E-book Series

Microsoft

A guided tour of Microsoft Entra Verified ID

Part 1 - A first walkthrough to illustrate how to get started with verifiable credentials in your Azure Active Directory (Azure AD) tenant



aka.ms/verifyonce

This page is intentionally left blank.

Notice

Microsoft Entra Verified ID, formerly known as Azure Active Directory Verifiable Credentials, is part of the Microsoft Entra family of products. Learn more about the Microsoft Entra family of identity solutions.

Microsoft Entra Verified ID is included with any Azure Active Directory subscription, including Azure AD Free, and is now generally available. Read the announcement.

The end-to-end illustrated walkthrough provided in the following pages is extrapolated from the documentaion Microsoft Entra Verified ID documentation available on docs.microsoft.com. **Please refer to this documentation for the latest information that pertains to Microsoft Entra Verified ID**.

MICROSOFT DISCLAIMS ALL WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, IN RELATION WIT H 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: September 2022
Version 1.0a

© 2022 Microsoft France. All rights reserved.



01 /
About this guide

02 /
Fullfil the prerequisites for your testing environment

03 /
Create, issue and verify your own Directory-based employees' credentials

04 /
Export or import your employees' credentials

05 /
Suspend or revoke employees' credentials

06/
As a conclusion

About this guide



This guide is about illustrating how easy it is to leverage the core functionalities of Microsoft Entra Verified ID to authenticate users.

In this first part, after discussing what a decentralized identity is, you'll start with an introduction of Microsoft Entra Verified ID.

With this understanding, you will be invited to set up the Microsoft Entra Verified ID service in your own Azure AD tenant, and create a first credential where the claims come from a user profile in the directory of your Azure AD tenant. With directory-based claims, you can create Verifiable Credentials of type VerifiedEmployee, if the users in the directory are employees, and thus issue them the digital counterpart of your employee physical badge.

So let's start with the beginning.

You said Microsoft Entra Verified ID, Què aco!?

What is Decentralized Identity?

Decentralized Identity represents the fourth and next stage of the models for (online) identity to which we advanced in the last three decades, starting with a **centralized identity** siloed to a specific system we were accessing, then moving to a **federated identity**, towards a **more user-centric identity**.

If you consider the above-mentioned stages, in most situation, what is noticeable is that your identity is in the hand on someone else, whatever entity it is, for instance, the so-called Identity Provider (IdP) with federated identity. As an employee or as a customer, such an IdP is involved in your transactions and beyond with the power to suspend or even worse revoke your identity at some point of time, thus removing you in the digital world for all the connections you have in it based on this identity.

As such, Decentralized Identity, a **disruptive next step**, is more than an attempt to bring things we take for granted in the analog physical world into the digital world in a more authentic, natural, and transparent way:

1. Organizations can issue to us portable information about us, which can be later cryptographically verified when presented, and which are received in the meantime in a digital wallet we own and control for that purpose. At Microsoft, as you will see, we used the metaphor of digital identity card that can convey potentially any type of personal information known by these organizations: fact, attestation, etc.
2. Our decentralized identifiers (DID) are self-owned IDs representing the way we are identified in lieu of usernames and emails. They correspond to our identities and where to ask a presentation of the related issued identity cards, and are registered and updated on a verifiable data registry, such as a blockchain, a distributed ledger technology, i.e., a tamper-resistant audit log, or even a web system.
3. Relying parties can request the presentation of one or multiple portable identity cards from us by digitally interacting via our digital wallet, and by adequately querying these decentralized systems.

   We are moving here from an online digital world based on authentication and access control to the notion of authenticity, meaning verifiability.

4. We can revoke access at any time to a specific portable identity card that has previously been presented.
5. We can recover our portable identity cards on a new device when we lose our mobile device.

As such, Decentralized Identity defines a truly shift from an identity that is centrally controlled or owned to one owned by individuals, businesses or assets.

It is a new way of establishing trust by presenting a number of expected credentials for the sake of the transactions with indications where to get them if you don't have them yet in your digital wallet. One should note that it has to be nearly universal to provide value.

Over the time, the control plane will transform into the fabric of many to many interactions where trust is established within those interactions, without going back to one central point of 'authority' of the previous above-mentioned identity stages, from one-to-many interactions to many-to-many interactions.

What are Verifiable Credentials?

The above portable information, fact, attestation, etc. are called from the "standardization jargon" Verifiable Credentials (VC).

VC are at the heart of Decentralized Identity that enables to replace usernames and passwords to verify you are who you say you are quickly and easily, and to truly reproduce in the digital world how we interact in our everyday analog physical world.

They allow to put us and people in control of their personal information, enabling more trustworthy digital experiences while respecting people's privacy (and possibly benefitting from an advanced privacy with Zero-Knowledge Proof (ZKP) capable credentials).

As such, it's a tempered-resistant, privacy-enabled format to issue by third parties and later present by ourselves verifiable information about us. As per W3C recommendation Verifiable Credentials Data Model v1.1 from the W3C Credentials Community Group (W3C CCG), "a verifiable credential is a tamper-evident credential that has authorship that can be cryptographically verified. Like any credential, it's a set of one or more claims, i.e. an assertion about a subject, made by an issuer", i.e., an authoritative source.

As introduced above, a suject is defined through a decentralized identifier or DID, i.e., a portable URL-based identifier anchored in a data verifiable repository, typically a distributed system such as a blockchain or a distributed ledger, also referred as to a Decentralized Public Key Infrastructure (DPKI). The repository can alternatively rely upon a system like the DNS infrastructure.

Technically speaking, DIDs are globally unique identifiers linked to the considered data verifiable repository or DPKI metadata composed of JSON documents that contain public key material, authentication descriptors, and service endpoints. DIDs are enablers, and they are important for that, see W3C proposed recommendation Decentralized Identifiers (DIDs) v1.0. Anyone can create, own, and control one or more DIDs independently of any organization or government.

People will typically possess and hold over the time one or more Verifiable Credentials (VCs) for which they are ususally the subject - this is not always the case -, and that have been issued to them by various issuers, potentially any entity they have a relationship with, for example your company, your university, etc. in the above Identity trust fabric (for tomorrow).

As a holder, typically but not exclusively a user, they will store their credentials in a credential repositiory, e.g., a digital wallet, under their control - A verifiable credential (VC)can be easily ported from one repository to another without the need to reissue the considered credential -.

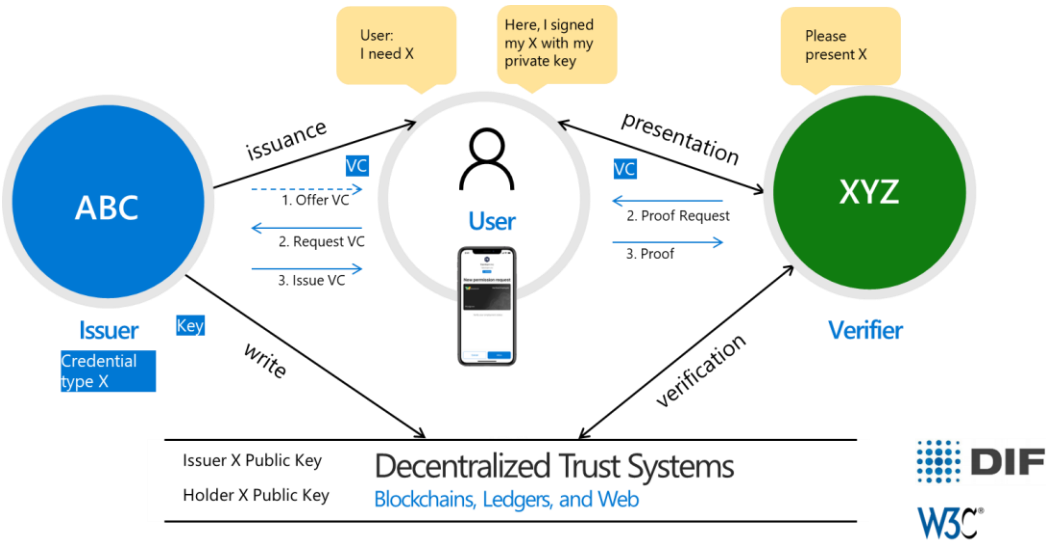
To prove their identity from a verifier's perspective, verifiable presentations can be generated from a VC. These presentations can also be cryptographically verified by a verifier, typically an entity. Verifiable presentations are packages of evidence - either credentials, or data derived from one or more credentials - built by holders to satisfy a verifier's requirements.

Verifiers learn with certainty which issuers have attested something by checking digital signatures against a verifiable data registry/DPKI.

How do verifiable credentials work?

As introduced above, Verifiable Credentials (VCs) facilitate interactions between three main persona, i.e., i) the issuer of a VC, ii) the holder of a VC, and iii) a verifier who ask the holder, or the owner if you prefer, of a specific VC to present it for verification:

- The issuer, e.g. your company, creates and issues VCs. For example, credential type X (with firstName and lastName).
- The holder requests, receives, holds and shares VCs. For example, credential type X (John Doe).
- The verifier asks for proof based upon VCs and verifies them.



They interact asynchronously like in the real world with physical card. Think of your national ID card, the way to obtain it, and how you use it. In the digital world, the trust is (immutably) anchored on a number of verifiable data registries/decentralized systems like blockchains and ledgers.

In terms of workflows:

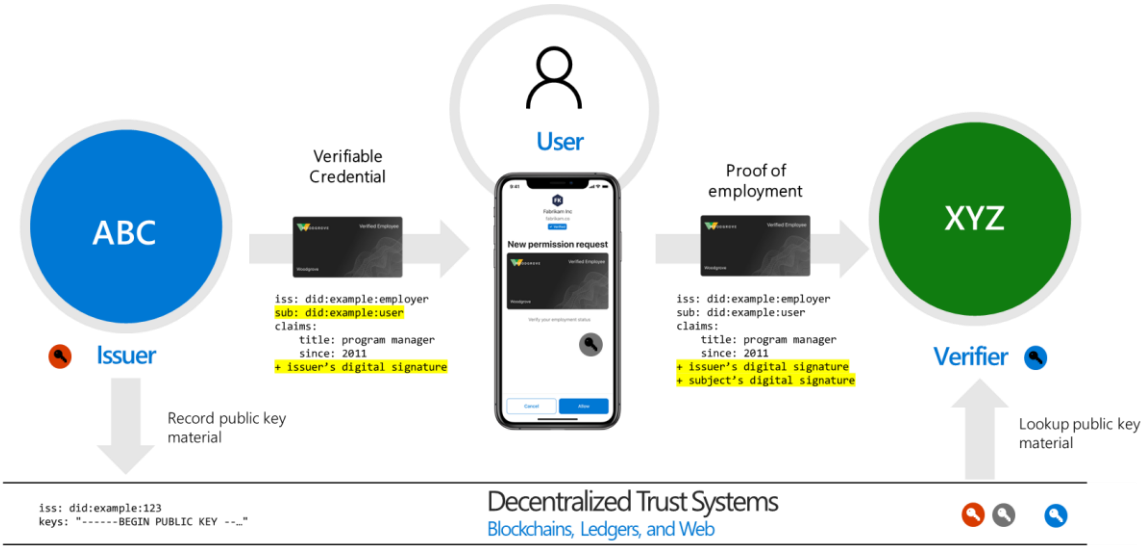
1. The issuer publishes its own DID and write it on a verifiable data registry so that everyone will be aware of the DID of the issuer. Later in this guide, a fictitious company will be issuing Employee card and other cards to its employees and customers.

   The issuer also publishes a schema called credential definition on the verifiable data repository/DPKI so that everyone can obtain information about the attributes of the verifiable credentials (e.g., an Employee Card). For example in our later illustration, the schema of the Employee Card will be (employeeNumber,) firstName, lastName, etc.

The issuer can also revoke an issued credential if they have to, and that too will be published on the verifiable data repository being leveraged. For example, if an employee leave the company, the company will revoke the related Employee Card that was issued to them. So the verifier will easily be able to see on the verifiable data registry that this Employee Card is non longer valid.

2. As already outlined, the holder is the subject to whom a VC will be issued. For example later in this guide, John Doe will hold an Employee Card issued by the fictititious company. The company will issue the card to John Doe and sign it digitally. John will store the issued Employee Card in his credential repositiory, e.g., Microsoft Authenticator.
3. The verifier can asked the holded to show one or multiples VCs, e.g. an Employee Card. The holder will digitally sign the presentation.

   The verifier will verify the digital signatures and the related DID on the verifiable data registries.



So left or right, the issuer issues a credential, and the user presents it, not all that different, but here's what's different under the hood.

What we've done is broken this up into two parts, 2 responsibilities if you will.

The issuer's responsibility is saying that the issued VCs, i.e., these facts, attestations, etc., are true, and then the sole responsibility for presenting it belongs to the user, i.e., the holder. The way that makes this technically possible is that the cryptographic keys used to present these credentials belong only to the holder are bound to the credential repositiory, e.g., as later illustrated in this guide, that mobile device to that authenticator app, to their biometric, to their PIN to do so.

So now when we talk about revocation, there's two different kinds of revocation that can occur. So, when a verifier verifies this Employee Card, i.e., a proof of employment, they can check "*Are the keys associated with the issue are still valid?*", and they can check that on the verifiable data repository being used.

This is how this works everywhere as opposed to having to federate and have to integrate with the issuer's identity system. This is what is painful today for verifiers and application developers versus now you say "*I've just talked to the user, and I can check revocation directly on the blockchain of the keys*", not the data, just the keys, and specifically public information, public key, as opposed to any personnaly identifiable information (PII).

And then similarly the holder can revoke their signature. They are key to say "*Hey look I presented it to you one time, but it's no longer valid for you*". So now we're a two-key system, much like how bank lockboxes work: one belongs to the bank, one belongs to you. Unless both keys turn, the lock box doesn't open. And the user is at the center.

Why starting to use verifiable credentials?

With the previous stages of the models for (online) identity, customers rely on a centralized identity system (or even muliple ones…) to secure access to corporate resources. However, enabling use of credentials for utility beyond their organization (e.g., proof employment for bank loan, alumni, etc.) is complex and comes with compliance risk. In contrast, identity documents from our everyday lives, like a driver's license or passport, are well suited for utility beyond travel (e.g. age or residency).

