



Getting started with the Confidential Consortium Framework on Azure

A starter guide for developers

Version 1.0, July 2020

For the latest information about Azure, please see
<https://azure.microsoft.com/en-us/overview/>

For the latest information on Azure Confidential Computing (ACC), please see
<https://azure.microsoft.com/en-us/solutions/confidential-compute/>

For the latest information about open source on Azure, please see
<https://azure.microsoft.com/en-us/overview/choose-azure-opensource/>

For the latest information on the Open Enclave (OE) SDK, please see
<https://openenclave.io/sdk/>

For the latest information on the Confidential Consortium Framework (CCF), please see
<https://aka.ms/ccf/>

This page is intentionally left blank.

Table of contents

- NOTICE 2**
- ABOUT THIS GUIDE 3**
 - GUIDE ELEMENTS..... 4
 - GUIDE PREREQUISITES..... 5
 - Installing OpenSSH on Windows 10 5
 - Generating your RSA Key pairs with OpenSSH 6
- MODULE 1: INTRODUCING THE CONFIDENTIAL CONSORTIUM FRAMEWORK (CCF) 8**
 - A BRIEF RECAP ON BLOCKCHAIN NETWORK..... 8
 - UNDERSTANDING CCF OBJECTIVES 9
 - A CONSORTIUM FIRST APPROACH IN CCF 11
- MODULE 2: BUILDING A CCF NETWORK ON AN ACC VM..... 12**
 - CONNECTING TO YOUR ACC VM..... 14
 - INSTALLING CCF ON YOUR ACC VM 15
 - STARTING A TEST CCF NETWORK..... 16
 - SENDING REQUESTS TO THE CCF APPLICATION 19

Notice

This mini guide for developers is intended to new way for companies to build and execute blockchain-type or any other distributed applications using the Confidential Consortium Framework (CCF). The Confidential Consortium Framework (CCF) is available in open source at <https://microsoft.github.io/CCF/>.

MICROSOFT DISCLAIMS ALL WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, IN RELATION WITH THE INFORMATION CONTAINED IN THIS WHITE PAPER. The white paper is provided "AS IS" without warranty of any kind and is not to be construed as a commitment on the part of Microsoft.

Microsoft cannot guarantee the veracity of the information presented. The information in this guide, including but not limited to internet website and URL references, is subject to change at any time without notice. Furthermore, the opinions expressed in this guide represent the current vision of Microsoft France on the issues cited at the date of publication of this guide and are subject to change at any time without notice.

All intellectual and industrial property rights (copyrights, patents, trademarks, logos), including exploitation rights, rights of reproduction, and extraction on any medium, of all or part of the data and all of the elements appearing in this paper, as well as the rights of representation, rights of modification, adaptation, or translation, are reserved exclusively to Microsoft France. This includes, in particular, downloadable documents, graphics, iconographics, photographic, digital, or audiovisual representations, subject to the pre-existing rights of third parties authorizing the digital reproduction and/or integration in this paper, by Microsoft France, of their works of any kind.

The partial or complete reproduction of the aforementioned elements and in general the reproduction of all or part of the work on any electronic medium is formally prohibited without the prior written consent of Microsoft France.

Publication: July 2020

Version 1.0

© 2020 Microsoft France. All rights reserved

About this guide

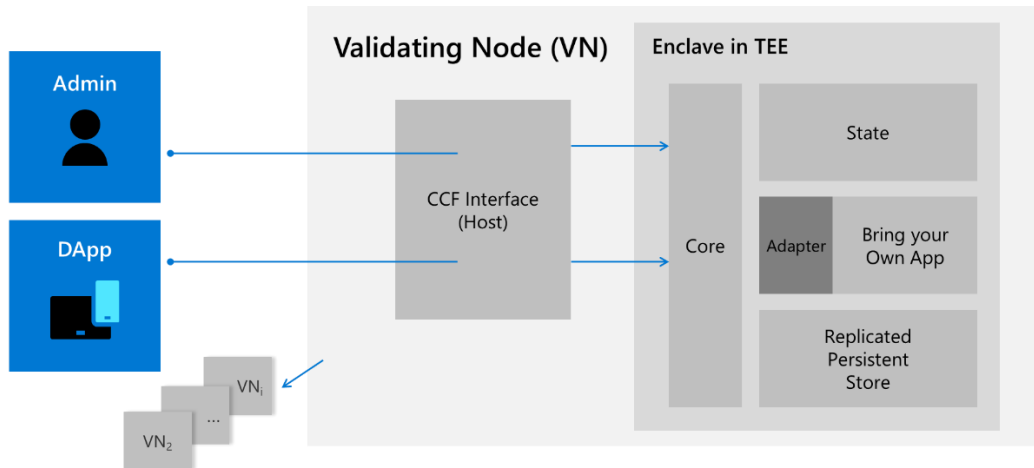
Welcome to the **Getting started with the Confidential Consortium Framework on Azure** starter guide for developers.

This document is part of a series of guides that covers confidential computing in the Cloud, and the Edge, and considerations that pertain to it from a development perspective and/or an infrastructure one. This series of guides is available at <https://aka.ms/CCDevGuides>.

The [Confidential Consortium Framework](#)¹ (CCF), a joint project between Microsoft Research and Azure Engineering, is an [open-source framework](#)² for building a new category of secure, highly available, and performant distributed applications that focus on multi-party compute and data.

Note CCF has been initially known as the Confidential Consortium Blockchain Framework, a.k.a. CoCo Framework.

While not limited just to blockchain applications, CCF can enable high-scale, confidential blockchain networks that meet key enterprise requirements, providing a means to accelerate production enterprise.



Note For an overview, watch the episodes [Confidential Consortium Framework \(CCF\) Overview](#)³ and [Confidential Consortium Framework \(CCF\) Part II: A deeper look and demos](#)⁴ on the Block Talk series on Channel 9.

Deployed as a blockchain-type network between parties, every node of a CCF network is running a [Trusted Execution Environment](#)⁵ (TEE) or enclave, - you can read the guide **Building and Executing Trusted Execution Environment (TEE) based applications on Azure** in this series of guides to learn about TEE -.

¹ Confidential Consortium Framework: <https://aka.ms/ccf>

² The Confidential Consortium Framework: <https://github.com/Microsoft/CCF>

³ Confidential Consortium Framework (CCF) Overview: <https://channel9.msdn.com/Shows/Blocktalk/Confidential-Consortium-Framework-CCF-Overview>

⁴ Confidential Consortium Framework (CCF) Part II: A deeper look and demos: <https://channel9.msdn.com/Shows/Blocktalk/Confidential-Consortium-Framework-CCF-Part-II-A-deeper-look-and-demos>

⁵ Trusted execution environment: https://en.wikipedia.org/wiki/Trusted_execution_environment

CCF is built on top of the [Microsoft Open Enclave SDK](#)⁶ (OESDK), (yet) another open source framework available on GitHub over two years. The OESDK aims at creating a single unified enclaving abstraction and consistent API surface for developers to build applications once that run across multiple TEE architectures, technologies and platforms⁷, such as the [Intel Software Extension Guard](#)⁸ (SGX).

CCF also provides a simple programming model of a highly available data store and a universally verifiable log that implements a ledger abstraction. Furthermore, CCF leverages trust in a consortium of governing members and in a network of replicated hardware-protected execution environments to achieve high throughput, low latency, strong integrity and strong confidentiality for application data and code executing on the ledger.

