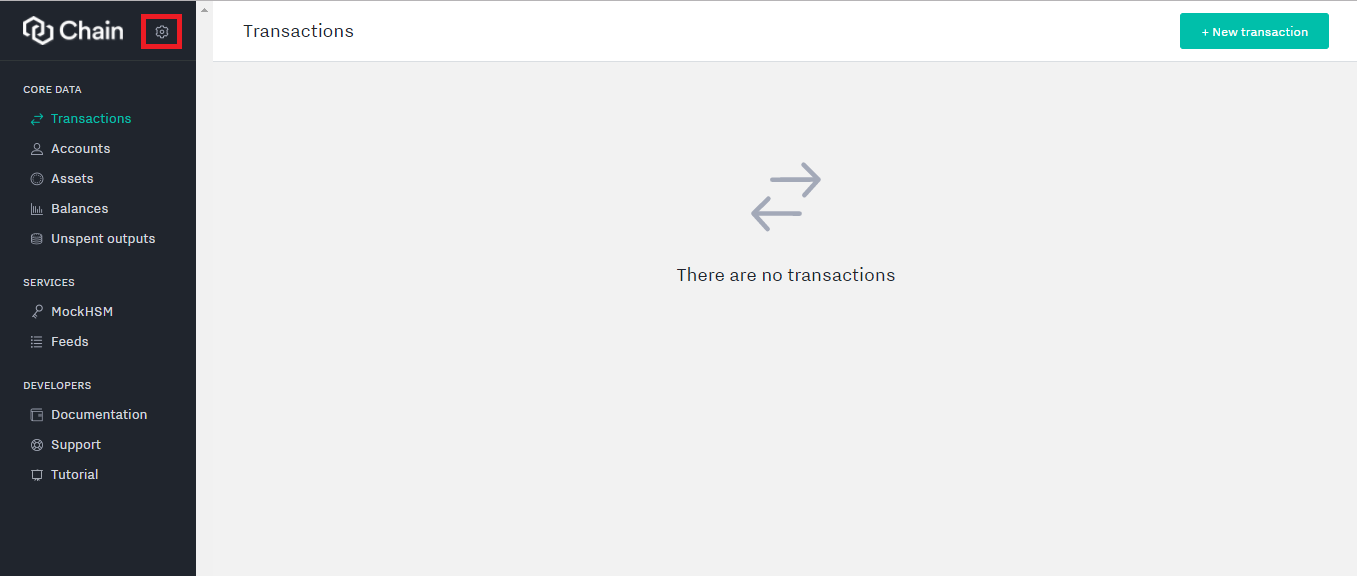




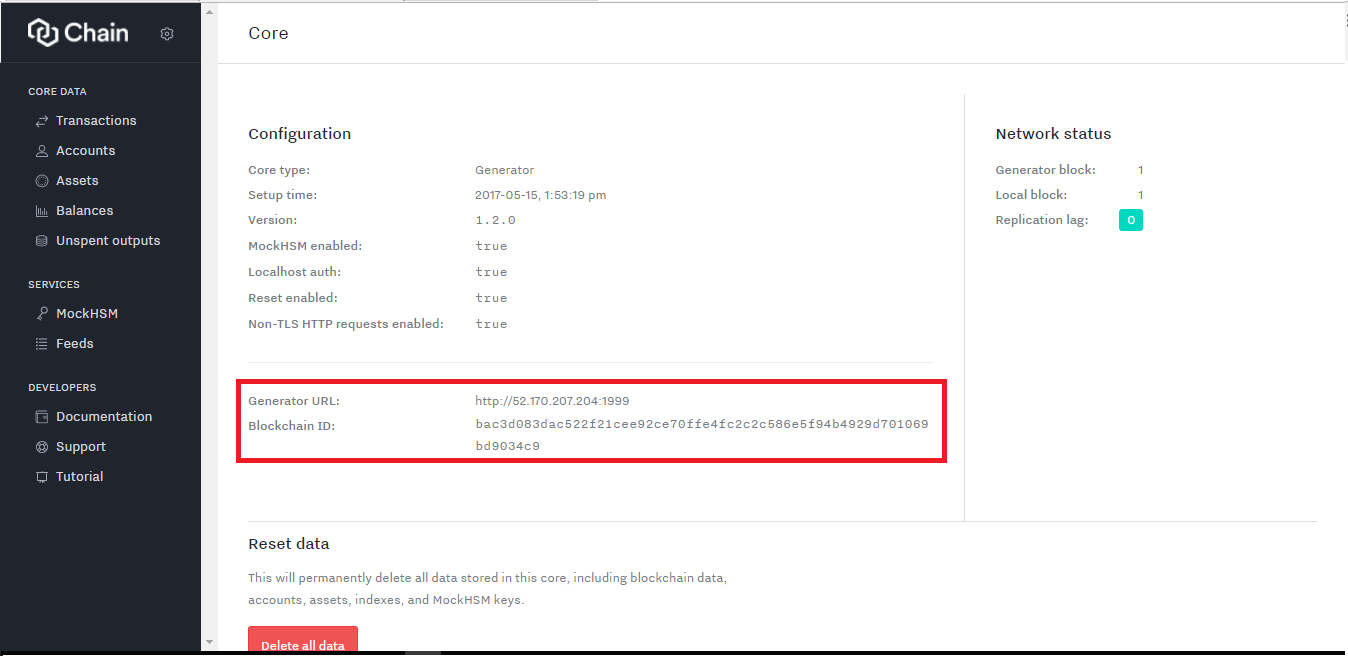


This shows the client access token needed to open the dashboard of generator.

Copy the token and paste it in the browser.



Click on the settings as shown in above screenshot, and select the core status to view the generator VM status