Considering the above introduction on both the Decentralized Identity, decentralized identifiers (DIDs), and Verifiable Credentials (VCs) themselves, the main benefits resulting from this disruptive next step are follows for everyone:

- **Privacy**. Today, your identity data is spread across countless entities and accounts, creating greater risks of fraud or breach. People once kept their valuable identity documents under lock and key. No matter who you are, you want some control over your privacy, control over your data. VCs are private:
    - The holder can choose to positively respond to a presentation request and what attributes they want to disclose – Zero-Knowledge Proof (ZKP) capable credentials will even allow to verify data without having to share the sensitive data required to verify something -. For example, the holder can show their Employee Card with the employee number and hide nay other attribute.
    - The holder is always in control of the relationship with the verifiers. They know what data was shared and when - there's an audit trail -, and can revoke that relationship at any point of time.
- **Portability**. VCs are yours to store in your credential repository, e.g., a digital wallet and share with whomever you want. The holder is not **locked-in** to the organization who issued a specific verifiable credential.

In turn, for organizations, in B2B, B2C, and G2C scenarios, being able to verify identity it's crucial. As you know, digital identity is a prime target for cybercrime and ransomware attacks. Compromised or fraudulent credentials are a very real and critical threat to public and private organizations today.

Organizations are also questioning whether it's even worth it to capture and manage personally identifiable data considering the laws and regulations that apply depending on your geography, like the EU's General Data Protection Regulation (GDPR), the U.S. state privacy laws, and many others across the world, and the resulting set of protections, including consumer control, business obligations, transparency requirements, and children's privacy enhancements.

The trust fabric provides a whole set of benefits:

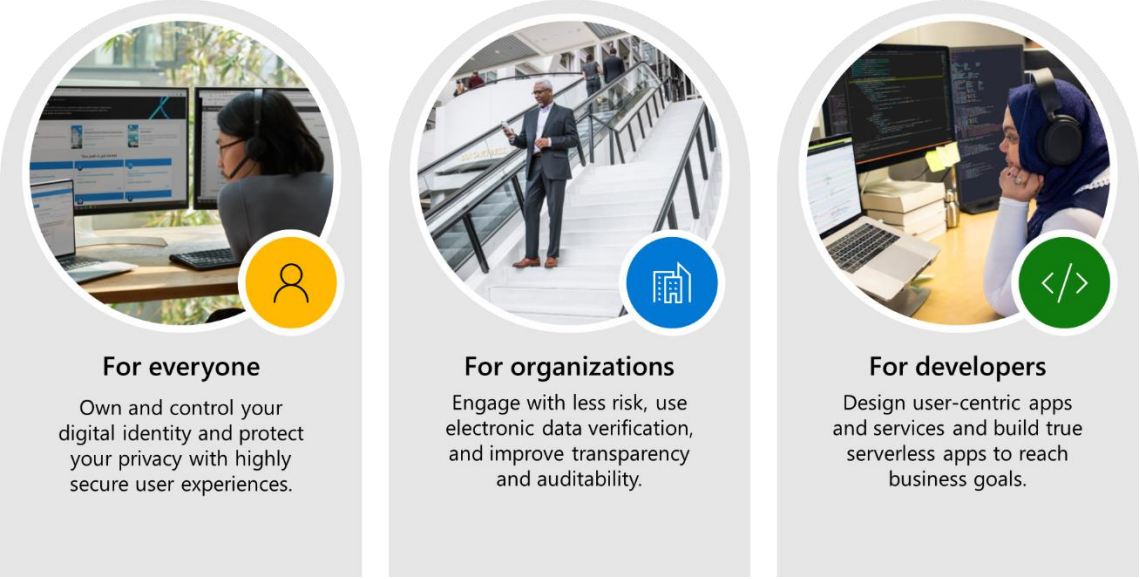
- **Security**. VCs are **tamper-proof** through the use of state-of the-art cryptography.
- **Availability**. VCs can be verified **anywhere, at any time**.
- **Interoperability**. In addition, the open standards and interop profiles (being) developed allow to build compatible digital wallets and enact and interoperable ecosystem of organizations, workplaces, governments, schools, institutions, and individuals participating in the trust fabric (with the user granting permission and access to the attestation(s)) (see below section).

And from a developer perspective, developers need a quick easy resource to enable to reach business goals. Unlike credentials issued by current centralized identity systems, VCs are standards-based (see below section). This makes it easy for developers to understand and does not require custom integrations of any sort.

Decentralized Identity is a standards-based framework that enables everyone to manage their digital identity and share it safely online with organizations. Organizations can verify someone's identity instantly without exposing personal data by using Verifiable Credentials (VCs).

As such, Decentralized Identity and Verifiable Credentials provide more secure, transparent digital experiences, giving individuals who ever they are more control over their identity, greater trust, and streamlined processes.

Individuals, organizations, etc. each store and manage their own identity data and transact in a distributed manner.



For more information about Decentralized Identity and Verifiable Credentials:

1. Understand the basics of decentralized identity via our five-part blog series.
2. Learn more about the decentralized identity movement at aka.ms/ownyouridentity.
3. Quick overview: http://aka.ms/didexplained.

What is the Microsoft implementation?

We believe an open standards-based Decentralized Identity system can unlock a new set of experiences that give both individuals and organizations greater control over their data—and deliver a higher degree of trust and security for apps, devices, and service providers (SPs), a.k.a. relying parties (RPs).

To (contribute to) enact this next step and the trust fabric (of tomorrow), Microsoft is committed to working closely with customers, partners, and the community to as a whole unlock the next generation of Decentralized Identity–based experiences, and we're excited to partner with the individuals and organizations that are making incredible contributions in this space. If the trust fabric ecosystem is to grow, standards, technical components, and code deliverables must be open source and accessible to all.

An implementation led with open standards

See Microsoft Entra Verified ID-supported standards.

Microsoft is actively collaborating with members of the above-mentioned W3C CCG, the OpenID Foundation, the Decentralized Identity Foundation (DIF), and the wider identity community. We've worked with these groups to identify and develop critical standards, and the following standards have been implemented in our services and stack.

DIDs and VCs form the bedrock, and the rest of the standards world comes out of the DIF and/or OpenID Foundation with the OpenID for Verifiable Credentials (OpenID4VC) specifications family. The way you request and present these credentials is based on the industry standard OpenID Connect, which is what nearly 95% of applications on the Internet rely on for their identity exchange. This is how we believe such a technology can be compatible with existing applications. See recently published whitepaper OpenID for Verifiable Credentials.

As of this writing, within the DIF, Microsoft contributes to the definition of the JWT VC Presentation Profile, i.e., a set of specifications to enable the interoperable presentation of VCs between digital aallets and verifiers/relying parties (RPs): https://aka.ms/vcinterop.

The VC data model indeed does not prescribe standards for transport protocol, key management, authentication, query language, etc. As a result, implementers must decide which standards to use for their presentations without a guarantee that others will support the same set of standards. This profile aims to provide a path to interoperability by standardizing the set of specifications that enable the presentation of JWT-VCs between implementers. Ultimately, this profile will define a standardized approach to Verifiable Credentials so that developers, distributed apps, and systems can share credentials through common means.

See for additional details:

- Decentralized Identifiers (DIDs) v1.0 (W3C).
- Verifiable Credentials Data Model v1.1 (W3C).
- Self-Issued OpenID Provider v2 (OpendID).
- OpenID for Verifiable Presentations (OpenID).
- Presentation Exchange v1.0.0 (DIF).
- Web DID Method.
- ION DID Method.
- OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1 (OpenID).
- Sidetree v1.0.0 (DIF).
- Well Known DID Configuration (DIF).
- Identity Hub - Decentralized Web Node 0.0.1 Predraft (DIF).

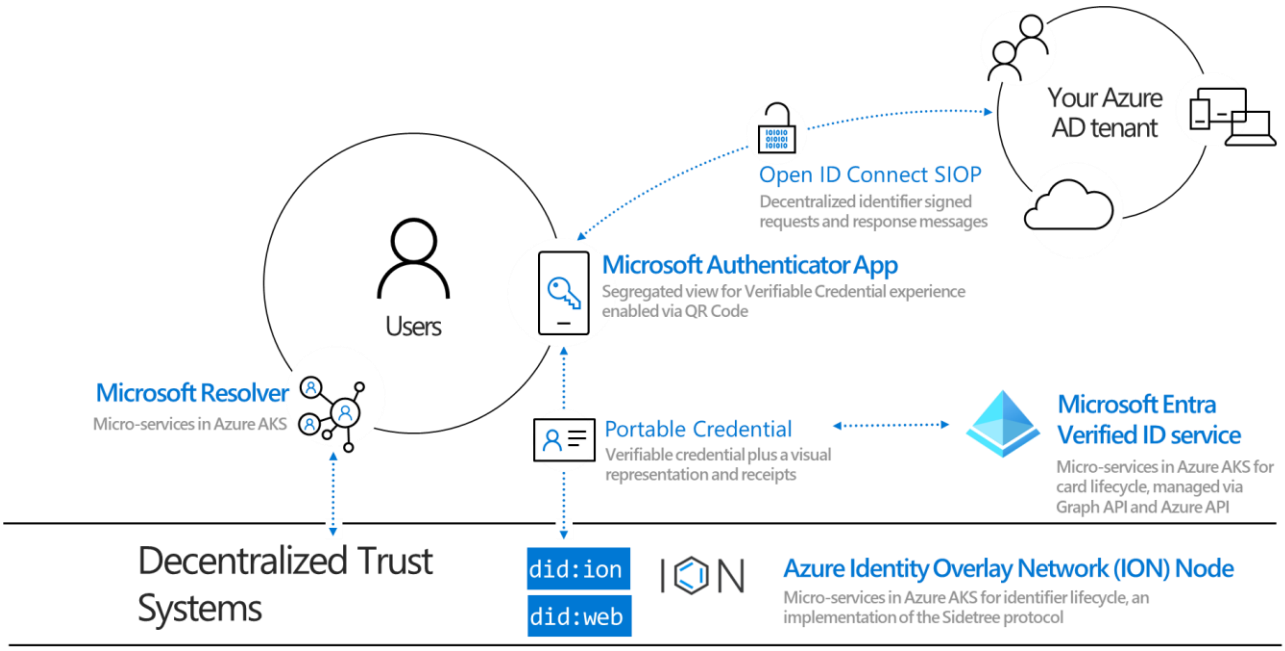
The current Microsoft stack

Microsoft Entra Verified ID is a managed service for verifiable credentials. It provides organizations with a scalable, standards-based way to issue and verify digital credentials with end users or other organizations as part of an identity verification process, such as:

- Faster remote onboarding to validate identity information for trustworthy self-service enrollment and reduced time-to-hire.
- Verify access to high value apps to quickly verify an individual's credentials and status to grant least-privilege access with confidence.
- Easy account recovery to replace support calls and security questions with a streamlined self-service process to verify identities.

With Microsoft Entra Verified ID, organizations can easily and more securely verify workplace credentials, education status, certifications, or other unique identity attributes in seconds. During the preview period, customers from around the world have issued and verified credentials across a variety of scenarios, ranging from remote onboarding at work, and collaboration across business boundaries to enabling education beyond the campus. Customers have issued and verified tens of thousands of credentials using our platform across a diverse set of industries to enable the previous processes.

Plus, it respects the privacy of individuals, corporate end-users, contractors, students, consumers, citizens, etc. who can own and control their own identity.



As of this writing, with Microsoft Entra Verified ID, the Microsoft implementation emcompasses:

1. Azure Active Directory (Azure AD) as the directory for the issuer.
2. Identity Overlay Network (ION) **network** (in preview) i.e. a decentralized Layer 2 open, permissionless network based on the purely deterministic Sidetree blockchain-agnostic protocol, which requires no special tokens, trusted validators, or other consensus mechanisms; the linear progression of Bitcoin's time chain is all that's required for its operations like publishing DIDs.

   Microsofy has open sourced a npm package to make working with the ION network easy to integrate into your apps and services. Libraries include creating a new DID, generating keys and anchoring your DID on the Bitcoin blockchain. See ION – Booting up the network and Sidetree & ION.

   We also support a Web DID method in conjunction with the blockchain-based ION DID method that allows to bootstrap trust using a web domain's existing reputation.
3. Microsoft Authenticator as the user-side DID user agent/digital wallet to interact with VCs and DIDs on the above ION network or on the Web.

   Microsoft Authenticator enables real people to use decentralized identities and VCs:

   - **Request credentials.** Individuals can request VCs that are issued by organizations with which they have a relationship of any kind.

- **Present credentials in a privacy respecting, secure manner.** Individuals can manage and present VCs. What makes this unique is it enables users to control who can access them, even if VCs are issued by organizations – VCs can be self-issued -. Over the time, we will be adding capabilities like selective disclosure, derived claims (e.g. proof of age instead of birth date) and measures preventing correlation.
- **Recover credentials.** Individuals can manage credentials and recover from loss using a mnemonic-based (pass phrase) scheme. We continue to partner with the identity community (see above) to find more user friendly, elegant means of enabling recovery. In addition, we will provide custodial and social recovery solutions so users can choose a circle of trusted parties, across friends, family, and organizations.

Microsoft Authenticator creates DIDs, facilitates issuance and presentation requests for VCs and manages the backup of your DID's seed through an encrypted wallet file. Microsoft Authenticator supports Single Long Form DID to be used of every issuer and verifier in terms of relaying party exchange. Both Android and iOS devices are supported today.

4. **Microsoft Resolver**. An API that connects to our ION node resp. the web to look up and resolve DIDs using the did:ion method resp. the did:web and return the DID Document Object (DDO). The DDO includes verifiable data repository/distributed system/DPKI metadata associated with the DID such as public keys and service endpoints.

   Think of this as a DNS look up for this decentralized identity world. So, if you are an identity provider (IdP) on your own and you have your own keys, *how does somebody find out where to send the authentication request to get your credentials?* That's what the resolver allows you to do, instead of relying on a centralized service.

5. And the **Microsoft Entra Verified ID** service in Azure to:

   - **Issue credentials**. you can design and issue W3C Verifiable Credentials to represent proof of employment, education, or other fact, attestation, etc. A holder of such a credential can decide when, and with whom, to share it. Each credential is signed with either the `did:ion` method or the `did:web` method, and with cryptographic keys that the user owns and controls. These credentials can be presented and validated even if the issuing entity no longer exists.
   - **Request and verify credentials.** Applications can request and verify the authenticity of these credentials from any organization using APIs included as part of the service.
   - **Search and revoke issued credentials**. As part of the process of working with VCs, you not only have to issue and verify credentials, but sometimes you also have to revoke them regarless of your own unique reason's for wanting to do so: the indivual to whom you issued a credential for a specific context is no longer active, the fact or the attestation is no longer valid, etc.