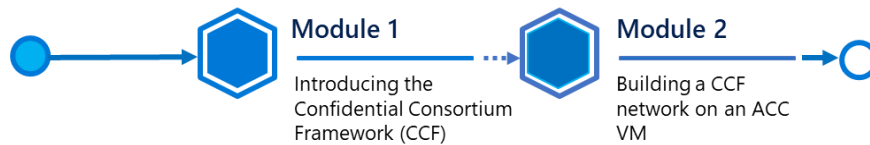
In this starter guide, and as its title suggest, we will cover the basics of the CCF.

This guide aims at helping you as a developer get started start with CCF, and in particular build a trusted network of nodes/TEEs, deploy an application on top of it, etc.

You will indeed learn how to create and deploy with CCF this new kind of multi-party applications on top of [Azure Confidential Computing](#)⁹ (ACC), and the SGX-based DC-series family of VMs it provides.

For that purposes, you're invited to follow a short series of two modules.

This guide is intended to be used in conjunction with the CCF documentation available at <https://microsoft.github.io/CCF/>. Please note, that besides Microsoft Azure here, networks can be run on-premises, in one or many cloud-hosted data centers, or in any hybrid configuration.



At the end of the mini guide, you will be able to:

- Instantiate an ACC VM well-suited for blockchain or any other trusted distributed multi-party applications development,
- Install the Confidential Consortium Framework (CCF) on top of that ACC VM,
- Deploy, manage, and use a multi-node CCF network.

Guide elements

In the starter guide modules, you will see the following elements:

- **Step-by-step directions.** Click-through instructions - along with relevant snapshots - or links to online documentation for completing each procedure or part.

⁶ Open Enclave SDK: <https://Open Enclave.io/sdk/>

⁷ The future of computing: intelligent cloud and intelligent edge: <https://azure.microsoft.com/en-us/overview/future-of-cloud>

⁸ Intel Software Extension Guard: <https://software.intel.com/en-us/sgx>

⁹ Azure Confidential Computing: <https://azure.microsoft.com/solutions/confidential-compute/>

- **Sample applications, and files.** A downloadable or cloneable version of the project containing the code that you will use in this guide, and other files you will need. **Please go to <https://github.com/Microsoft/CCF> on GitHub to download or clone all necessary assets.**

Guide prerequisites

To successfully leverage the provided code in this starter guide, you will need:

- A [Microsoft account](#)¹⁰.
- An Azure subscription. If you don't have an Azure subscription, create a [free account](#)¹¹ before you begin.
- A windows 10 local machine.
- A terminal console for your Windows 10 local machine, which allows you to remotely connect to a virtual machine (VM) in SSH, such as [PuTTY](#)¹², [Git for Windows](#)¹³ (2.10 or later).

Important note With Git, ensure that long paths are enabled: `git config --global core.longpaths true`.

As far as the latter is concerned, recent versions of Windows 10 provide OpenSSH client commands to create and manage SSH keys and make SSH connections from a command prompt.

Note For more information, see blogpost [What's new for the Command Line in Windows 10 version 1803](#)¹⁴.

Installing OpenSSH on Windows 10

The OpenSSH Client and OpenSSH Server are separately installable components in Windows 10 1809 and above.

Note For information about the OpenSSH availability on Windows 10, see [here](#)¹⁵.

To install OpenSSH on your Windows 10 local machine, perform the following steps.

1. Open an elevated PowerShell console.
2. Run the following command:

```
PS C:\> Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0
```

¹⁰ Microsoft Account: <https://account.microsoft.com/account?lang=en-us>

¹¹ Create your Azure free account today: https://azure.microsoft.com/en-us/free/?WT.mc_id=A261C142F

¹² PuTTY: <https://www.chiark.greenend.org.uk/~sgtatham/putty/>

¹³ Git for Windows: <https://git-for-windows.github.io/>

¹⁴ What's new for the Command Line in Windows 10 version 1803: <https://blogs.msdn.microsoft.com/commandline/2018/03/07/windows10v1803/>

¹⁵ Installation of OpenSSH For Windows Server 2019 and Windows 10: https://docs.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0

Path      :
Online    : True
RestartNeeded : False

PS C:\WINDOWS\system32>
```

Once the installation completes, you can use the OpenSSH client from PowerShell or the Windows 10 command shell.

Generating your RSA Key pairs with OpenSSH

OpenSSH includes different tools and more specifically the `ssh-keygen` command for generating secure RSA key pairs, that can be in turn used for key authentication with SSH.

RSA Key pairs refer to the public and private key files that are used by certain authentication protocols.

To generate your RSA Key pairs, perform the following steps.

1. Open an elevated PowerShell console.
2. Run the following command:

```
PS C:\> ssh-keygen
```

You can just hit ENTER to generate them, but you can also specify your own filename if you want. At this point, you'll be prompted to use a passphrase to encrypt your private key files. The passphrase works with the key file to provide 2-factor authentication. For this example, we are leaving the passphrase empty.

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\philber\.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\philber\.ssh/id_rsa.
Your public key has been saved in C:\Users\philber\.ssh/id_rsa.pub.
The key fingerprint is:
SHA256: zHdApZpI0+hzKynqCvEzJxWgdmSkiaTfqQqi9SLIuCO_europe\philber@PHILBER001
The key's randomart image is:
+---[RSA 3072]-----+
|.o=O.   .
|+.=.    .
|++=.   .o
|.o.*.  .+.
|. .o+S .
|. . . + .
|E.o .o
|X=.o. .o
|+++o.. .o.
+-----[SHA256]-----+
PS C:\WINDOWS\system32>
```


Note SSH public-key authentication uses asymmetric cryptographic algorithms to generate two key files – one "private" and the other "public". The **private key** file is the equivalent of a password and should be protected under all circumstances. If someone acquires your private key, they can log in as you to any SSH server you have access to. The **public key** is what is placed on the SSH server and may be shared without compromising the private key.

When using key authentication with an SSH server, the SSH server and client compare the public key for username provided against the private key. If the public key cannot be validated against the client-side private key, authentication fails.

By default, the files are saved in the following folder `%USERPROFILE%\.ssh`:

File	Description
<code>%USERPROFILE%\.ssh\id_rsa</code>	Contains the RSA private key
<code>%USERPROFILE%\.ssh\id_rsa.pub</code>	Contains the RSA public key.

Module 1: Introducing the Confidential Consortium Framework (CCF)

Before starting to create your Confidential Consortium Framework (CCF) network node(s), let's take the time to (shortly) remind or present you what blockchain (network) is and how TEEs are keeping the network safe from fraudulent transactions or external attacks in CCF.

This is the purpose of this first module to introduce some important concepts regarding blockchain network, and in contrast/parallel, outlines the benefits CCF provide. If you already familiar with these considerations, you can skip this module and jump into the next one start building your own network.

A brief recap on blockchain network

A blockchain (network) is a tamper-proof, highly available, decentralized ledger.

It's a system that maintains cryptographically chained blocks of transactions (containing cryptocurrencies, code or assets) across nodes that are linked in a peer-to-peer network:

- It records **what happened** and the **order** it happened in.
- There isn't any central server or organization, which rules the network. It's a fully decentralized environment.

There are 2 types of blockchain: **public** vs. **private**.

On a public blockchain, members are usually unknown - you only know public addresses - and everybody can join the network. This is a **permissionless network**.

Conversely, on a private blockchain, everybody knows each other and when someone tries to join the blockchain, he usually needs to be accepted inside the network by other peers. This is a **permissioned network**.