The blockchain ID generated in the generator VM is visible on this screen.

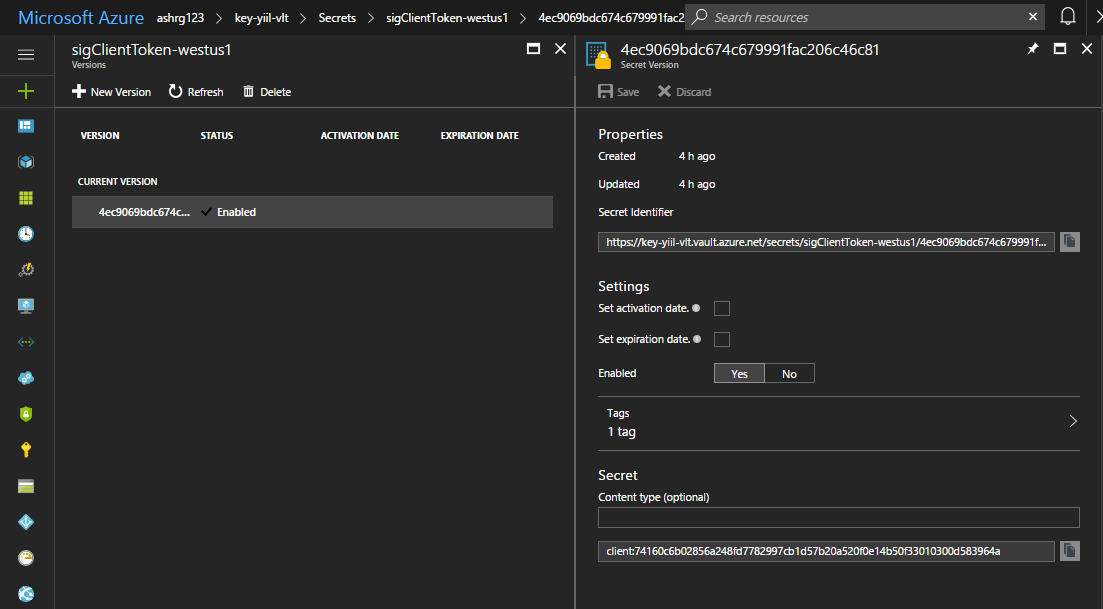
Now open any of the signer VMs to see if that VM is in sync with the generator VM. You can get the public IP address of any of the signer VM by looking for the public ip address resource in the resource group that includes the name segment of sig (e.g. chn-signer-pip1-eastus2-reg2).