   Interestringly enough, Microsoft Entra Verified ID takes an API first approach that makes it easy for you to integrate VCs in your existing apps and enables an easy jumpstart model to improve verifiability and compliance. As such:

   - The Request Service REST API enables you to issue and verify VCs.
   - The Admin API enables you to manage all aspects of the Microsoft Entra Verified ID service. It offers a way to set up a brand new service, manage and create VC contracts, revoke VCs, and completely opt out the service as well. This API is not covered in this guide.
   - The Network API enables you to search for published credentials in the Microsoft Entra Verified ID Network, where you and/or any other entity as an issuer can publish credential types to it and you and.or any other entity as a verifier, can search for published credential types and schemas in it. This API is not covered in this guide.

These are the building blocks of the Microsoft implementation with users at the center along with the complete open standards' stack around it to enable or fuel the previous principles. Everything is defined in the open and based on open standards (see previous section). They enable identity owners/holder/verifier to generate, present, and verify credentials. This forms the basis of trust between the participant entities.

Microsoft Entra Verified ID is indeed currently a free service that you can use either independently or with one of their premium Azure Active Directory (Azure AD) plans for high-value scenarios. Customers can get started with Microsoft Entra Verified ID for free with Azure AD Free license. Azure AD Premium license will be required for enterprise scale (more than fifty thousand transactions per month).

For more information, see Introduction to Microsoft Entra Verified ID.

The Microsoft open-sourced components

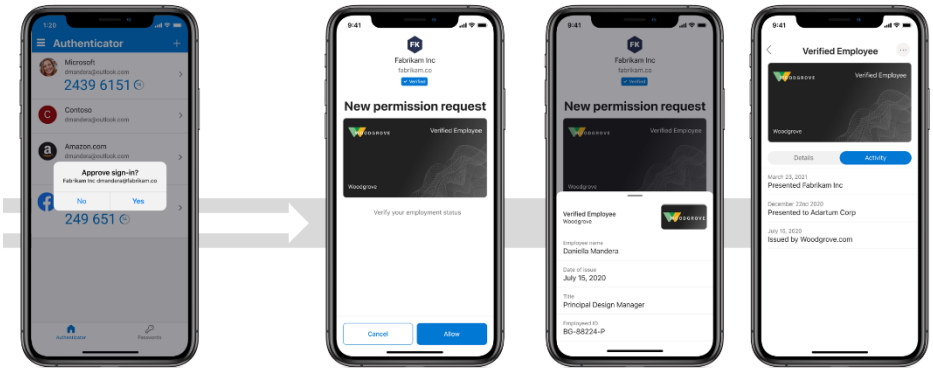
As of this writing, the following repositories are the open-sourced components of our implementation:

- SideTree, on GitHub.
- An Android SDK for building decentralized identity wallets, on GitHub.
- An iOS SDK for building decentralized identity wallets, on GitHub.



Towards a (new) employee/contractor journey

We believe Microsoft Entra Verified ID is a better way to verify identity.

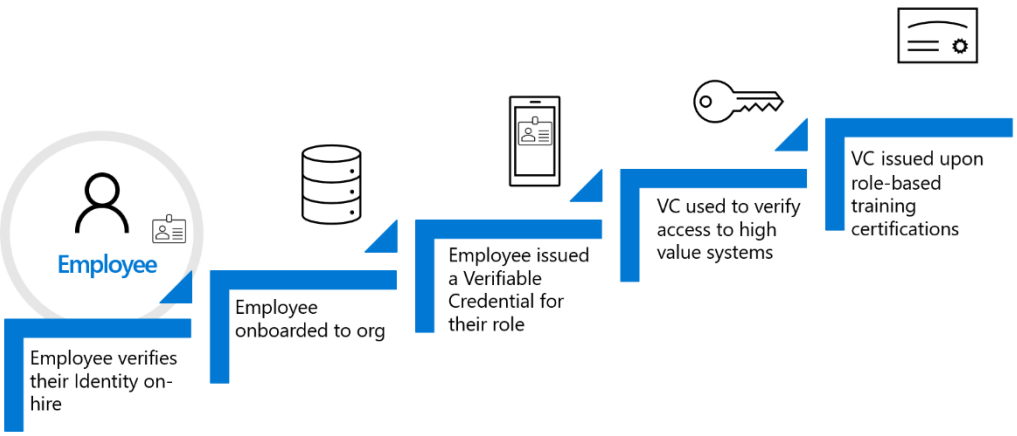


easy to use and secure | verifiable | transparent | convenient

So far Microsoft Authenticator has been good for ease of use and security, and we believe an end user can understand what is going on in the identity dance: "*I understand what is being asked of me and the concept is more human. It's verifiable, it's transparent*", so users understand what attributes are being shared and for what purpose, and ultimately, we believe it's more convenient because it helps them keep track of what information was shared with whom and for what purpose.

This guide features all the above building blocks as per the Microsoft Entra Verified ID and this first part illustrates in particular how to issue, verify, suspend and revoke verifiable credentials with Azure AD.

With the pandemic of COVID-19 and the remote or the hybrid modern work environment, being able to onboard remote employees and providing them suitable access is one of the use cases we increasingly hear about.



Passwordless onboarding of remote employees can be achieved today thanks to VCs and the identity proofing of government issued ID, and temporary access pass (TAP).

So, we worked with leading ID verification partners around the world to perform such digital ID verification remotely.

Our leading identity verification provider partners are able to match selfies with the picture in government issued ID, and actually cover 192 countries, 6000 different identification document types, and thousands of organizational attributes and individuals like *What your Social Security number? What's your residency, etc.* to enable you to remotely verify foundational identity attributes. See [Identity Proofing and Verification (IDV) Partner gallery for Entra Verified ID](#).

So, depending on your trust process for onboarding your (new) employee, you can ask for many, many things, which is relative to your burden that you want to put on the participant, and the level of trust you need to establish.

This could be as complicated as you like, or as simple as you want to make it, just like in the analog physical life you would have said "*Come see us in person, bring your government issued document*. *We're going to take a picture for your badge*".

Similarly you could also onboard contractors in the same way. Next comes secure access to high value systems. You could say "*Hey, if you can show me a Verified Employee card from Litware 369 Inc., and then you are allowed to come access to this high value system at Litware 369 Inc. »* or any other resource somewhere else. It's so much easier to do that than asking for setting up federations as we do today.

This can also apply to applications where passing a prior certification is required. So, instead of just relying on usernames and passwords which, if compromised, is an access to everything, what if you could ask for additional credentials such as that you really are this person, and that you have a certification from a third party.

End-users can present these credentials to quickly start a job, apply for a loan, or access secure apps and services - without having to repeatedly share their sensitive information -. Now, a bad actor has to compromise multiple disparate systems in order to gain access to what valuable resource.



You can already get a glimpse of what such an online process could be to onboard a new employee/contactor, and how it can translate in terms of user experience/journey with the live demo available at

aka.ms/vcdemo

The demo indeed illustrates a new remote employee at Woodgrove being onboarded before they're ready to start their first day on the job. Typically, a new employee would be required to go to the office, fill out forms, and receive a laptop. However the fictitious company has adopted cutting edge technology to accomplish these things digitally, streamline onboarding and discounted equipment purchases.

To illustrate the above and such a new employee/contractor journey, and get your hand « dirty » with Microsoft Entra Verified ID, formerly known as Azure AD Verifiable Credentials, this first part of the guide provides you with an end-to-end walkthrough to i) issue a VC where the claims come from a user profile in the directory of the Azure AD tenant, i.e., a so-called Directory-based Employee's VC, ii) verify it, and possibly iii) revoke it later for some reasons.

Guide elements

For the sake of this walkthrough, we provide the following elements:

- **Hands-on instructions** containing the most important steps and their outputs. These are meant to show you the core elements.
- **Code sample application** demonstrating how to use Microsoft Entra Verified ID functionality to issue and consume verifiable credentials, that you can access by downloading or cloning the following GitHub repository: **philber/entra-verifiedid-tour (github.com)**.

Let's start with the prerequisites.

Guide prerequisites

We assume here that you have a Windows 10 local environment or above. For example, Windows 11 in our illustration.

Install Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency

You can download and install Git:

- Download the Git for Windows and run it.
- In the pop-up window, click **Install**. Follow the instructions.

Git will help us to clone the sample code project and related resources to complete this guide.

Clone the repo using Git

To clone the sample code application's project on your Windows 11 local machine, open a PowerShell console, and run the following commands:

```
PS C:\> cd c:\
PS C:\> git clone https://github.com/philber/entra-verifiedid-tour.git
PS C:\> cd entra-verifiedid-tour\part-1
```

This code sample application in NodeJS for this first part is forked from the repo available at Azure-Samples/active-directory-verifiable-credentials-node (github.com).

The sample code application is provided in other languages, namely .NET, Java, and Python.

Whilst it does not feature the other flavors in terms of language, most of its content can apply to it in a similar way.

Repository organization

The repo contains a code sample application for using the above-mentioned Request Service REST API to issue and verify VCs with a credential contract which allows the Request Service REST API to pass in a payload for the VCs.

For the sake of this walkthrough, under the folder where you cloned the repo, e.g., *C:\>entra-verifiedid-tour* in our illustration, we will mainly be interested by the directory *part-1*, the .js and .json files in this directory, as well as the folders underneath:

- *CredentialFiles:* contains VCs display and rules definition files.
    - The *display* definition file provides the content to control the branding of the credential and styling of the claims.
    - The *rules* definition file provides the content to describe important properties of VCs, and the rules that apply. In particular, it describes the claims that subjects (users) need to provide before a credential is issued for them.
- *Public:* contains some html files for the code sample application front-end side.

So, at this stage, you're all set! It's high time to move on to the first module.

Fullfil the prerequisites for your testing environment

In order to complete this walkthrough, you'll need to start fulling a set of prerequisites for both your Azure testing environement as well as your local development environment.

Let's consider these prerequisites in order.

Fullfil the prerequisites for your Azure testing environment

Azure subscription

An Azure subscription is required to complete this guide. If you don't have Azure subscription, create one for free.

See Build in the cloud with an Azure account to subscribe.

Custom domain

A custom domain in a public DNS zone is also required to both resolve the host name of the code sample application, i.e., a web site on the Internet, and verify the organization that will issue VCs, for example the fictititious company Litware369 Inc. in our illustration.

For illustration purposes, we've opted to purchase and configure the domain litware369.com. You will have to choose in lieu of a domain name of yours.

For checking purpose, you can for instance use the domain search capability provided by several popular domain providers. For example, https://www.godaddy.com/domains with GoDaddy.

Make sure you can edit the DNS records for your custom domain. To edit DNS records, you need access to the DNS registry for your domain provider, such as GoDaddy. For example, to add DNS entries for litware369.com and www.litware369.com, you must be able to configure the DNS settings for the litware369.com root domain.

Whenever a reference to litware369.com is made in a procedure, it has to be replaced by the DNS domain name of your choice to reflect accordingly the change in naming.

See Buy a custom domain name.

Fullfil the prerequisites for your local environment

In order to complete this walkthrough, you'll also need in your local development environment:

1. Visual Studio Code or similar IDE.
2. PowerShell 7.
3. Azure Az PowerShell module.
4. NodeJS version 10.14 or higher.
5. ngrok to expose on the Internet through an instant, secure URL the above local Node server if needed.
6. And optionnaly Fiddler as a remote proxy to capture Android/iOS traffic.

Throughout this guide, we assume you have a Windows local machine for the purpose of the illustration.

This said, please note that the same exact above prerequisites can be installed on a macOS or a Linux machine instead, and all the provided instructions apply in a similar way on such environments as well. For the sake of brievety, we neither cover these configurations in the rest of this guide nor the use of the Windows Subsystem for Linux 2 (WSL2) with a Linux distro (e.g., Ubuntu 20.04.4 LTS).

In addition, you will also need to have a mobile test device with Microsoft Authenticator:

- Android version 6.2108.5654 or later installed.
- iOS version 6.5.82 or later installed.

Let's see how to install the above prerequisites.

Install Visual Studio Code or similar IDE

We will use Visual Studio Code throught this guide, but you are free to use the integrated development environment (IDE) of your choice instead.

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for your Windows local machine, as well as on macOS and Linux. It comes with built-in support for JavaScript, TypeScript and NodeJS and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

Visual Studio Code benefits from a gallery of extensions, which are add-ons that allow you to customize and enhance your experience in Visual Studio Code by adding new features or integrating existing tools. The Azure App Service extension extension allows you to quickly create, manage, and deploy your websites with Azure App Service, a fully-managed Platform as a Service (PaaS) that lets you deploy and scale web, mobile, and API apps.

Proceed with the following steps:

1. Download and install Visual Studio Code on your local machine.
2. Install the Azure App Service extension for Visual Studio Code.

Install PowerShell 7

See Installing PowerShell on Windows.

PowerShell was first introduced as a command-line alternative to the old and legacy Windows Command Prompt (CMD) on Windows. PowerShell 7 is a new edition of PowerShell that is cross-platform (Windows, macOS, and Linux), open-source, and built for heterogeneous environments and the hybrid cloud.

Throughout this guide, PowerShell 7 will be used to handle some tasks from the command line.

To install the latest version of PowerShell 7, perform the following steps:

1. Download the install package from GitHub. As of this writing, the version is PowerShell 7.2.3 LTS-x64, which has support for Microsoft Update.
2. Once downloaded, double-click the installer file and follow the prompts. The installer creates a shortcut in the Windows Start menu.

Install Azure Az PowerShell module

See Install the Az PowerShell module via PowerShellGet.

The Az PowerShell module is a set of cmdlets for managing Azure resources directly from PowerShell. The Azure Az PowerShell module is a rollup module. Installing it downloads the generally available Az PowerShell modules, and makes their cmdlets available for use.

To install the Azure Az PowerShell module, perform the following steps:

1. Open a PowerShell 7 window.
2. Check your PowerShell version:

```
PS C:\> $PSVersionTable.PSVersion
```

   PowerShell 7.0.6 LTS-x64, PowerShell 7.1.3-x64, or higher – here 7.2.3 - are the recommended version of PowerShell for use with the Azure Az PowerShell module. When run on these versions, Azure PowerShell has indeed no additional requirements.