To keep the network tamper-proof, a blockchain is usually based on/govern by algorithms. The goal of those algorithms aims at validating that a transaction is valid and spreading this transaction across the peer network. Indeed, every peer of the network runs its own copy of the transaction ledger, so such an algorithm is in charge of keeping every ledger consistent across nodes.

On a public blockchain, the algorithm requires that a peer, which wants to validate a transaction, needs to bet a lot of "something" (computation power, cryptocurrency, etc.). So, considering that, at least half of the members inside the network are not trying to attack or steal something on the network, this part of peers will overwhelm cheaters, which will lost their "bets". This is why a public blockchain can be considered as secure: if you try to attack the blockchain, you likely will lose more than what you could win as potential rewards.

Those algorithms such as the well-known [Proof of work](https://en.wikipedia.org/wiki/Proof_of_work)¹⁶ (PoW) or [Proof of stake](https://en.wikipedia.org/wiki/Proof_of_stake)¹⁷ (PoS), provide reliability but there are some drawbacks, such as a low transaction throughput, a high computation cost, or an impossibility to scale up. To illustrate the point, Bitcoin currently uses approximately 69 TWh/year of electricity, about the same as

¹⁶ Proof of work: https://en.wikipedia.org/wiki/Proof_of_work

¹⁷ Proof of stake: https://en.wikipedia.org/wiki/Proof_of_stake

Austria, to process between 2 and 3 transactions (Tx)/second. Determining if a transaction was successful is probabilistic rather than deterministic and takes at least an hour. In addition, transactions are not confidential.

This is (currently) not "enterprise ready", resulting in the rise of private or consortium blockchains (in the meantime). In so far as you know every member inside the network, you can implement lighter algorithms (with less confidence) to verify transactions. Indeed, if someone tries to cheat, he/she will simply be kicked out of the blockchain. But this is not a perfect solution either, as this is still possible to cheat at least one time before being kicked, or, even if performances of the network are better, this isn't as good as a database.

Furthermore, governance is a difficult problem, making changes and bug fixes problematic.

Understanding CCF objectives

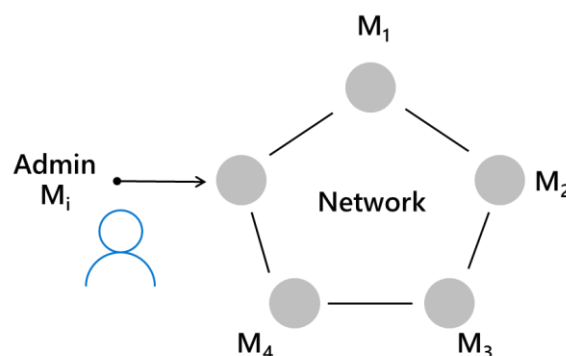
In this context, Microsoft tried another approach: what if the algorithm were just a way to establish a consensus between nodes, and the security part would be managed by something else, such as Trusted Execution Environments (TEEs) or enclaves.

As you saw before, an enclave is a secure area inside the processor, in a fully isolated environment. A user connected to the machine (even an administrator) can't see what is running and processing inside this enclave. However, the user can get an attestation about the code running in that enclave. This means that if you put some (trusted) code inside an enclave, you can verify at any moment if your code was modified or not.

And this is basically how CCF works: by putting the code in charge of verifying transactions inside enclaves, you can be sure that a transaction signed by the code inside the enclave and the code itself are correct and not fraudulent.

Also, since you can verify transaction by yourselves, you can make them private if needed, as other nodes, which will append their ledger with this transaction, will only have to know if the transaction is signed, they don't have to know about their content anymore.

With the use of Trusted Execution Environments (TEEs) or enclaves, CCF creates a trusted distributed blockchain network of physical nodes on which to run a distributed ledger, providing secure, reliable components for the protocol to use.



CCF is an infrastructure framework that leverages HW based (e.g. Intel SGX via ACC) Trusted Execution Environments (TEEs), [Paxos](https://en.wikipedia.org/wiki/Paxos_%28computer_science%29)¹⁸ and similar consensus mechanisms (namely [Raft](https://www.usenix.org/system/files/conference/atc14/atc14-paper-ongaro.pdf)¹⁹ for Crash Fault Tolerance and

¹⁸ Paxos (computer science): https://en.wikipedia.org/wiki/Paxos_%28computer_science%29

¹⁹ In Search of an Understandable Consensus Algorithm: <https://www.usenix.org/system/files/conference/atc14/atc14-paper-ongaro.pdf>

[PTBF](#)²⁰ for Byzantine Fault Tolerance), a governance model, standard cryptography to enable arbitrary blockchain protocols to deliver high-throughput, secure computation with confidential transaction models for consortium blockchain networks.

Note For additional information on governance, see paper [Blockchain Governance Models: Insights for Enterprises](#)²¹.

The Confidential Consortium Framework (CCF) doesn't compete with blockchain's technologies. It rather provides the foundation, the groundwork for turning these technologies into high-performance, confidential implementations.

All of these simplify consensus and transaction processing for high throughput and confidentiality.

Leveraging the power of trusted execution environments (TEEs), decentralized systems concepts, and cryptography, CCF provides for blockchain and multi-party application builders with enterprise-ready computation or blockchain networks that deliver:

- **Throughput and latency approaching database speeds.** Through its use of TEEs, CCF creates a network of remotely attestable enclaves - you can read the guide **Leveraging Attestations with Trusted Execution Environment (TEE) based applications on Azure** in this series of guides to learn about TEE -.

This gives a Web of Trust across the distributed system, allowing a user that verifies a single cryptographic quote from a CCF node to effectively verify the entire network. This simplifies consensus and thus improves transaction speed - currently running 50,000+ Tx/sec. For context, Visa averages 2,000 Tx/sec, with 50,000 Tx/sec peaks - and latency - Transaction latency in 10s of milliseconds -, all without compromising security or assuming trust.

- **Richer, more flexible confidentiality models.** Beyond safeguarding data access with encryption-in-use via TEEs, CCF uses industry standards (mTLS and remote attestation) to ensure secure node communication. Transactions can be processed in the clear or revealed only to authorized parties, without requiring complicated confidentiality schemes. Confidential transactions is more than welcome in a number of multi-party scenarios.
- **Network and service policy management through non-centralized governance.** CCF provides a network and service configuration to express and manage consortium and multi-party policies. Strong, smart contract-based governance actions, such as adding members to the governing consortium or initiating catastrophic recovery, can be managed and recorded through standard ledger transactions agreed upon via stakeholder voting.
- **Improved efficiency versus traditional blockchain networks.** CCF improves on bottlenecks and energy consumption by eliminating computationally intensive consensus algorithms for data integrity, such as proof-of-work or proof-of-stake.
- **Attestation at scale.** CCF Maintains integrity, resilience, and accountability properties of existing blockchains' technologies.

²⁰ Practical Byzantine Fault Tolerance: <http://pmg.csail.mit.edu/papers/osdi99.pdf>

²¹ Blockchain Governance Models: Insights for Enterprises: <https://blockchain.uark.edu/files/2019/11/BCCoEWhitePaper022019OPEN.pdf>

A consortium first approach in CCF

In a public blockchain network, anyone can transact on the network, actors on the network are pseudo-anonymous and untrusted, and anyone can add nodes to the network — with full access to the ledger and with the ability to participate in consensus. Similarly, other distributed data technologies (such as distributed databases) can have challenges in multi-party scenarios when it comes to deciding what party operates it and whether that party could choose or could be compelled to act maliciously.