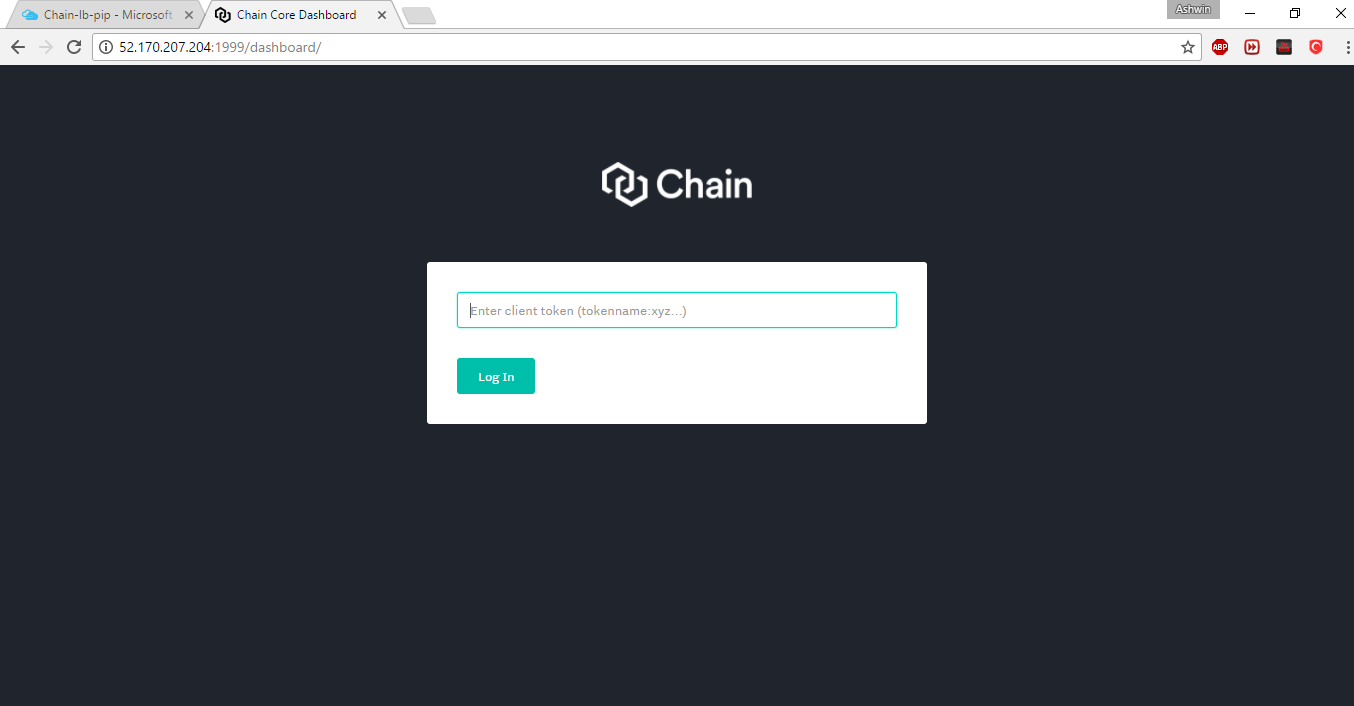
Open the dashboard of one of the signer VMs using its client token, which is stored in the key vault as part of this deployment.

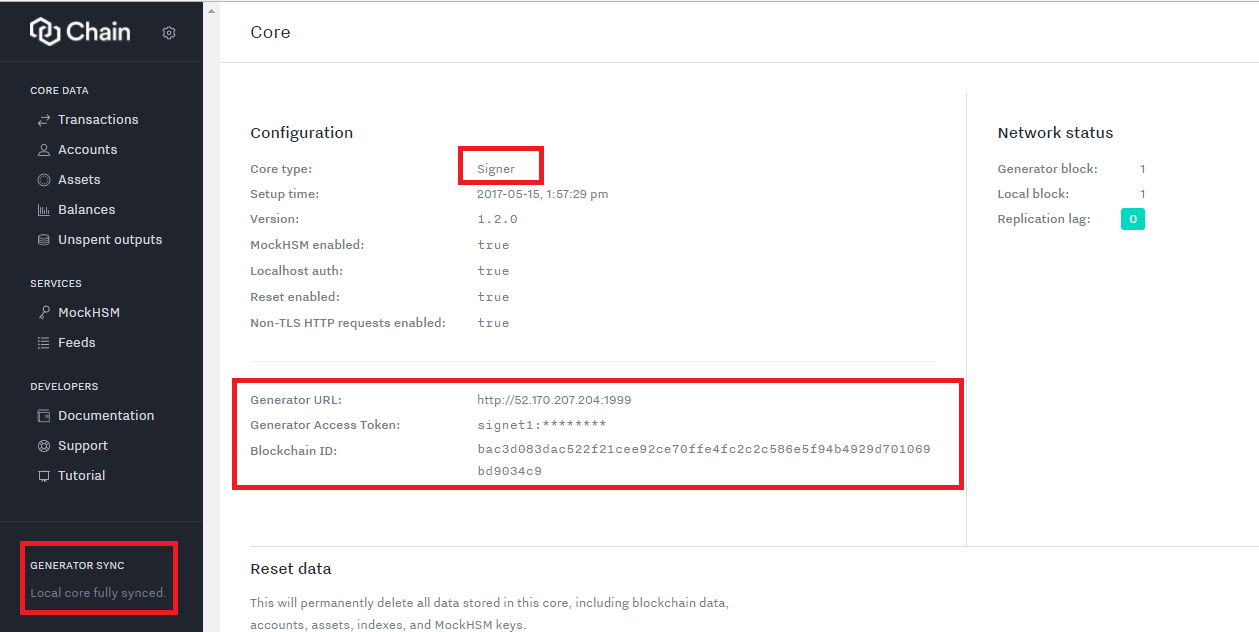




Open the signer dashboard using the signer client token and open the core status to see that it is synced with the generator VM.