3. Use the `Install-Module` cmdlet to install the Az PowerShell module for the current user only. This is the recommended installation scope:

```
PS C:\> Install-Module -Name Az -Scope CurrentUser -Repository PSGallery -Force
```

4. Set the PowerShell script execution policy to remote signed or less restrictive. See about_Execution_Policies.

```
PS C:\> Install-Module -Name Az -Scope CurrentUser -Repository PSGallery -Force
```

To connect to Azure with an authenticated account for use with cmdlets from the Az PowerShell modules, perform the following steps:

1. Open a PowerShell 7 window, and run the `Connect-azAccount` command.

```
PS C:\> Connect-azAccount
```

   When asked for, open a browser session on your Windows 11 local machine and navigate to at https://aka.ms/devicelogin.

   - Enter the authorization code displayed in your PowerShell 7 windows.
   - Sign in with your account credentials in the browser.

If no context is found for the current user, the user's context list is populated with a context for each of their first 25 subscriptions. The list of contexts created for the user can be found by running `Get-AzContext -ListAvailable`. To skip this context population, specify the `SkipContextPopulation` switch parameter.

See Sign in with Azure PowerShell.

2. If you have multiple Azure subscriptions, set the context to use the subscription that you'd like for the sake of this walkthrough.

```
PS C:\> Set-AzContext -Subscription <your_subscription_id>
```

Where <*your_subscription_id*> is the ID of your subscription retrieved from the Azure portal. For example in our illustration :

```
PS C:\> Set-AzContext -Subscription "8848a529-9d69-4049-8469-8218547a61e2"
```

After executing this cmdlet, you can disconnect from an Azure account using the `Disconnect-AzAccount` cmdlet.

Install NodeJS

See How to install nodeJS for more details.

NodeJS allows you to run simple NodeJS apps, which is the perfect fit for the sake of this walkthrough to illustrate how to leverage the Request Service REST API of the Microsoft Entra Verified ID service. The Request Service REST API can be use with any other language.

You can install NodeJS:

- Either via a .MSI installation file: navigate to the NodeJS website and download the LTS version - At the time of writing this section, the version is 16.14.0 LTS (17.7.1 current) – and execute the installation file.
- Or by using the Chocolatey repository manager. Open an elevated PowerShell 7 window and run the following command:

```
PS C:\> choco install nodejs
```

After successful installation, enter `node –v` in your command line interface to see the version of NodeJS you have installed

Install ngrok

ngrok is a cross-platform application that can create a tunnelling or forwarding URL, so that internet requests reach your local machine for local devlopment.

When needed, you will use this tool to expose our localhost on a public URL.

See ngrok - secure introspectable tunnels to localhost to download and install ngrok. You can also use here the Chocolaty repository manager.

```
PS C:\> choco install ngrok
```

Please note that you need to sign up for an account to get and install an Authtoken to use the tool. So, regardless of the above approach, proceed with the following steps:

1. Open a browser session and navigate to the ngrok web site at https://ngrok.com/.
2. Click **Sign up for free** and follow the instructions.
3. Back to the home page, now click **Login** and enter your related credentials.
4. On the left pane, select Authtoken to get your personal Authtoken. It is used to authenticate the ngrok agent that you downloaded. Take a note of it.
5. Now open a PowerShell 7 window and run the following command:

```
PS C:\> ngrok authtoken <yourAuthToken>
```

Replace the placeholder by your own value. The Authtoken is saved in the default configuration file.

At this stage, you are ready to further configure your Azure testing environment to issue your own Verifiable Credentials (VCs) with Microsoft Entra Verified ID.

Create, issue and verify your own Directory-based employees' credentials

Microsoft Entra Verified ID safeguards your organization with an identity solution that's seamless and decentralized. The service allows you to both issue and verify credentials.

In this section, we will illustrate how to issue your own Verifiable Credentials (VCs) where the claims come from a user profile in the directory of the Azure AD tenant. With Directory-based claims, you can create VCs of type VerifiedEmployee, if the users in the directory are employees. Such a capability is useful for your users when they need to present proof of employment, internally vs. externally.

In a few simple steps covered hereafeter, you will see how to configure your Azure AD tenant to produce a Directory-based employee VCs that meet the needs of your particular use case.

It is however important to plan your VC solution so that in addition to issuing and or validating credentials, you have a complete view of the architectural and business impacts of your solution.

See Plan your Microsoft Entra Verified ID issuance solution and Microsoft Entra Verified ID architecture overview for more details.

Unless noticed otherwise, all the required resources will be created in the FranceCentral region. If you want to setup the environment in another region, you will need to adapt the instructions accordingly.

The next sections will guide you throughout the setup a VCs issuer, and demonstrate how to issue a configured Verified Employee credential to an employee, i.e. a so-called "digital employee badge". Additional considerations and explanations will be added by the end.

Let's start with the creation of an Azure AD tenant.

Create your Azure AD tenant

See Create a free Azure Active Directory developer tenant for more details.

To get started, you first need an Azure AD tenant. In the world of Verifiable Credentials (VCs), your Azure AD tenant will represent your organization. When your tenant is enabled for the Verifiable Credentials with the Microsoft Entra Verified ID service, it will be assigned a decentralized identifier (DID) and be equipped with an issuer service for issuing credentials. Any credential you issue will be issued by your tenant and its DID.

Create an Azure AD tenant if you do not already have one. As of this writing, this can be a Free or Premium one, see Expanding the public preview of verifiable credentials.

(To instead activate a free trial of Azure AD Premium editions (P1 or P2) in your tenant, you can join the Microsoft 365 Developer Program and create a time limited sandbox tenant.)

From the Microsoft Entra Verified ID perspective, to issue Verified employee credentials, your directory needs to be populated with user accounts to source the claims to issue.

Conversely, for Custom credentials as covered in the second part of the guide, the tenant can be completely empty. It can be understood here as a "technical tenant" to handle all the specifics that pertain to VCs. You can use instead an existing Azure AD (B2C) tenant if any, or any OpenID Connect (OIDC) conformant identity provider.

Your Azure subscription must have the above Azure AD tenant as its directory. If it doesn't, you need to switch directory for the considered subscription. You also need to ensure that you have the global administrator permission for this directory.

Check the region of your Azure AD tenant

As of this writing, the Microsoft Entra Verified ID service is available in both the Azure North America region and the Azure Europe region with different endpoints.

If your Azure AD tenant is located in a country or a region in Europe, your Microsoft Entra Verified ID service will be set up in Europe, with your Verifiable Credentials data located and processed in the Azure Europe region. Learn more about data residency in Azure and our commitment to protecting customer data as part of the Microsoft EU Data Boundary program.

Proceed with the following steps:

1. Open a browser session and navigate to Azure portal at https://portal.azure.com.
2. Sign-in with your administrative credentials.
3. Go to **Azure Active Directory** for the subscription you will use for your Microsoft Entra Verified ID deployment.
4. Under **Manage**, select **Properties**.
5. See the value for **Country or Region**.



As already indicated, the resources of this walkthrough are created in the FranceCentral region.

Take a note of your Tenant ID

The Tenant ID will be used to later connect to your Azure AD tenant.

From the previous **Properties** page, take a note of the Tenant ID. For example, `164f6854-a553-4084-9dc8-499819f6fceb` in our illustration.

Create a test user

To later issue the Verified Employee credential, let's create a test user. For example John Doe in our illustration. It could be instead TestUser1, or whatever you want.

Proceed with the following steps:

1. Open a PowerShell 7 windows, and run the `Connect-azAccount` cmdlet to authenticate as per section **Install Azure Az PowerShell module**.

```
PS C:\> Connect-azAccount -tenantID <yourTenantID>
```

   Replace <*yourTenantID*> with your Azure AD tenant ID as per section **Take a note of your Tenant ID**. For example, in our illustration:

```
PS C:\> Connect-azAccount -tenantID "164f6854-a553-4084-9dc8-499819f6fceb"
```

2. Set the password of the user.

```
PS C:\> $secureStrPassword = ConvertTo-SecureString -String '<yourPassword>' -AsPlainText -Force
```

   Replace the placeholder value in brackets with the above value. For example, in our illustration:

```
PS C:\> $secureStrPassword = ConvertTo-SecureString -String 'Gr34!%abc853' -AsPlainText -Force
```

3. Then create the test user.

```
PS C:\> New-AzADUser -DisplayName "John Doe" -UserPrincipalName johndoe@litware369inc.onmicrosoft.com -Password
$secureStrPassword -MailNickname "JohnD" -ForceChangePasswordNextLogin
```

   Modify the above command to accommodate your configuration and the user account to create in your Azure AD tenant.

4. Now open a browser session, navigate to Azure portal, and sign-in with your administrative credentials.
5. Go to **Azure Active Directory**.
6. Under **Manage**, select **Users**.
7. Find the newly created user, select to view profile. In the **Overview**, select **Properties**. Update the following attributes by clicking **Edit** for the appropriate sections:
   a. **Job Title** under **Job Information**. specify a title of your choice and click **Save**.
   b. **Email** under **Contact Information**: set an email address - note that this doesn't have to be an existing email address - and click **Save**.
   c. **Photo**: click the camera icon. In **Upload new photo**, click **Browse for files**, select JPG/PNG file with low, thumbnail like, resolution, and click **Upload**.
8. Open a new, private, browser session and navigate to the MyApps page at https://myapps.microsoft.com/, and sign in with your new user. For example, johndoe@litware369inc.onmicrosoft.com in our illustration. You'll be prompted to change your password.

Set up the user for Microsoft Authenticator

Your test user needs to have Microsoft Authenticator setup for their account.

To enable Microsoft Authenticator on the previous test user account, proceed with the following steps:

1. On your mobile test device (see section **Fullfil the prerequisites for your local environment** above), open Microsoft Authenticator, go to the Authenticator tab at the bottom and tap + sign to add a new account. Select **Work or school account**.
2. At the prompt, select **Sign in**. Don't select **Scan QR code**.
3. Click +, and sign in with the above test user's credentials in your Azure AD tenant.
4. Microsoft Authenticator will launch https://aka.ms/mfasetup in the browser on your mobile device. You will need to sign in again with your test user's credentials.
5. In the **Set up your account** in the app, select **Pair your account to the app by clicking this link**. The Microsoft Authenticator app and opens and you see your test user as an added account.

Note that if https://aka.ms/mfasetup launches without prompting you to sign in, this means you have already set up authenticator for another user on this device. When already configured with a user, Microsoft Authenticator signs you in automatically. Sign out the browser's currently logged in user and then repeat the steps above. If you zoom in on the page, you find the **Sign out** button at the top right corner.

Setup a Verifiable Credentials Issuer and Verifier

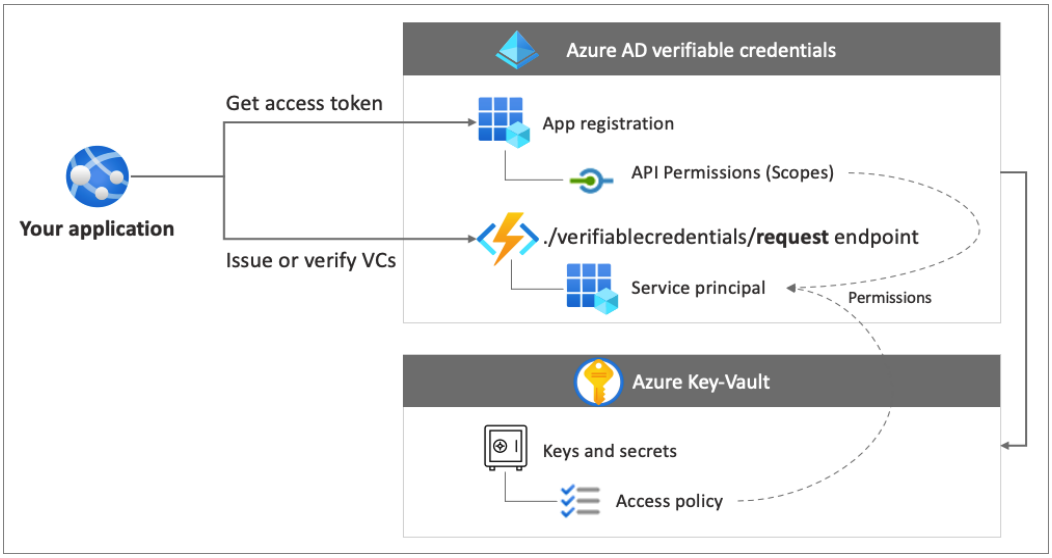
See Tutorial - Configure your tenant for Microsoft Entra Verified ID.

At this stage, we assume that:

- You have an Azure AD tenant with an active Azure subscription and that your local machine fullfills the prerequisites outlined in the previous section **Fullfil the prerequisites for your testing environment**.
- You have the global administrator permission for the directory you want to configure.

With that in place, let's see how to configure your Azure AD tenant so it can use the aforementioned the Microsoft Entra Verified ID service.

The following diagram illustrates the Microsoft Entra Verified ID architecture and the necessary Azure components you will have to configure to eventually get the code sample application working and issue VCs.



Specifically, you will need to complete the following steps:

- Create service principals for the Microsoft Entra Verified ID service and the Verifiable Credentials Service Request.
- Create your Azure Key Vault instance to store the keys used by the Microsoft Entra Verified ID service, and thus set the appropriate access policy.
- Register your application in Azure AD to set the appropriate API permissions to call the Verifiable Credentials Service Request, and thus obtain an OAuth 2.0 access token.
- Set up Microsoft Entra Verified ID, the organization, a private container storage for the Verifiable Credentials configuration files, etc.

  Once the above Azure components are in place, you will need to configure the code sample application.

The above steps are covered below in order for setting up the Microsoft Entra Verified ID service step by step, and whenever necessary, additionnal considerations and/explanations are provided.

While PowerShell commands are widely used in these steps to ease the setup and the configuration of your environment, one should note the availability of an ARM template along with a series PowerShell scripts to completly automate both the setup and the configuration. Please note that the resources are provided as-is.

The objectives we pursue here is to help you understand the setup and the configuration of the needed components, so we do not leverage that ARM template.

Create service principals for the Microsoft Entra Verified ID service

Proceed with the following steps:

1. Open a PowerShell 7 windows, and run the `Connect-azAccount` cmdlet to authenticate as per section **Install Azure Az PowerShell module**.

```
PS C:\> Connect-azAccount -tenantID <yourTenantID>
```

   Replace the placeholder value in brackets with your Azure AD tenant ID as per section **Take a note of your Tenant ID**.