In contrast, in a consortium or multi-party network backed by TEEs, as enabled by CCF, consortium member identities and node identities are known and controlled. A trusted network of TEEs, or enclaves, running on physical nodes is established without requiring the actors that control those nodes to trust one another, i.e. what code is run is controlled and correctness of its output can be guaranteed, simplifying the consensus methods and reducing duplicative validation of data.

In CCF, using TEE technology through all the goodness of the OESDK (in terms of abstraction level and unified API surface), the enclave of each node in the network (where cryptographically protected data is processed) can decide whether it can trust the enclaves of other nodes based on mutual attestation exchange and mutual authentication, regardless of whether the parties involved trust each other or not. This enables a network of verifiable, remotely attestable enclaves on which to run a distributed ledger and execute confidential and secure transactions in highly performant and highly available fashion.

If you want to learn more about CCF, we strongly advise to read the CCF technical report [CCF: A Framework for Building Confidential Verifiable Replicated Services](#)²².

Also feel free to explore the GitHub repository available [here](#)²³, or the website [at this address](#).²⁴

Now that you're equipped with an understanding of the core capabilities of CCF, you are ready to build our own network, based on that great framework.

This concludes this brief introduction of CCF and it's high time to get your hands a little bit dirty with CCF.

²² CCF: A Framework for Building Confidential Verifiable Replicated Services: <https://github.com/microsoft/CCF/blob/master/CCF-TECHNICAL-REPORT.pdf>

²³ Confidential Consortium Framework: <https://github.com/microsoft/CCF>

²⁴ Welcome to CCF's documentation: <https://microsoft.github.io/CCF/>

Module 2: Building a CCF network on an ACC VM

You will see in this second module how to create CCF nodes based on the DC-series VMs available in Azure Confidential Computing (ACC).

This section covers the setup of your ACC VM and installation of CCF.

For this sake of this guide, you will create a single node blockchain network. As CCF runs on top of the Open Enclave SDK (OESDK), a node will unsurprisingly be a [DCsv2-series](#)²⁵ VM which is an easy to instantiate enclave-capable VM in Azure.

Important Note V1 DC-Series VMs are in Preview and deploy an older version of the DC-Series VMs. They aren't going to be generally available and will remain in preview until deprecation.

For the most up-to-date technology and Confidential Computing VMs, you will need to use the new [Generation 2](#)²⁶ DCsv2-Series instead that correspond to an Azure Confidential Compute (Virtual Machine) V2 deployment. Besides the initially released v1 DC-Series VMs, DCsv2-Series VMs are backed by the latest generation of Intel XEON E-2288G Processor with the Intel SGX technology.



Azure Confidential Computing (Virtual Machine)
Microsoft Azure Compute
★★★★★ (0) Write a review
Overview Plans Reviews

GET IT NOW

Pricing information
Cost of deployed template components

Categories
Compute
Security

Support
Support
Help

Legal
License Agreement
Privacy Policy

Deploy the latest virtual machine from Azure with Intel SGX-enabled hardware.

Ensure that your business-critical data is secured while in use, by leveraging Azure's leading confidential infrastructure, tools, and SDK.

Take security to the next level and protect data while it's processed in the cloud by using secure enclaves. These enclaves are used to fully encrypt your data, and take Microsoft out of the Trusted Computing Base (TCB). This template will allow you to deploy the newest family of virtual machines that enable confidential computing features. With just a few configurations and a single-click deployment, you can build secure enclave-based applications to run inside of the virtual machine to protect your data and code, end-to-end. The DCsv2-Series Virtual Machines are backed by the latest generation of Intel Xeon processors with Intel SGX technology.

It is recommended that you deploy DCsv2 VMs for a greater selection of VM sizes, higher EPC (Enclave Page Cache), and a higher level of support.

Learn more

- Azure Confidential Computing
- Open-Enclave SDK
- DCsv2-Series
- Virtual Machine Pricing
- Available Regions
- Azure Confidential Computing Documentation

The DCsv2-series uses the latest generation of 3.7GHz Intel XEON E-2176G Processor with SGX technology, and with the Intel Turbo Boost Technology can go up to 4.7GHz. This family is currently available in Canada Central, East US, and UK South regions only. Future availability is on the way, see [Products available by region](#)²⁷.

²⁵ Preview: DC-series: <https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-previous-gen#preview-dc-series>

²⁶ Support for generation 2 VMs on Azure: <https://docs.microsoft.com/en-us/azure/virtual-machines/linux/generation-2>

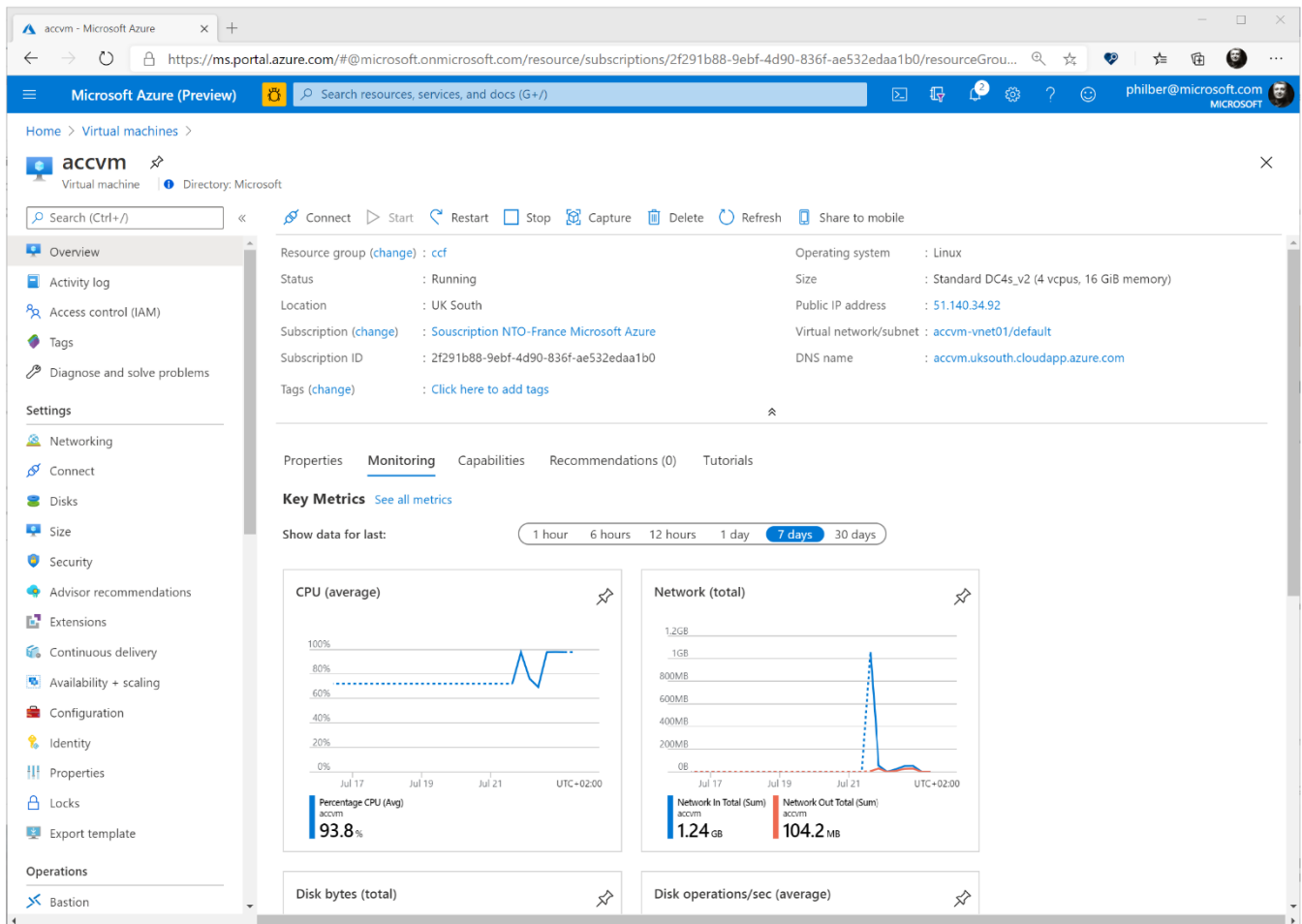
²⁷ Products available by region: <https://azure.microsoft.com/en-us/global-infrastructure/services/?products=virtual-machines®ions=all>