The signer VM is in sync with the generator VM.

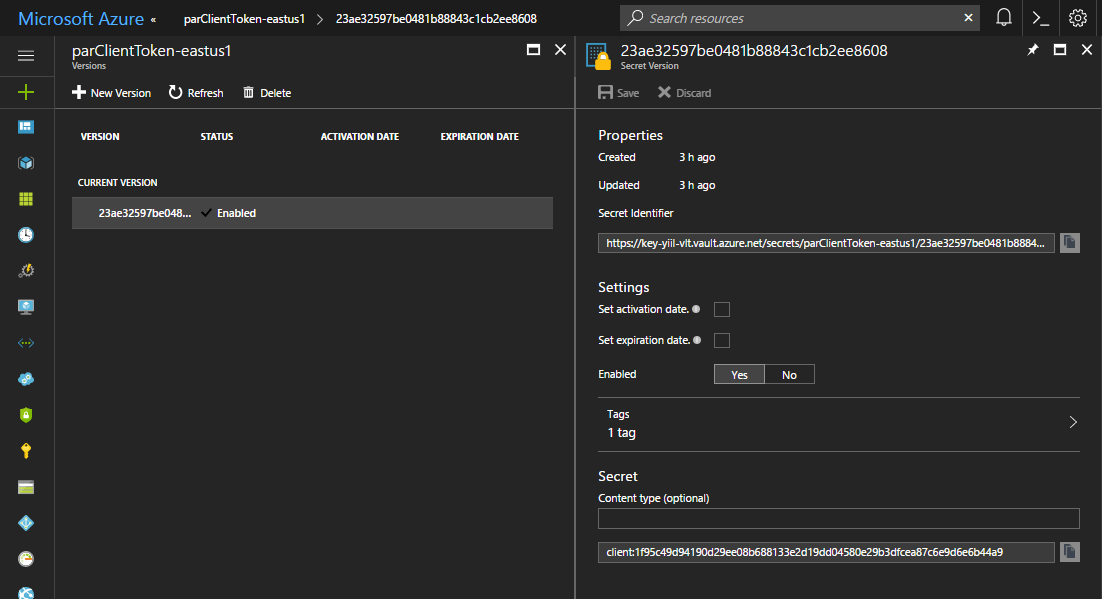
Now similarly open any of the participant VM to see if that VM is in sync with the generator VM. You can get the public IP address of any of the participant VM by looking for the public ip address resource in the resource group that includes the name segment of sig (e.g. chn-part-pip1-eastus2-reg2)