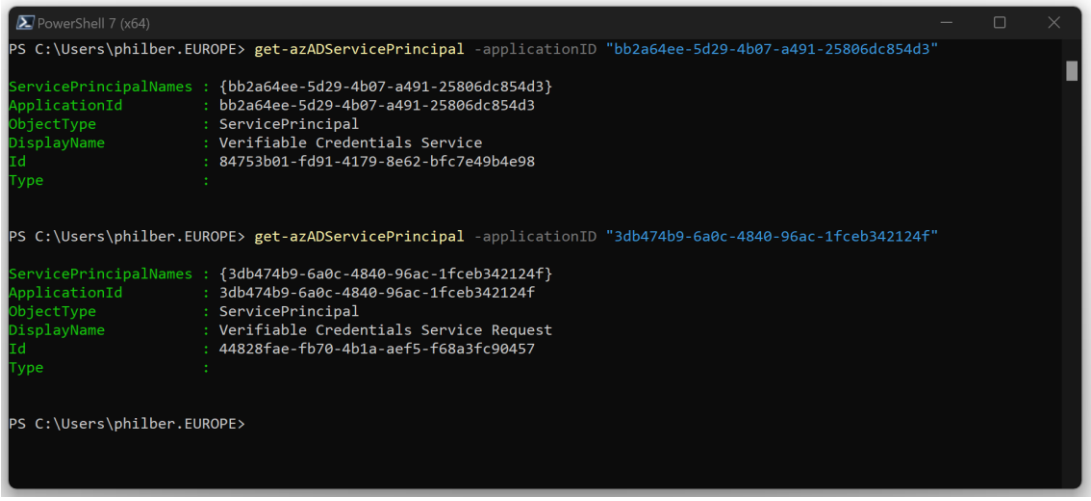
For example, in our illustration:

```
PS C:\> Connect-azAccount -tenantID "164f6854-a553-4084-9dc8-499819f6fceb"
```

2. Create the required service principals.

```
PS C:\> new-azADServicePrincipal -applicationID "bb2a64ee-5d29-4b07-a491-25806dc854d3"
PS C:\> new-azADServicePrincipal -applicationID "3db474b9-6a0c-4840-96ac-1fceb342124f"
```

The applicationID `bb2a64ee-5d29-4b07-a491-25806dc854d3` and `3db474b9-6a0c-4840-96ac-1fceb342124f` respectiveley refer to the Verifiable Credentials Service and the Verifiable Credentials Service Request service principals.



Create your Azure Key Vault instance

In the Microsoft Entra Verified ID, you have complete control and management of the cryptographic keys your tenant will use to digitally sign Verifiable Credentials (VCs). To issue credentials, you must provide Azure AD with an access to your own instance of Azure Key Vault.

Azure Key Vault is a cloud service that enables the secure storage and access of secrets and keys. Your Verifiable Credentials service stores public and private keys in Azure Key Vault. These keys are used to sign and verify credentials. Please note that each Key Vault transaction will incur a cost to your Azure subscription, review the Key Vault pricing details here.

If you don't have an instance of Azure Key Vault, you need to create one.

Create a key vault

See Azure Key Vault for more details.

On your local machine, proceed with the following steps:

1. Open a PowerShell 7 windows, and run the `Connect-azAccount` cmdlet to authenticate as per section **Install Azure Az PowerShell module**.

2. Create a resource group:

```
PS C:\> New-AzResourceGroup -Name <yourResourceGroup> -Location "France Central"
```

Replace the placeholder value in brackets with your own value. For example in our illustration:

```
PS C:\> New-AzResourceGroup -Name "VerifiableCredentialsTest" -Location "France Central"
```

3. Next create a Key Vault. When doing this step, you need some information, and we recommend the following resource details:

Detail	Recommended
Region	*FranceCentral*
Pricing Tier	*Standard*
Key Vault name	*VCIssuer-Vault**
Resource group name	*VerifiableCredentialsTest*

*Although we use "*VCIssuer-Vault*" as the name for the Key Vault throughout this illustration, you must use a unique name.

Run the following command:

```
PS C:\> New-AzKeyVault -VaultName "<yourVault>" -ResourceGroupName "<yourResourceGroup>" -Location "France
Central"
```

Because the command does not specify a value for the SKU parameter, it creates a Standard key vault.

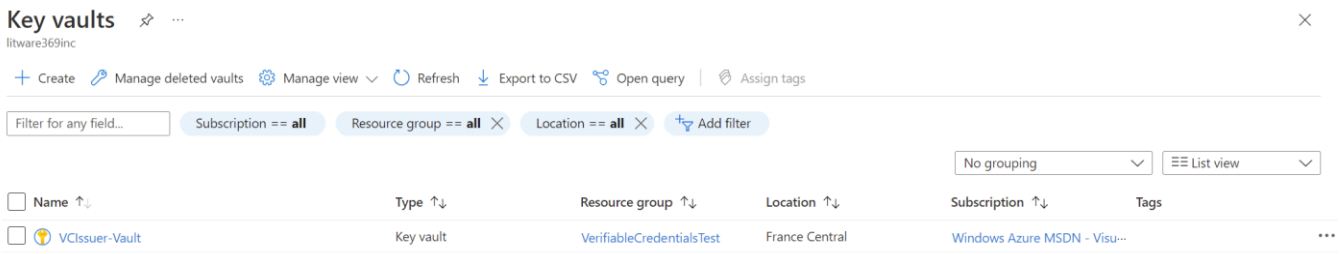
Replace the placeholder value in brackets with the above value. For example, in our illustration:

```
PS C:\> New-AzKeyVault -VaultName "VCIssuer-Vault" -ResourceGroupName "VerifiableCredentialsTest" -Location
"France Central"
```

The output of the `New-AzKeyVault` cmdlet shows properties of the newly created key vault. Take note of the two properties listed hereafter:

d. `name`: For example, "*VCIssuer-Vault*" in our illustration. You will use this name later to obtain an access token.
e. `vaultURI`: For example, https://vcissuer-vault.vault.azure.net/ in our illustration. Applications and scripts that use your vault through its REST API must use this URI.

4. From the Azure portal, search for "*Key vaults*". You should see your newly created Key vault.



After the vault creation, by default, the account that creates the key vault is the only one with access. The Verifiable Credentials Admin user as well as the Verifiable Credentials Service and the API Verifiable Credentials Request Service need access to the key vault.

Set access policies for the key vault

A Key Vault access policy defines whether a specified security principal can perform operations on Key Vault keys (and secrets). You need to set access policies in your key vault for both the administrator account of the Microsoft Entra Verified ID service, and for the Verifiable Credentials Service and the Verifiable Credentials Request Service service principals that you created.

At this time, do not remove any permissions!

Add an access policy for the Verifiable Credentials Admin user

You must configure the key vault with an access policy that allows the account used during configuration to create and delete keys. The account used during configuration also requires permission to sign to later create the domain binding for VCs.

If you use the same account while testing, you will modify the default policy to grant the account sign permission, in addition to the default permissions granted to vault creators.

From the previous PowerShell 7 window, run the following command to set the acces policy for the Verifiable Credentials admin user:

```
PS C:\> Set-AzKeyVaultAccessPolicy –VaultName "<yourVault>" -UserPrincipalName "<yourAdminUser>" -
PermissionsToKeys create, delete, sign
```

Replace <*yourVault*> by the name of your keyvault instance, and <*yourAdminUser*> by the account name used during configuration to create and delete keys. For example in our illustration:

```
PS C:\> Set-AzKeyVaultAccessPolicy – VaultName "VCIssuer-Vault" -UserPrincipalName
"admin@litware369inc.onmicrosoft.com" -PermissionsToKeys create, delete, sign
```

Add an access policy for the Verifiable Credentials Service

From the previous PowerShell 7 session, run the following command to set the acces policy for the Verifiable Credentials Service (bb2a64ee-5d29-4b07-a491-25806dc854d3):

```
PS C:\> Set-AzKeyVaultAccessPolicy – VaultName "<yourVault>" -UserPrincipalName "bb2a64ee-5d29-4b07-a491-
25806dc854d3" -PermissionsToKeys get,sign
```

Replace the placeholder value in brackets with your own value. For example in our illustration:

```
PS C:\> Set-AzKeyVaultAccessPolicy – VaultName "VCIssuer-Vault" -UserPrincipalName "bb2a64ee-5d29-4b07-a491-
25806dc854d3" -PermissionsToKeys get,sign
```

Add an access policy for the Verifiable Credentials Service Request

From the previous PowerShell 7 session, run the following command to set the acces policy for the Verifiable Credentials Service Request, part of AD free (`3db474b9-6a0c-4840-96ac-1fceb342124f`):

```
PS C:\> Set-AzKeyVaultAccessPolicy – VaultName "<yourVault>" -UserPrincipalName "3db474b9-6a0c-4840-96ac-
1fceb342124f" -PermissionsToKeys get,sign
```

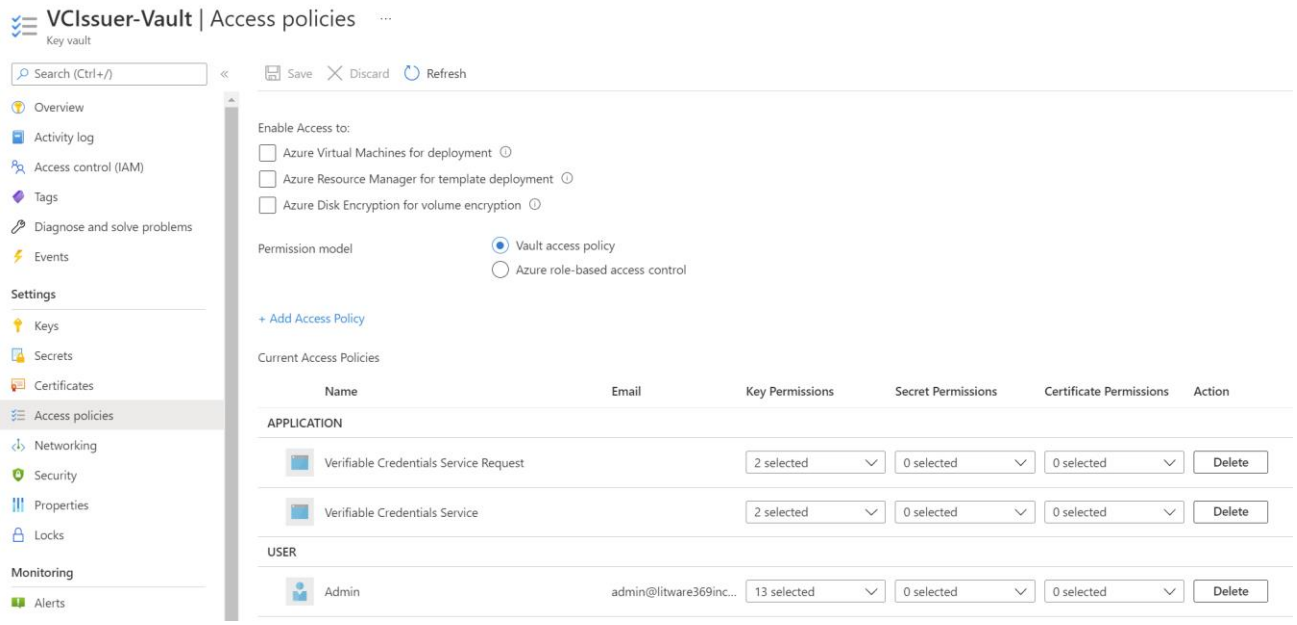
Replace the placeholder value in brackets with your own value. For example in our illustration:

```
PS C:\> Set-AzKeyVaultAccessPolicy – VaultName "VCIssuer-Vault" -UserPrincipalName "3db474b9-6a0c-4840-96ac-
1fceb342124f" -PermissionsToKeys get,sign
```

View the access policies

Proceed with the following steps:

1. Open a browser session, navigate to the Azure portal, and authenticate with your Azure administrative credentials.
2. Go to **Key vaults** and click your key vault.
3. Under **Settings**, select **Access policies**.
4. You should see the user and the Microsoft Entra Verified ID service principals your granted access to your key vault as illustrated in this screeshot.



For **Key permissions**, verify that the above permissions are selected as expected.

Register your application in Azure AD

Calling the Request Service REST API requires an OAuth 2.0 access token. The code sample application uses the Microsoft Authentication Library (MSAL), see section **Erreur ! Source du renvoi introuvable.**, and the client

credential flow to obtain such an access token. To get an access token, your application must first be registered in Azure AD, and be granted permission to the REST API.

Register a new application

From the previous PowerShell 7 window, run the following command:

```
PS C:\> New-AzADApplication -DisplayName <yourApp>
```

Replace the placeholder value in brackets with your own value. Make sure there is no app with the same name in your Azure AD tenant. For example in our illustration.

```
PS C:\> New-AzADApplication -DisplayName "Litware369-VC"
```

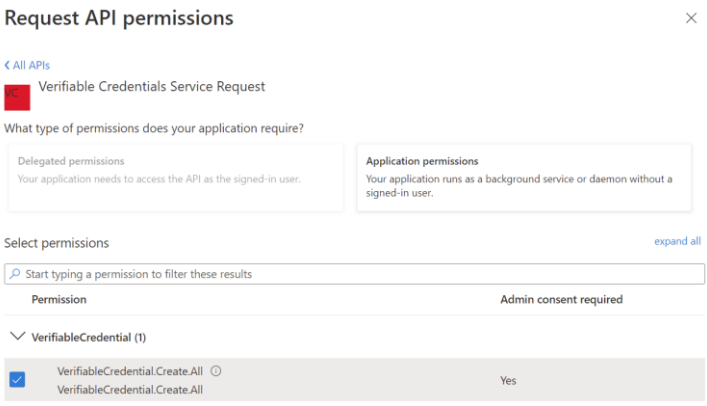
Grant permissions to get access tokens

Proceed with the following steps:

1. From the previous browser session, in the Azure portal, go to **Azure Active Directory** for the subscription you will use for your Microsoft Entra Verified ID deployment.
2. Click **App Registration**.
3. Select **All applications** and select your newly registered application. For example Litware369-VC in our illustration:
    a. Click **API permissions**, and then **Add a permission**.
    b. Select **Microsoft API**, then **Delegated Permission**, and finally **User**. Select **User.Read**
    c. Click **Add a permission**, and then select **APIs my organization uses**.
    d. Search for **Verifiable Credentials Service Request** and select it.