For the sake of this guide, between the four available size, you can opt for a [Standard DC4s v2²⁸](#) VMs with 4 vCPUs and 16 GiB of memory, or instead to minimize the implied cost a [Standard DC2s v2²⁹](#) with 2 vCPUs and 8 GiB of memory.

Furthermore, in terms of OS, amongst the three operating systems are supported for the above family of VMs, you will choose Ubuntu Server 18.04 TLS.

You can refer to the section *Module 1: Setting up a confidential computing VM in Azure* of the **guide Building and Executing Trusted Execution Environment (TEE) based application in Azure** in this series of guide to see how to instantiate such VMs.

At this stage, you should have an ACC VM up and running.



²⁸ DCsv2-series: <https://docs.microsoft.com/en-us/azure/virtual-machines/dcv2-series>

²⁹ Ibid

Connecting to your ACC VM

Once your ACC VM is online, by using a SSH client of your choice, such as OpenSSH, PuTTY, etc. you can test your remote connection to the newly created VM using the administrator credentials provided above. The public IP address of the VM can be found on the VM Networking page.

Note Depending on your configuration, you may need to configure a proxy in the SSH client to connect to the virtual machine.

To connect to your one of your VMs using OpenSSH on your Windows 10 local machine, perform the following steps:

1. From the Azure portal, search for your VM and click on it to display its menu.
2. Click on **Connect**, select SSH, and then make a note of the public IP address and the SSH connection string.

RDP **SSH** BASTION

Connect via SSH with client

1. Open the client of your choice, e.g. [PuTTY](#) or [other clients](#).
2. Ensure you have read-only access to the private key.

```
chmod 400 danywin.pem
```
3. Provide a path to your SSH private key file. ⓘ
Private key path
4. Run the example command below to connect to your VM.

```
ssh -i <private key path> danywin@accvm.uksouth.cloudapp.azure.com
```

Can't connect?

- [Test your connection](#)
- [Troubleshoot SSH connectivity issues](#)

In our illustration, the DNS FQDN name of the VM is `accvm.uksouth.cloudapp.azure.com`, and the SSH connection string is `danywin@accvm.uksouth.cloudapp.azure.com`.

The public IP address can be used in lieu of the above DNS name. For example, `51.140.34.92` in our illustration.

3. Open a PowerShell console, and the SSH to your VM:

```
PS C:\> ssh danywin@accvm.uksouth.cloudapp.azure.com
```

4. When prompted, type "yes". Optionally specify your passphrase if any for your private key.

Et voila! You should now be connected to your ACC VM.

Installing CCF on your ACC VM

In your Azure subscription, you should now have one ACC VM up and running and be connected to it.

Perform the following:

1. Open a remote terminal console on your first DC-series VM as per previous. We will use here [Windows Terminal](#)³⁰, i.e. a modern, fast, efficient, powerful, and productive terminal application for users of command-line tools and shells.
2. Now that you are logged in your ACC VM, first make sure that you're in your home directory on each of the VMs. Type the following command if not:

```
$ cd ~
```

3. It's now time to download CCF. To do so, on each of the VMs, type the following command:

```
$ git clone https://github.com/microsoft/CCF
```

4. Before building CCF, you will need to install a few dependencies. Fortunately, a Shell script is available inside the CCF project to do that for you.
 - a. Simply type the following command:

```
$ cd CCF/getting_started/setup_vm
```

- b. and then, execute the script:

```
$ ./run.sh driver.yml ccf-dev.yml
```

5. The next step consists in building CCF from its source files.
 - a. Go back into the root directory of CCF by typing:

```
$ cd ../../
```

- b. Create a *build* directory:

```
$ mkdir build
```

- c. Then, go inside it:

```
$ cd build
```

³⁰ Windows Terminal: <https://www.microsoft.com/en-us/p/windows-terminal/9n0dx20hk701>

d. You can then build run cmake.

```
$ cmake -GNinja ..
```

e. And finally build the solution:

```
$ ninja
```

Note To learn more about CCF installation (build arguments, etc.), see article [Getting Started](#)³¹ of the CCF documentation.

Congratulations! CCF is installed at this stage, you're now able to leverage it to [start a CCF network](#)³².

Starting a test CCF network

As per [CCF documentation](#)³³, "a CCF network consists of several nodes, each running on top of a Trusted Execution Environment (TEE), here and SGX enclave in your ACC VM. As such, a CCF network is both decentralized and highly available.

Nodes are run and maintained by operators, who are in charge of operating the considered CCF network (e.g. adding or removing nodes). Their identities are not registered in CCF.

However, nodes must be trusted by the consortium of members before participating in a CCF network. Members constitute the consortium governing a CCF network. Their public identity should be registered in CCF.

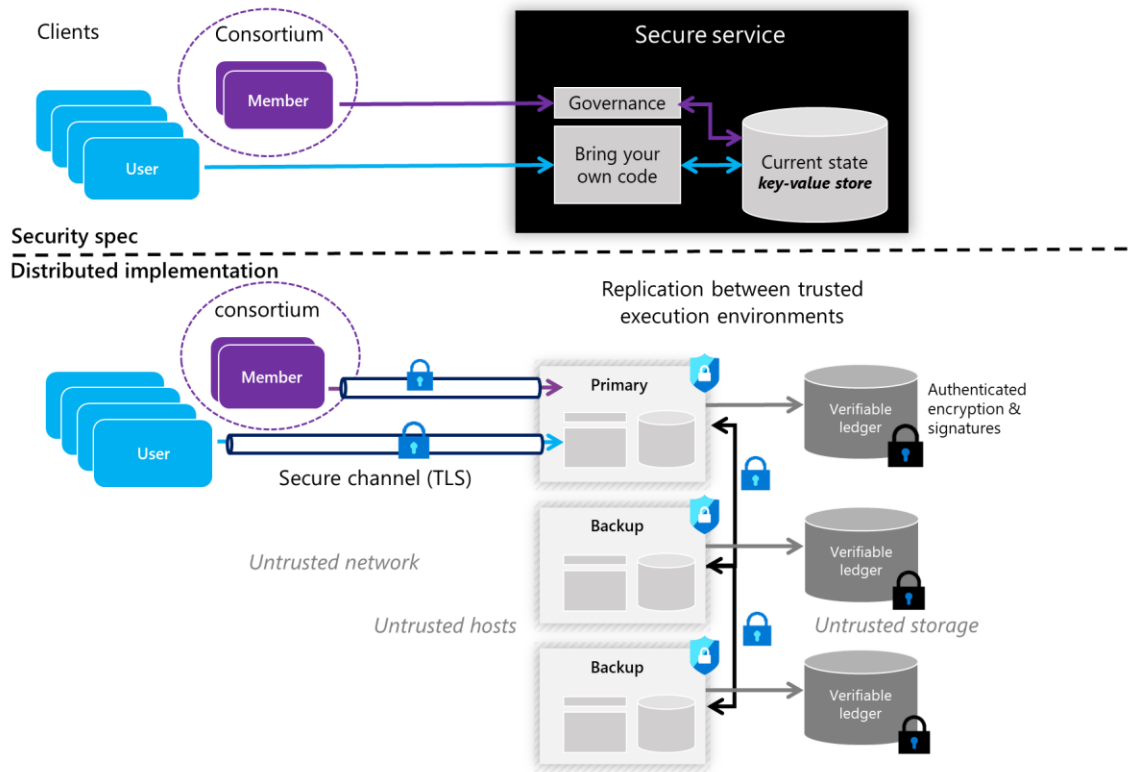
Each node runs the same CCF (a.k.a. transaction engine). A CCF application is consists of a collection of endpoints that can be triggered by users via commands over TLS. Their public identity should be voted in by members before they are allowed to issue requests.