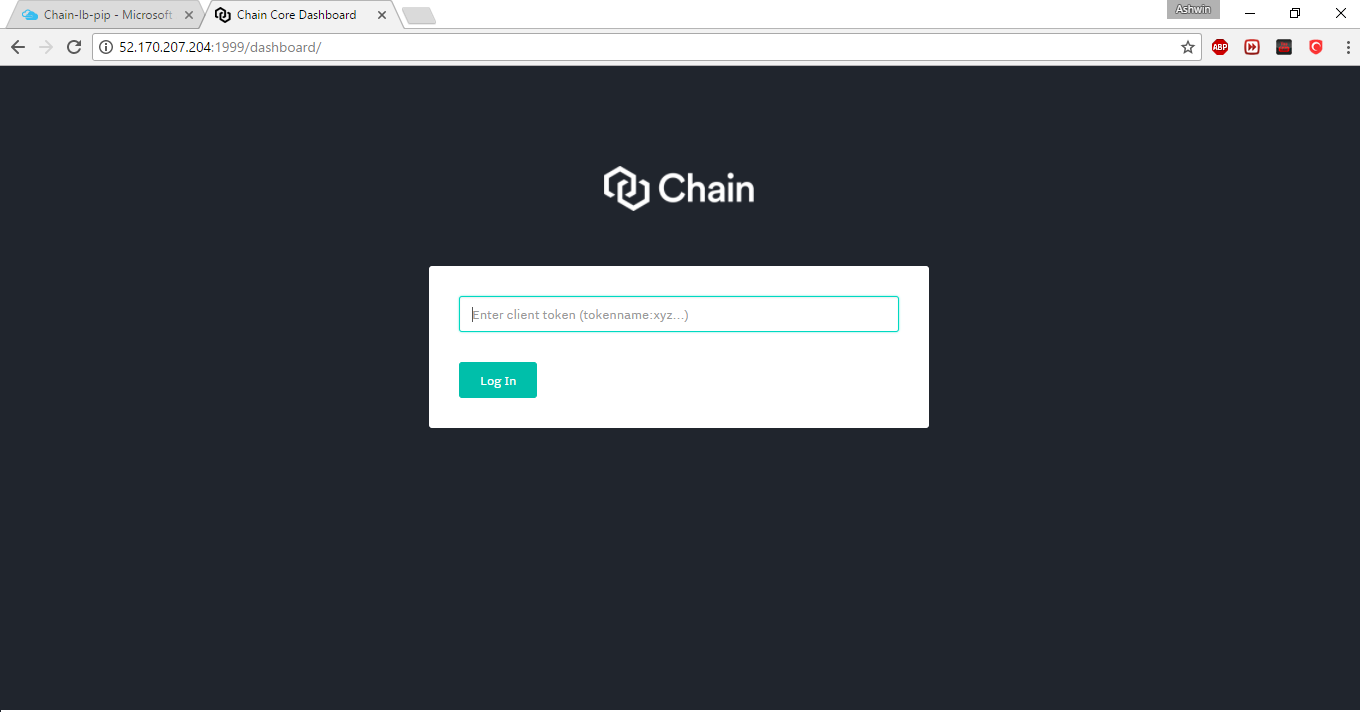
Open the dashboard of one of the participant VM using its client token which is stored in the key vault as part of this deployment.

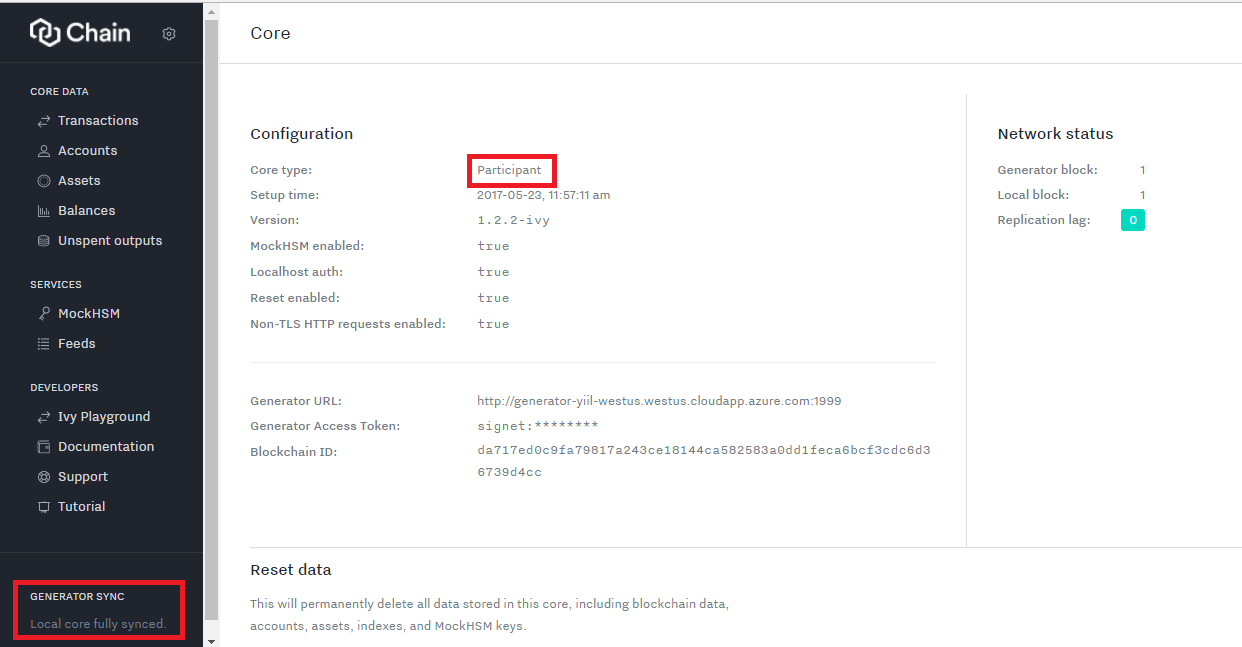




Open the participant dashboard using the participant client token and open the core status to see that it is synced with the generator VM.







The participant VM is in sync with the generator VM.