    Make sure you aren't selecting the Verifiable Credential Request Service! Before proceeding, confirm that the Application Client ID is 3db474b9-6a0c-4840-96ac-1fceb342124f.

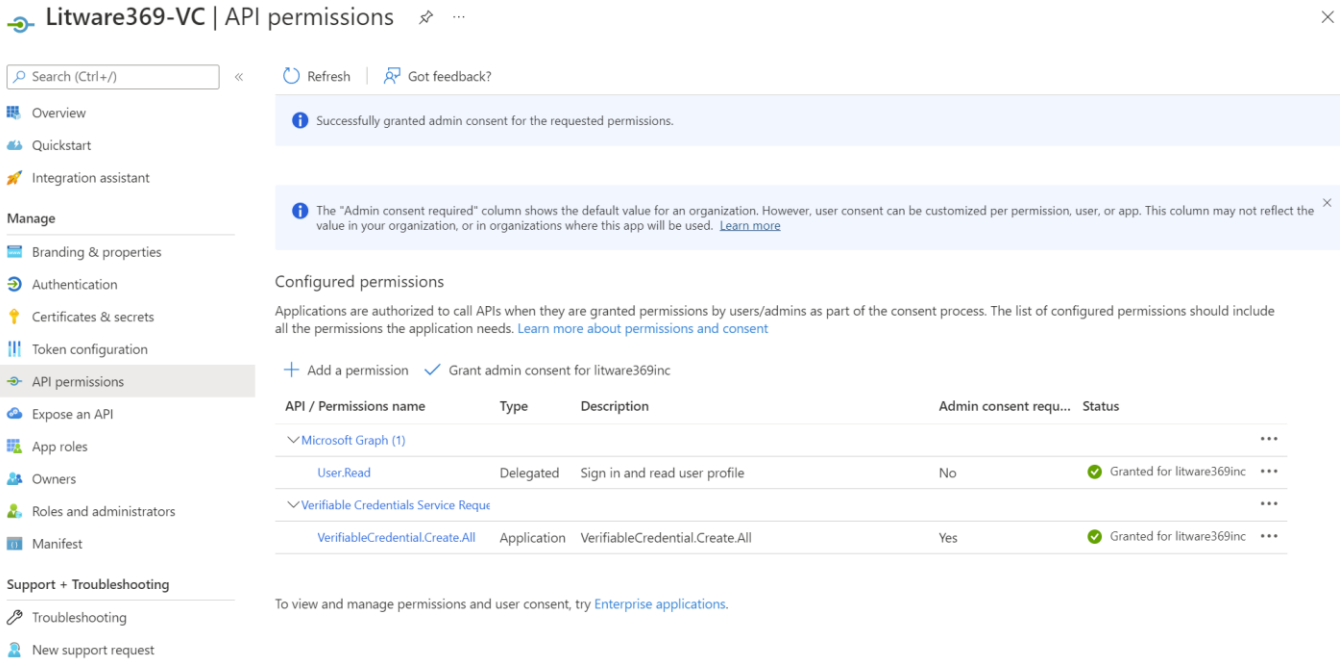
    e. Choose **Application Permission**, and expand **VerifiableCredential**.
    f. Select **VerifiableCredential.Create.All**.



    g. Click **Add Permissions**.

h. Select **Grant admin consent for <*yourTenant*>** where <*yourTenant*> is your Azure AD tenant,. Click **OK** to confirm.

After the completing all of the above steps, you should see something similar to the screenshot herafter:



For the Request API, the scope for your application or Postman is 3db474b9-6a0c-4840-96ac-1fceb342124f/.default.

Set up Microsoft Entra Verified ID

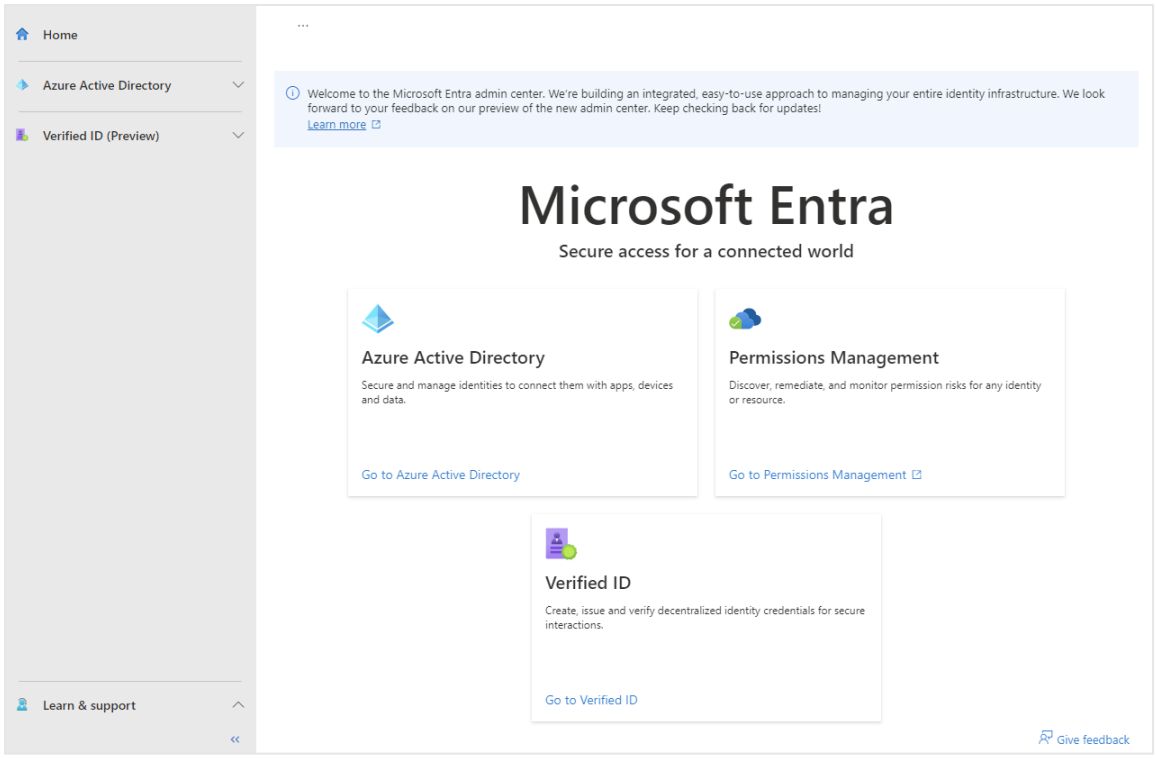
Set up your organization

To set up your organization in Microsoft Entra Verified ID, proceeed with the following steps:

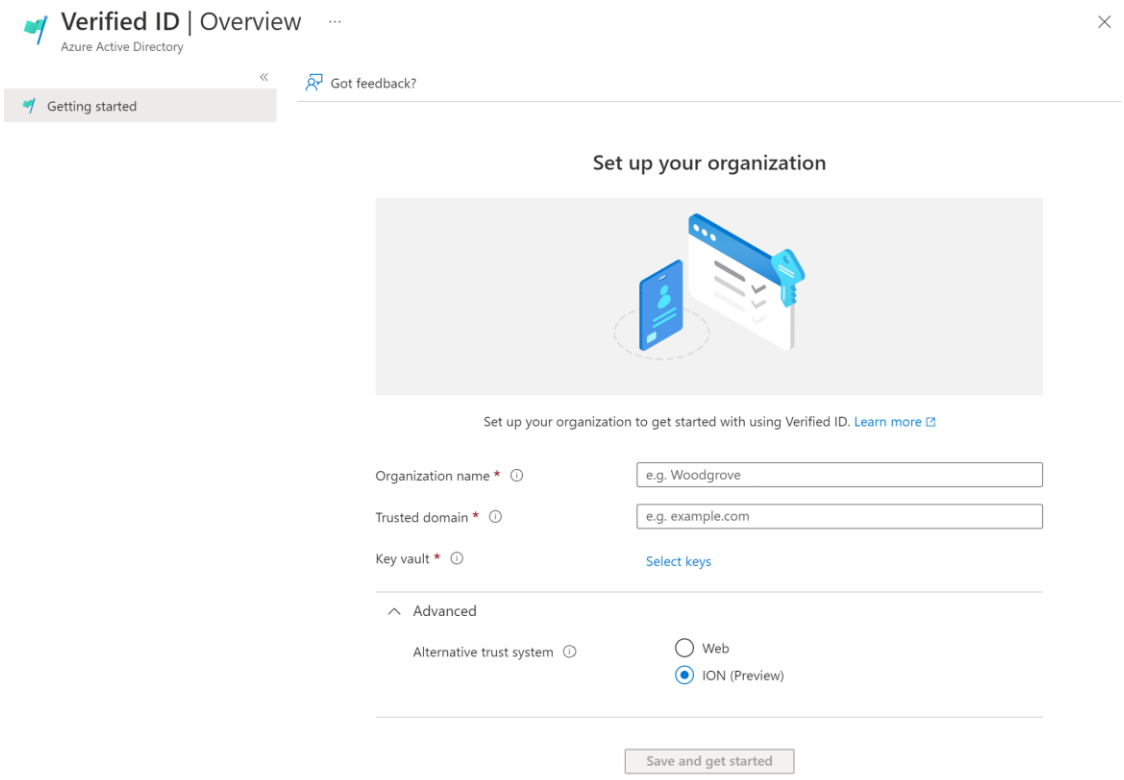
1. From the previous browser session, in the Azure portal, search for "*verified id*", and then click **Verified ID** to open the eponym blade.

   The blade can be found using a direct link or via the search box at the top of the Azure Portal – search for "*verified id*". To access this blade, your user account must be assigned to a "Global Administrator" or "Authentication Policy Administrator" Azure AD role in your Azure AD tenant.

   Please note that you can start using the **unified Microsoft Entra admin center** instead at https://entra.microsoft.com/. The admin center represents an integrated, easy-to-use approach to managing your entire identity infrastructure. Click **Verified ID**.



2. From the left pane, select **Getting started**.



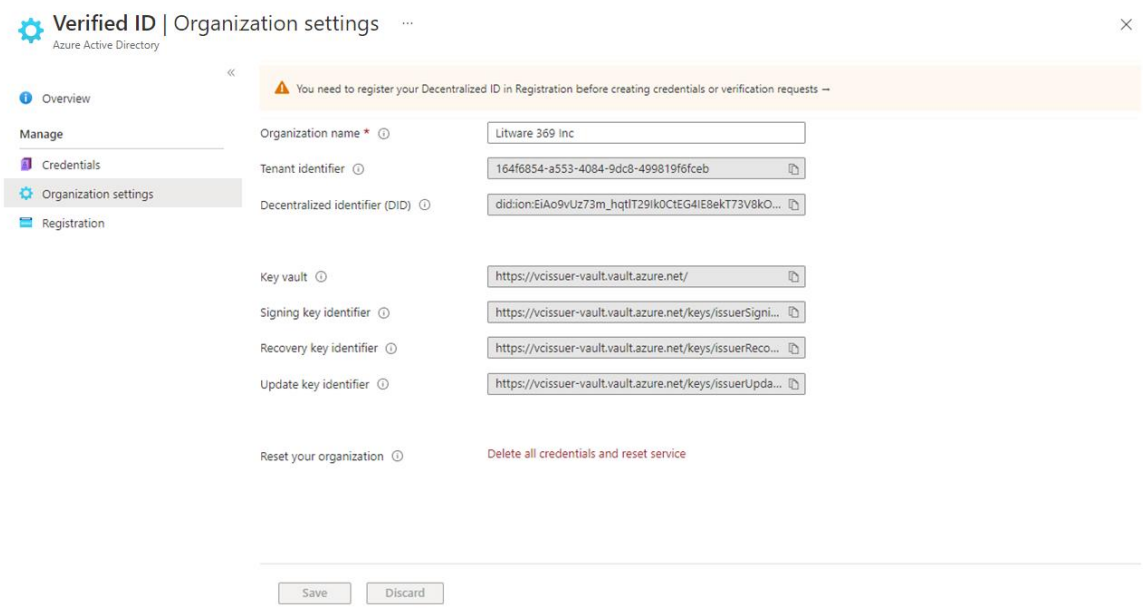
3. Set up your organization by providing the following information:
    - **Organization name**: Enter a name to reference your business within Verifiable Credentials. The users to whom you will issue VCs will not see this name. For example, Litware369Inc. in our illustration.
    - **Domain**: Enter a domain that will be added to a service endpoint in your decentralized identity (DID) document.

      The domain is what binds your DID to something tangible that the user might know about your business. Microsoft Authenticator and other digital wallets will use this information to validate that your DID is linked to your domain. If for example, Microsoft Authenticator can verify the DID, it displays a verified symbol. Conversely, if it can't verify the DID, it will inform the user that the credential was issued by an organization it couldn't validate.

      The domain can't be a redirect. Otherwise, the DID and domain can't be linked – This will be our next step in the configuration. For example, https://www.litware369.com in our illustration.
    - **Key vault**: Click **Select keys**. Enter the name of the key vault that you created earlier.
    - You may choose the trust system that you want to use for your Azure AD tenant.

      Under **Advanced**, in **Alternative trust system**, choose from either **Web** (default) or **ION**. Web means your tenant uses did:web as the did method and ION means it uses the did:ion method instead.

      The only way to change the trust system is to opt-out of verifiable credentials and redo the onboarding. See Opt out of the Microsoft Entra Verified ID.
4. Select **Save and create credentials**.
5. In the left pane, click **Organization settings**.



After you've registered your key vault, the Microsoft Entra Verified ID generates a set of keys used to provide message security. These keys are stored in the specified Key Vault. You use a key set for signing, recovering, und updating verifiable credentials as suggested by the related key identifier in the above snapshot.