Each endpoint mutates an in-TEE-memory Key-Value store that is replicated across all nodes in the network. Changes to the Key-Value store must be agreed by a variable number of nodes, depending on the consensus algorithm selected (either Raft or PBFT), before being applied."

³¹ GETTING STARTED: https://microsoft.github.io/CCF/getting_started.html

³² Starting a New Network: https://microsoft.github.io/CCF/master/operators/start_network.html#starting-a-new-network

³³ Concepts: <https://microsoft.github.io/CCF/master/concepts.html>



As a straightforward illustration, we will leverage the sample [logging application](#)³⁴, which provides an interface for users to write private messages available only to nodes in the network, and public messages available to anyone who has access to the ledger.

This application is implemented in C++ and [Lua](#)³⁵, i.e. a powerful, efficient, lightweight, embeddable scripting language. The implementation details in terms of RPC handlers and interface are discussed in the CCF documentation.

For convenience, we will run [tmux](#)³⁶ to split our console so that we can run the server on one tab, and execute commands on it in the other. Another solution would be to create another console and connect to it using SSH as before.

Here we will choose to use tmux. Therefore, start by running:

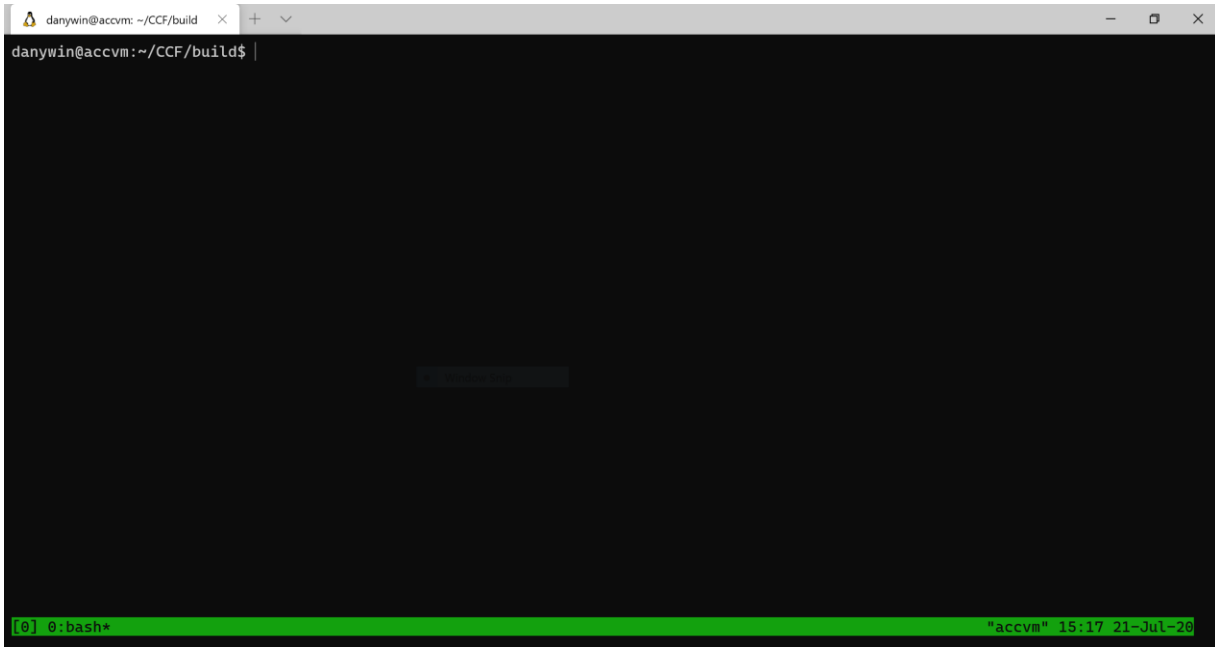
```
$ tmux
```

³⁴ Example Application: <https://microsoft.github.io/CCF/master/developers/example.html>

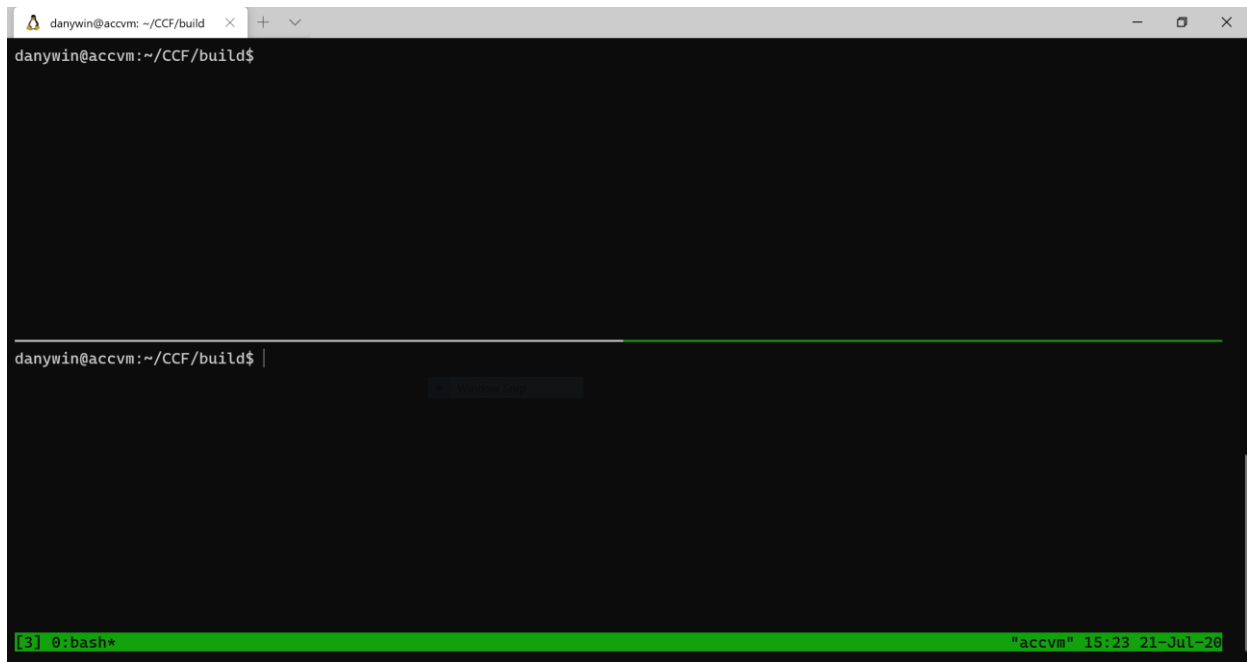
³⁵ The programming Language Lua: <https://www.lua.org/>

³⁶ A Quick and Easy Guide to tmux: <https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/>

This will give you the following screen:



Then by pressing first CTRL + b, then press ", this should split the console in two:



Now you can start a simple 3-node test network on your ACC VM by running in the bottom console the following command:

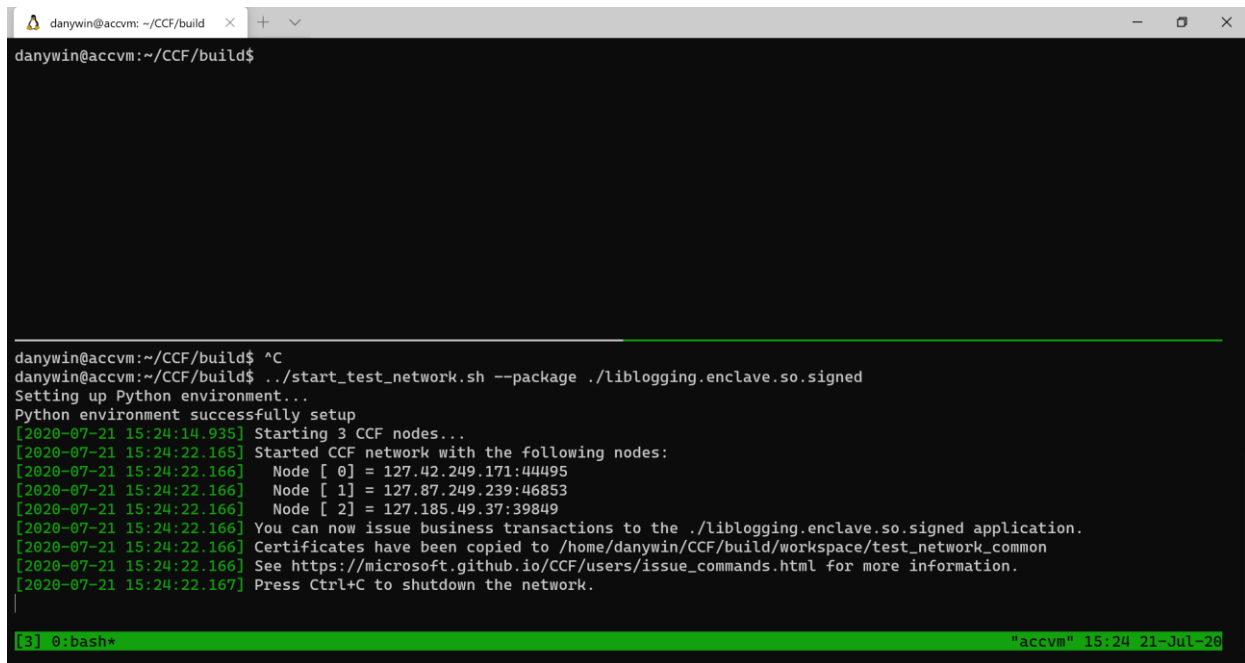
```
$ ../start_test_network.sh --package ./liblogging.enclave.so.signed
```

`start_test_network.sh` is a thin wrapper around the python script `start_network.py`. It ensures the necessary Python dependencies are available, sets some sensible default values, and uses curl to submit CCF commands.

As such, this script automates the steps described in [Starting a New Network](#)³⁷:

1. Generating new identities (private keys and certs) for the initial members and users.
2. Starting an initial node.
3. Starting two additional nodes, instructed to join the initial node.
4. Verifying that each node has successfully joined the new service, i.e. the referenced CCF application.
5. Proposing and passing governance votes using the generated member identities.

As a result, this should give you the following output:



```
danywin@accvm:~/CCF/build$  
  
danywin@accvm:~/CCF/build$ ^C  
danywin@accvm:~/CCF/build$ ./start_test_network.sh --package ./liblogging.enclave.so.signed  
Setting up Python environment...  
Python environment successfully setup  
[2020-07-21 15:24:14.935] Starting 3 CCF nodes...  
[2020-07-21 15:24:22.165] Started CCF network with the following nodes:  
[2020-07-21 15:24:22.166] Node [ 0 ] = 127.42.249.171:44495  
[2020-07-21 15:24:22.166] Node [ 1 ] = 127.87.249.239:46853  
[2020-07-21 15:24:22.166] Node [ 2 ] = 127.185.49.37:39849  
[2020-07-21 15:24:22.166] You can now issue business transactions to the ./liblogging.enclave.so.signed application.  
[2020-07-21 15:24:22.166] Certificates have been copied to /home/danywin/CCF/build/workspace/test_network_common  
[2020-07-21 15:24:22.166] See https://microsoft.github.io/CCF/users/issue_commands.html for more information.  
[2020-07-21 15:24:22.167] Press Ctrl+C to shutdown the network.  
  
[3] 0: bash* "accvm" 15:24 21-Jul-20
```

Great! Now the test CCF network with the example logging application has been started for us. We can see that it has created three nodes to which we can send requests.

Sending requests to the CCF application

Now that we have launched the server in the bottom panel, we can switch to the top panel by pressing CTRL+b, then TopArrow.

You will notice now that the top panel is responsive. Then you simply need to change folder:

```
$ cd workspace/test_network_common/
```

³⁷ Starting a New Network: https://microsoft.github.io/CCF/master/operators/start_network.html#starting-a-new-network

You should then get this:

```
danywin@accvm: ~/CCF/build
danywin@accvm:~/CCF/build$ cd workspace/test_network_common/
danywin@accvm:~/CCF/build/workspace/test_network_common$

danywin@accvm:~/CCF/build$ ^C
danywin@accvm:~/CCF/build$ ./start_test_network.sh --package ./liblogging.enclave.so.signed
Setting up Python environment...
Python environment successfully setup
[2020-07-21 15:24:14.935] Starting 3 CCF nodes...
[2020-07-21 15:24:22.166] Started CCF network with the following nodes:
[2020-07-21 15:24:22.166]   Node [ 0 ] = 127.42.249.171:44495
[2020-07-21 15:24:22.166]   Node [ 1 ] = 127.87.249.239:46853
[2020-07-21 15:24:22.166]   Node [ 2 ] = 127.185.49.37:39849
[2020-07-21 15:24:22.166] You can now issue business transactions to the ./liblogging.enclave.so.signed application.
[2020-07-21 15:24:22.166] Certificates have been copied to /home/danywin/CCF/build/workspace/test_network_common
[2020-07-21 15:24:22.166] See https://microsoft.github.io/CCF/users/issue_commands.html for more information.
[2020-07-21 15:24:22.167] Press Ctrl+C to shutdown the network.

[3] 0:bash+ "accvm" 15:55 21-Jul-20
```

We can now start to interact with our test CCF network.

You can communicate with the network using HTTPS requests. We will first start by logging a simple "Hello there" message at id 42.