With ION selected for the trust system, you should have 3 keys in your vault with the names:

1. issuerRecoveryKeyIon-<*GUID*>. For example, in our illustration:

   https://vcissuer-vault.vault.azure.net/keys/issuerSigningKeyIon-1eed9713-f62a-d384-cf74-9bbb249d24e8/24a62b35a5994947950ee71527eb4e73

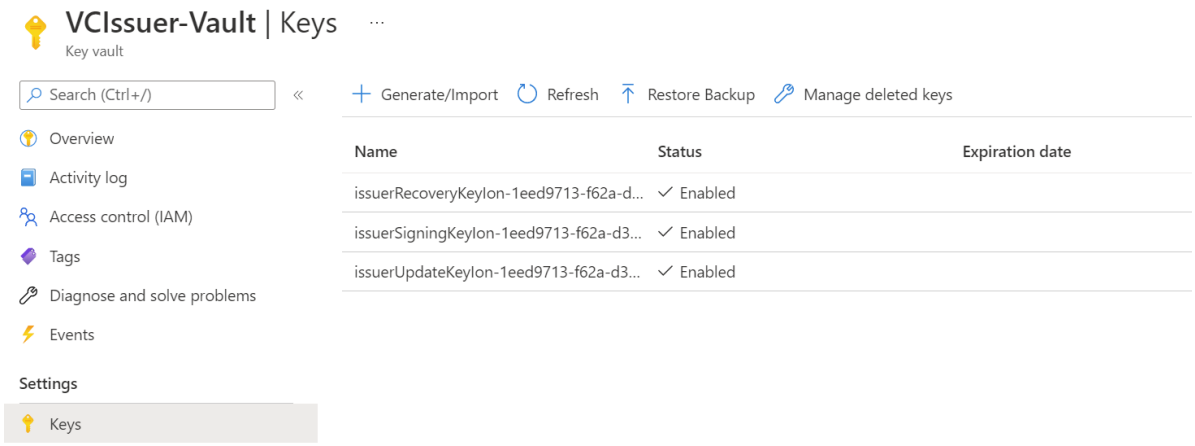
2. issuerRecoveryKeyIon-<*GUID*>. For example, in our illustration:

   https://vcissuer-vault.vault.azure.net/keys/issuerRecoveryKeyIon-1eed9713-f62a-d384-cf74-9bbb249d24e8/67a4232f021a4619b4820e125ab7e45c

3. issuerUpdateKeyIon-<*GUID*>. For example, in our illustration:

   https://vcissuer-vault.vault.azure.net/keys/issuerUpdateKeyIon-1eed9713-f62a-d384-cf74-9bbb249d24e8/0ddeef812fa54abf97fdf8fbcdf06f98

These keys appears in your keyvault. You can observe that now expiration date has been set.



As of this writing, keys created in your vault should not be modified once created. Deleting, disabling, or updating your keys will invalidate any credentials issued in the future. Do not modify your keys.

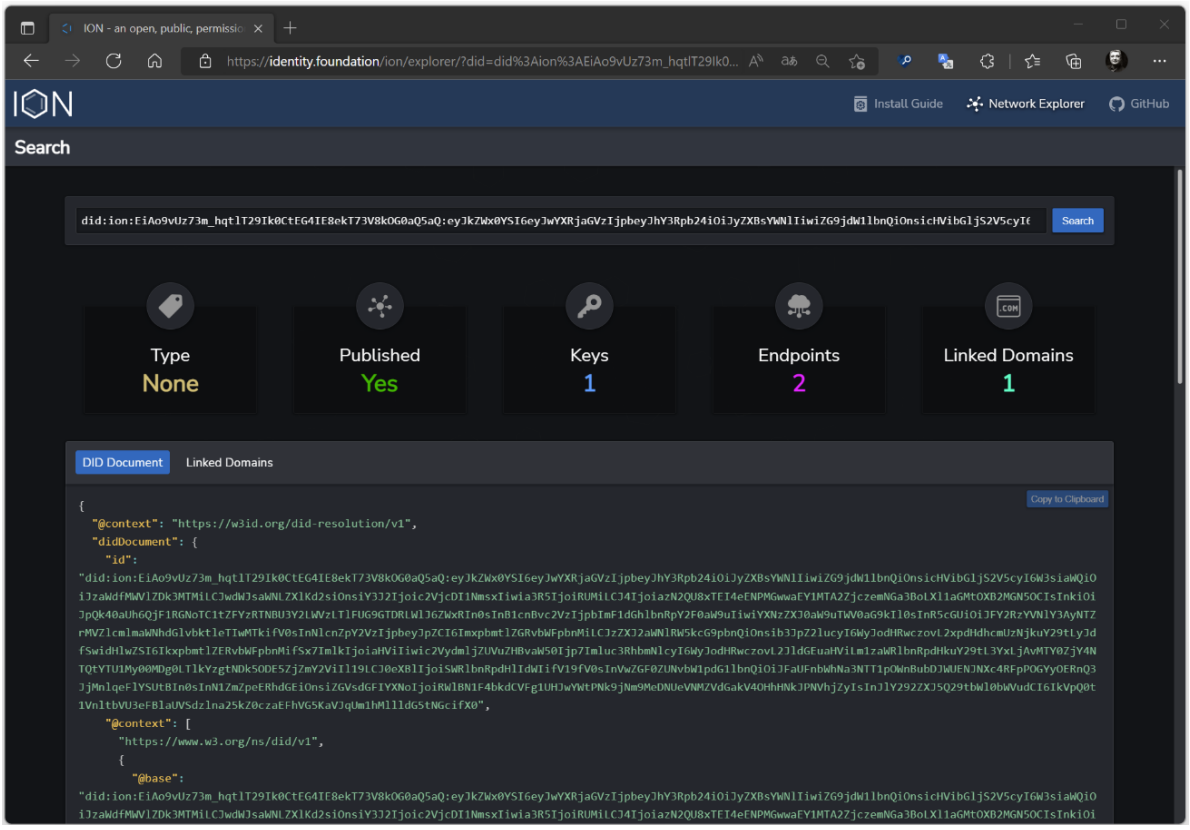
Additionaly, you can see that a DID was created for your organization. For example, in our illustration:

`did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1NmsxIiwia3R5IjoiRUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LWVzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3AyNTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6Imxpbmtlc ZGRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYyOERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNkJPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0`

It's in the form of `did:method:method-specific-identifier`. The `method` identifies the underlying distributed trust system's technology and the `method-specific-identifier` uniquely identifies the DID document on the ledger. As previously mentioned, Microsoft is currently using the Web method or the ION method (in preview) to store the DIDs, here the latter on in our illustration.

So, to see the related DID document, proceed with the following steps:

1. Open a new browser window/tab, and navigate to the ION Explorer at https://identity.foundation/ion/explorer/.
2. Paste your DID in the search box, and click **Search**. You should see the DID Document for your organization.



You can now see all the details of the public key and a service endpoint where you will have to publish a JSON document in the next section, i.e. the well-known DID configuration file.

A DID starts out as an identifier that isn't anchored to existing systems. A DID is useful because an organization or user can own it and control it. If an entity interacting with the organization, here our fictitious company Litware369 Inc., doesn't know 'who' the DID belongs to, then the DID isn't as useful.

Link your domain to your DID

See Link your Domain to your Decentralized Identifier (DID).

When a user is going through an issuance flow or presenting a VC, they should know something about the organization and its DID. Microsoft Authenticator validates a DID's relationship with the domain in the DID document and presents users with two different experiences depending on the outcome.

Before Microsoft Authenticator displays a **Verified** icon, a few things need to be true:

- The DID signing the self-issued open ID (SIOP) request must have a Service endpoint for Linked Domain.
- The root domain doesn't use a redirect and uses https.
- The domain listed in the DID Document has a resolvable well-known resource.

- The well-known resource's verifiable credential is signed with the same DID that was used to sign the SIOP that Microsoft Authenticator used to kick start the flow.

If all of the previously mentioned are true, then Microsoft Authenticator displays a verified page and includes the domain that was validated. As like in the following screenshot for our illustration

If any of the above aren't true, Microsoft Authenticator displays a full page warning to the user indicating that the domain is unverified. The user is warned that they are in the middle of a potential risky transaction and they should proceed with caution. Linking a DID to a domain solves the initial trust problem by allowing any entity to cryptographically verify the relationship between a DID and a domain, here the above DID and www.litware369.com.

When creating the link, we follow the Well-Known DID configuration (DIF) specification. The Verifiable Credentials Service links your DID and domain. The service includes the domain information that you provided in your DID, and generates the well-known configuration file:

1. The Microsoft Entra Verified ID service uses the domain information you've provide during organization setup to write a Service Endpoint within the above DID Document for your organization. All parties who interact with your DID can see the domain your DID proclaims to be associated with.

```
"service": [
  {
    "id": "#linkeddomains",
    "type": "LinkedDomains",
    "serviceEndpoint": {
      "origins": [
        "https://www.litware369.com/"
      ]
    }
  }
]
```

2. The Microsoft Entra Verified ID service generates a compliant well-known configuration resource that you can host on your domain. The configuration file includes a self-issued verifiable credential of credentialType `DomainLinkageCredential` signed with your DID that has an origin of your domain.

```
{
  "@context": "https://identity.foundation/.well-known/contexts/did-configuration-v0.0.jsonld",
  "linked_dids": [
    "<self-issued verifiable credential>"
  ]
}
```

Retrieve the well-known configuration file

To do so, proceed with the following steps:

1. From the Azure portal, go to **Verified ID**, and click **Registration**. This blade contains have three fields with the did:ion method:
    - **Domain** : contains the domain name, e.g., https://www.litware369.com/ in our illustration.
    - **Status** : indicates that your domain is published.
    - **Domain verification** : has a **Verification instructions** link for instructions to follow in order to verify your domain. Verifying a domain will help your users to trust that your organization is authentic.

2. Click the **Verification instructions** link. This opens up a new **Verify domain** blade. for domain verification.
3. Download the *did-configuration.json* well-known configuration file. Click **Download**. For example, the content is as follows in our configuration – The self-issued verifiable credential is shortened -.

```
{
  "@context": "https://identity.foundation/.well-known/contexts/did-configuration-v0.0.jsonld",
  "linked_dids": [
    "eyJhbGciOiJFUzI1NksiL[…]j99Tqr0ihKxpmNkV5y4HzJ6cPpGhAFlLQyM8oau3w"
  ]
}
```

After you have the well-known configuration file, you need to make the file available using the domain name you specified when you enabled your Microsoft Entra Verified ID service:

- Host the well-known DID configuration file at the following location:

  https://<*yourDomain*>/.well-known/did-configuration.json

  The placeholder value in brackets should be replaced with your own value. For example in our illustration:

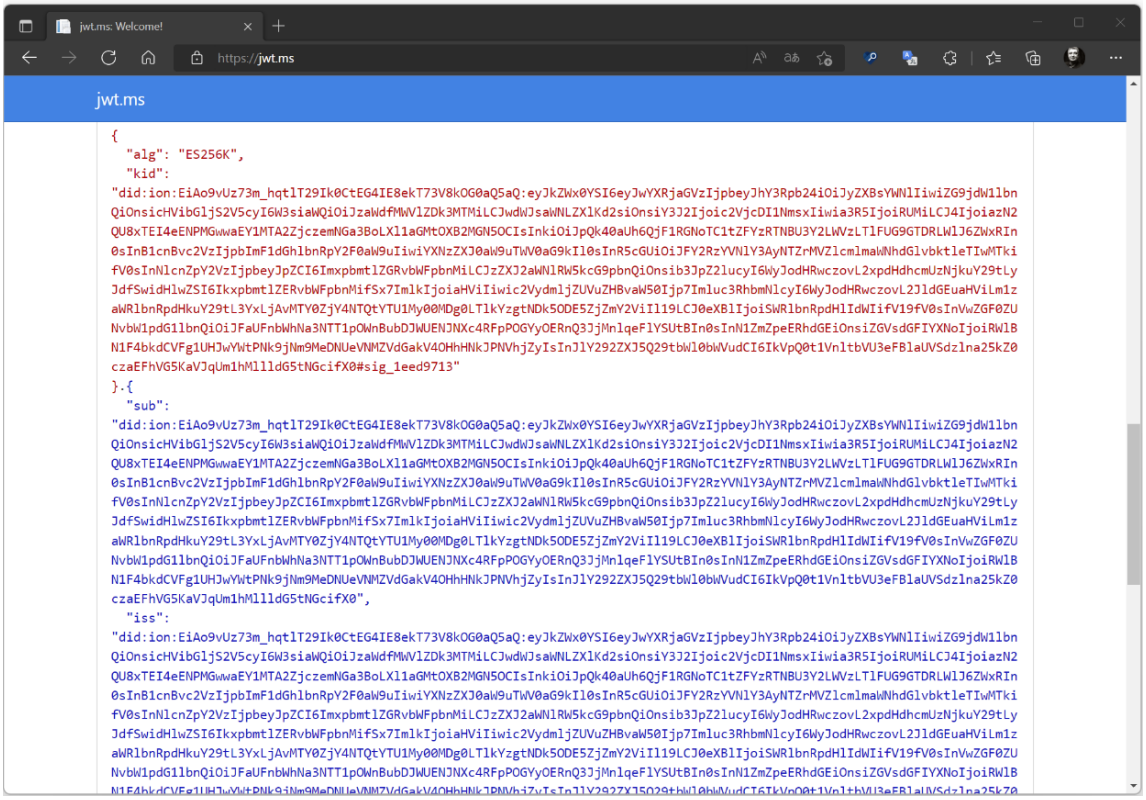
  https://www.litware369.com/.well-known/did-configuration.json

  We will use Azure App Service to host this file as well as the code sample application bits. Azure App Service indeed enables you to easily build and host web apps, mobile back ends, and RESTful APIs in the programming language of your choice without managing infrastructure.

- Don't use redirects. Microsoft Authenticator does not honor redirects, the URL specified must be the final destination URL.
- Use https to distribute the configuration file.