To do so, first create a `request.json` using vim:

```
$ vim request.json
```

As a reminder, to edit a file using vim, you must first press `I` to enter insertion mode. Then enter this:

```
{
  "id": 42,
  "msg": "Hello There"
}
```

Then press `ESC` and type `:.wq` to exit and save. To make sure the file was properly edited you can cat it:

```
$ cat request.json
>
{
    "id": 42,
    "msg": "Hello There"
}
```

Now we can send this request to our CCF logging application.

To interact with the test CCF network, you need to provide your certification and private key, as well as the certification of the network which was generated by the first node.

This is used as the Certificate Authority (CA) to authenticate the server's identity.

The certificates and private keys were generated using the *keygenerator.sh* present in the folder *build/workspace/test_network_common*, and you can also notice that it was already called by the *start_test_network.sh* script we used earlier, as the current folder is already populated with certificates and private keys.

One more thing to notice, is that there are members and users created. Members can participate in the governance, which means that they can access the */gov* endpoint, while users can only interact with the */app* endpoint.

Now, we can send our first message to the logging app using a POST request. To do so, simply run:

```
$ curl https://<ccf-node-address>/app/log/private --cacert networkcert.pem --key user0_privk.pem --cert user0_cert.pem --data-binary @request.json -H "content-type: application/json" -i
```

with *<ccf-node-address>* the address of one node. This address is available in your screen below where you started the test CCF network.

You should get the following output:

```
>
HTTP/1.1 200 OK
content-length: 5
content-type: application/json
x-ccf-tx-seqno: 23
x-ccf-tx-view: 2
true
```

We can then have a look at the message using a GET request:

```
curl https://<ccf-node-address>/app/log/private?id=42 -X GET --cacert networkcert.pem --cert user0_cert.pem --key user0_privk.pem
```

Which should give the output:

```
>
{"msg": "Hello There"}
```

Finally, if you want to see all methods available on this network, you can simply query the */app/api* endpoint by running:

```
$ curl https://<ccf-node-address>/app/api -X GET --cacert networkcert.pem --cert user0_cert.pem --key user0_privk.pem
>
{"methods":["api","api/schema","code","commit","log/private","log/private/anonymous","log/private/historical","log/private/prefix_cert","log/private/raw_text/{id}","log/public","metrics","network_info","node/ids","primary_info","receipt","receipt/verify","tx","user_id"]}
```

To have more information on a specific method, you can query the */app/api/schema* endpoint with the appropriate method. For instance, to have more information on the */app/api* method, simply run:

Now we can finally propose this to the test CCF network by running:

```
$ ./scurl.sh https://127.12.222.235:42199/gov/proposals --cacert networkcert.pem --key
member0_privk.pem --cert member0_cert.pem --data-binary @new_member.json -H "content-type:
application/json"
>
{"proposal_id":4,"proposer_id":0,"state":"OPEN"}
```

In this case, a new proposal with id 4 has successfully been created and the proposer member has voted to accept it (they may instead pass a voting ballot with their proposal if they wish to vote conditionally, or withhold their vote until later). Other members can then vote to accept or reject the proposal.

Now we can other members' ballot. Create two other .json files (with Vim for instance) with the following content:

- *vote_reject.json*

```
{
  "ballot": {
    "text": "return false"
  }
}
```

- *vote_accept.json*

```
{
  "ballot": {
    "text": "return true"
  }
}
```

Now to simulate other members voting, we will just send their ballot (the json file), with the appropriate credentials and private keys to the right endpoint, which is /gov/proposals/4/votes (4 is the id of the proposition):

```
$ ./scurl.sh https://127.12.222.235:42199/gov/proposals/4/votes --cacert networkcert.pem --key
member1_privk.pem --cert member1_cert.pem --data-binary @vote_reject.json -H "content-type:
application/json"
>
{"proposal_id":4,"proposer_id":0,"state":"OPEN"}
```

Now that we made the member 1 reject, we will make the member 2 accept by sending the other ballot:

```
$ ./scurl.sh https://127.12.222.235:42199/gov/proposals/4/votes --cacert networkcert.pem --key
member2_privk.pem --cert member2_cert.pem --data-binary @vote_accept.json -H "content-type:
application/json"
>
{"proposal_id":4,"proposer_id":0,"state":"ACCEPTED"}
```

The proposal has now been accepted.

To view the details of the proposal we can do a GET on `/gov/proposals/{proposal_id}`:

```
$ ./scurl.sh https://127.12.222.235:42199/gov/proposals/4 --cacert networkcert.pem --key
member2_privk.pem --cert member2_cert.pem -H "content-type: application/json" -X GET
>
{"parameter":{"cert":"-----BEGIN CERTIFICATE-----
\nMIIBrzCCATSgAwIBAgIUTu47sG/Ziz4hgoeMhKzs/alrEYcwCgYIKoZIzj0EAwMw\nDjEMMAoGA1UEAwDYm9iMB4XDTIwMDcwO
TE0NTc0FoXDTIxMDcwOTE0NTc0Fow\nDjEMMAoGA1UEAwDYm9iMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAENhB3M5fWT5YQ\nn+v
BO10T9xt29CvYBsJyLCGeflqLFA4YDs7Bb3mMH46EiJg+BFT0HmIPtGw91qE5\nZEPMINQ2zuU0IU6uomPBi76pQ5vhm/2HDy3SL
DwRytrSDNqTXZXfo1MwUTAdBgNV\nHQ4EFgQUBchpeGuTHjy4XuwdgQqC3p0q0dEwHwYDVR0jBBgwFoAUBchpeGuTHjy4\nXuwdgQ
qC3p0q0dEwDwYDVR0TAQH/BAUwAwEB/zAKBggqhkjOPQDDAwNpADBmAJEA\nnmNPNpZvqn3piEepKGFJtqKtq+bZxUZuWZxxXILj4/
qnC1fLatJaMQ/DHRtCxcwU/\nAjEAtZe3LAQ6NtVIrn4qFG3ruuEgFL9arCpFGEBLFkVdkL2nFIBTp1L4C1/aJBqk\nnK9d9\n----
- END CERTIFICATE-----\n", "keyshare":"-----BEGIN PUBLIC KEY-----
\nMCowBQYDK2VuAyEA063rFGghB1p4DUvFQ6437ZGB1B8LNHzgNEjW5hRPHM=\n-----END PUBLIC KEY-----
\n"}, "proposer":0, "script":{"text":"tables, args = ...; return Calls:call(\"new_member\",
args)"},"state":"ACCEPTED", "votes":[[1, {"text":"return false"}], [0, {"text":"return
true"}], [2, {"text":"return true"}]]}]}
```

Here we can see at the end of the response that the state is “ACCEPTED”, and that members 0 and 2 voted yes, while member 1 voted no.

This concludes this module as well as [this starter guide](#) for a first look at CCF.

Copyright © 2020 Microsoft France. All right reserved.

Microsoft France
39 Quai du Président Roosevelt
92130 Issy-Les-Moulineaux

The reproduction in part or in full of this document, and of the associated trademarks and logos, without the written permission of Microsoft France, is forbidden under French and international law applicable to intellectual property.

MICROSOFT EXCLUDES ANY EXPRESS, IMPLICIT OR LEGAL GUARANTEE RELATING TO THE INFORMATION IN THIS DOCUMENT.

Microsoft, Azure, Office 365, Microsoft 365, Dynamics 365 and other names of products and services are, or may be, registered trademarks and/or commercial brands in the United States and/or in other countries.