To see the content of the self-issued verifiable credential for your organization, proceed with the following steps:

1. Copy the value of the `linked_dids` field.
2. Open a broswer session and navigate to https://jwt.ms. The self-issued verifiable credential is formatted as a JSON Web Token (JWT) token.


jwt.ms: Welcome!
https://jwt.ms
jwt.ms
{
"alg": "ES256K",
"kid":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG9jdW1lbn
QiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1NmsxIiwia3R5IjoiRUMiLCJ4IjoiazN2
QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LWVzLTlFUG9GTDRLWlJ6ZWxRIn
0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3AyNTZrMVZlcmlmaWNhdGlvbktleTIwMTki
fV0sInNlcnZpY2VzIjpbeyJpZCI6Imxpbmtl ZGRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLy
JdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1z
aWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlUY29tLy9

```
VzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3Ay
NTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6ImxpbmtlZGRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2
lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50
Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5Zj
ZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYy
OERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNk
JPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0",
 "nbf": 1651650297,
 "exp": 2440568697,
 "vc": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://identity.foundation/.well-known/contexts/did-configuration-v0.0.jsonld"
    ],
    "issuer":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG
9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1NmsxIiwia3R5Ijoi
RUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LW
VzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3Ay
NTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6ImxpbmtlZGRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2
lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50
Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5Zj
ZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYy
OERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNk
JPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0",
    "issuanceDate": "2022-09-04T07:44:57.045Z",
    "expirationDate": "2047-09-04T07:44:57.045Z",
    "type": [
       "VerifiableCredential",
       "DomainLinkageCredential"
    ],
    "credentialSubject": {
       "id":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG
9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1NmsxIiwia3R5Ijoi
RUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LW
VzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3Ay
NTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6ImxpbmtlZGRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2
lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50
Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5Zj
ZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYy
OERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNk
JPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0",
      "origin": "https://www.litware369.com/"
    }
  }
}.[Signature]
```


Update the code sample application

Proceed with the following steps:

1. Copy the manifest file into the *CredentialFiles* folder of the code sample you previously cloned in your local environment. See section **Clone the repo using Git**.
2. At the root of the directory where you previoulsy cloned the code sample, e.g., *C:\>vc-waltkthrough\part-1* in our in illustration, open the *app.js* file with the code editor of your choice, e.g., Visual Studio Code, and add the following code snippet before starting the server.

```
app.use('/.well-known', express.static('.well-known'), serveIndex('.well-known'));

app.get('/.well-known/did-configuration.json', (req, res) => {
    const wellknown = require('./CredentialFiles/did-configuration.json');
    res.send(wellknown);
});
```

At this stage, you have now successfully configured Microsoft Entra Verified ID using cryptographic keys in your Azure Key Vault.

Once that you have successfully completed the verification steps, you are ready to continue to the next tutorial.

Note that with the did:web method in lieu of the chosen did:ion method, you will need to also register the Website ID in addition to the above Domain verification. This requires to expose yet another json configuration file: https://www.litware369.com/.well-known/did.json. See How to register your website ID.

In the next section, we'll begin to define the details of the Verified Employee credential you will issue.

Issue a Verified Employee credential

At this stage, to effectively issue your first Verified Employee credential, you will more specifically need to :

- Create the Verified Employee credential in Azure.
- Gather credentials and environment details to set up your code sample application based on the code you previoulsy cloned onto your local computer.
- Update the code sample application accorfingly.
- Run the code sample application locally vs. on the Internet.
- Issue and verify your first Verified Employee credential.

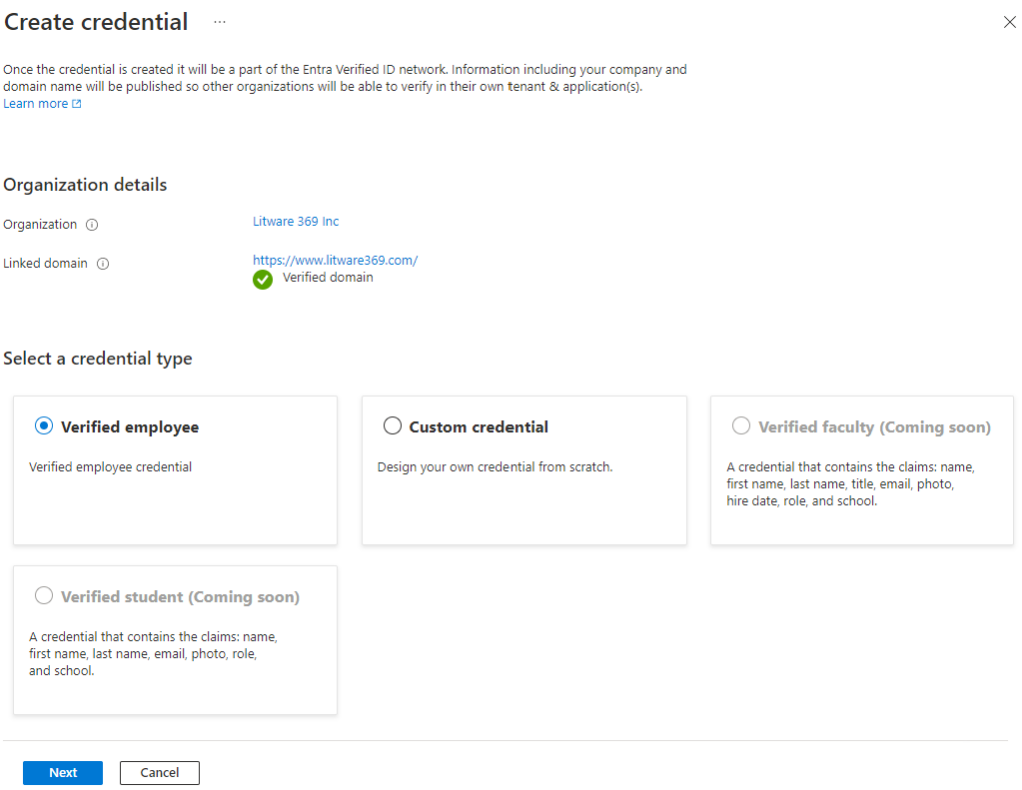
Create a Verified Employee credential

See Tutorial - Issue a Verifiable Credential for directory based claims.

Let's now create a Verified employe credential by using the new quick start. The Verified Employee is a verifiable credential of type verifiedEmployee that is based on a pre-defined set of claims from your tenant's Azure Active Directory. As such, a Verified Employee credential is a managed credentials.

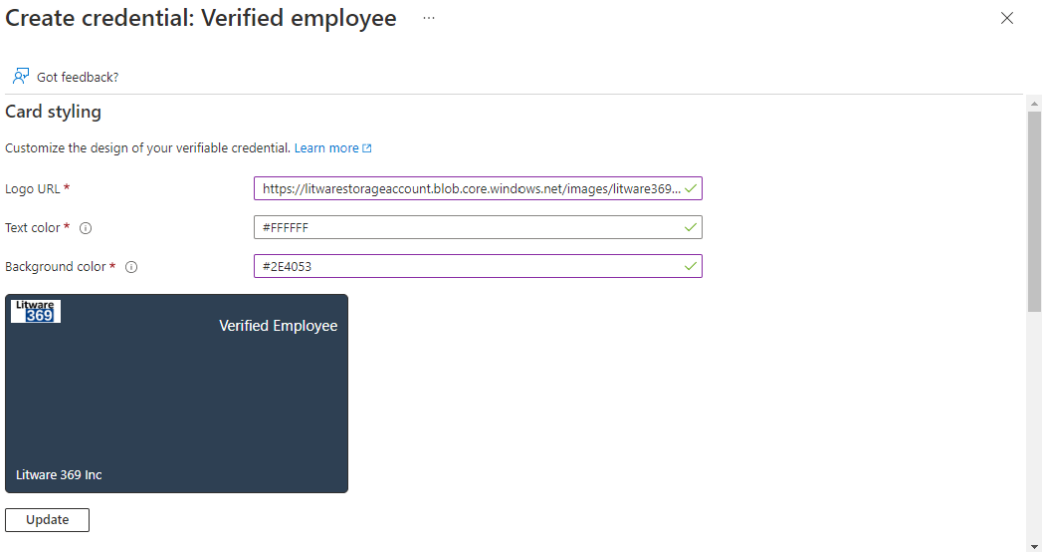
Proceed with the following steps:

1. Using the Azure portal, search for "*verifiable id*", and select **Verified ID**. As already mentioned, you can use instead the newly introduced the unified Microsoft Entra admin center. In this case, click **Verified ID**.
2. Click **Credentials**, and click **+ Add a credential**. A **Create credential** blade opens up.

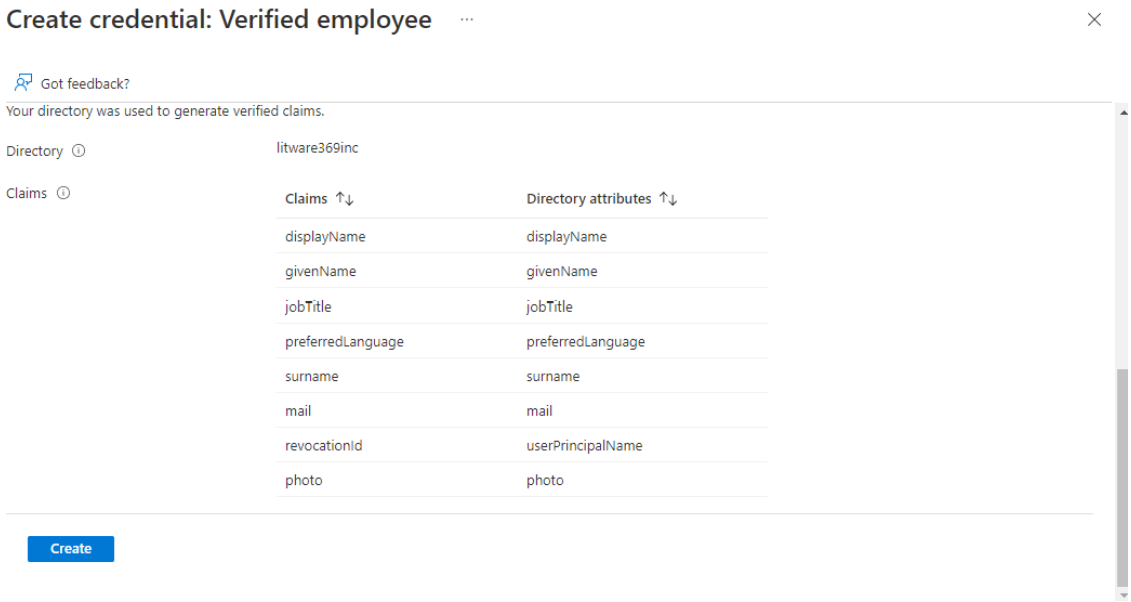


You get the option to launch two available quickstarts, with more to come. Select **Verified employee** and select **Next**.

3. In the next screen, under **Card styling**, specify the some of the **Display** definitions of the VC, namely the **Logo URL**, the **Text color** and the **Background color** for our fictious company, and click **Update**.



4. Since the VC is a managed credential with directory based claims, rules definitions are predefined. You don't need to enter rule definition details. The credential type will be **VerifiedEmployee** and the claims from the user's profile are pre-set.



All of the claims in the Verified Employee credential come from attributes in the user's profile in your (issuing) Azure AD tenant. More specifically:

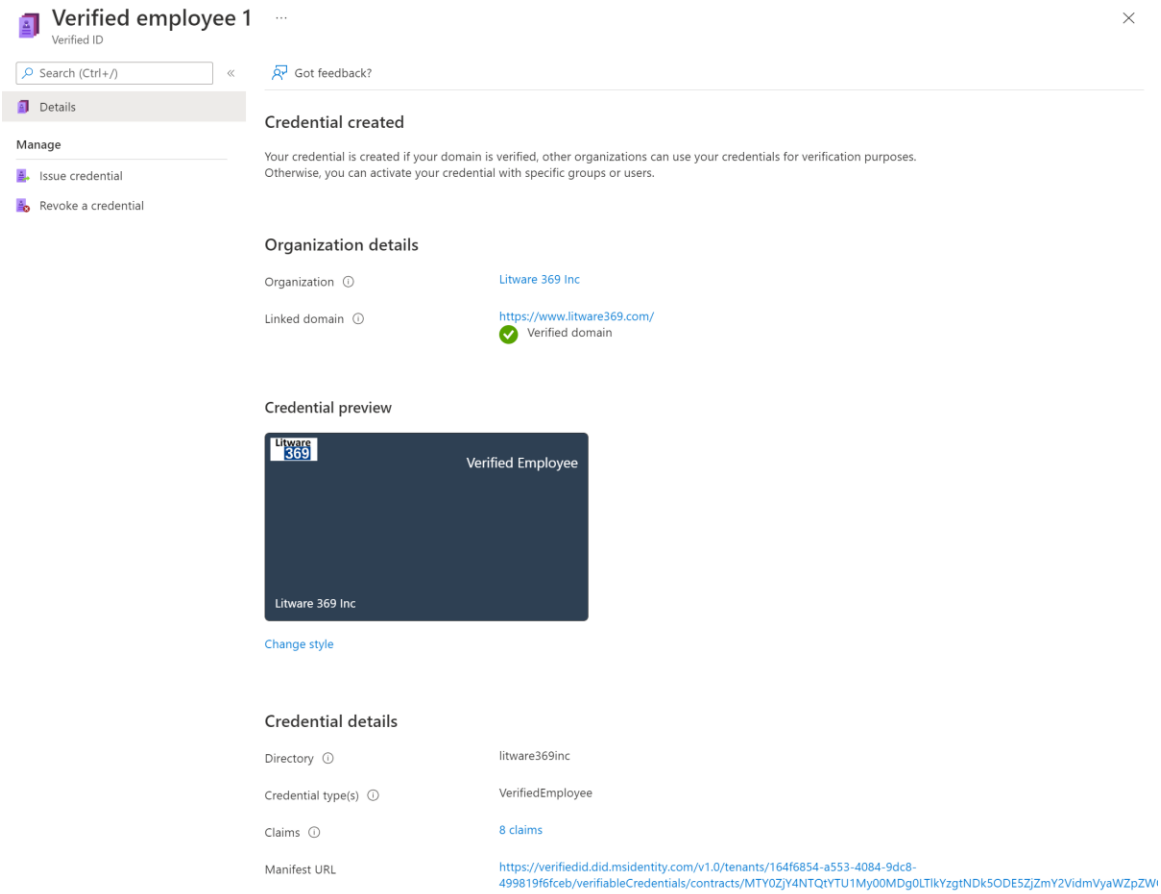
a. All claims, except photo, come from the Microsoft Graph Query https://graph.microsoft.com/v1.0/me.
b. The photo claim comes from the value returned from the Microsoft Graph Query https://graph.microsoft.com/v1.0/me/photo/$value.

Claims	Directory attribute	Value
`displayName`	displayName	The displayName of the user
`givenName`	givenName	First name of the user
`surname`	surname	Last name of the user
`jobTitle`	jobTitle	The user's job title. This attribute doesn't have a value by default in the user's profile. If the user's profile has no value specified, there's no jobTitle claim in the issued VC.
`preferredLanguage`	preferredLanguage	Should follow ISO 639-1 and contain a value like en-us. There's no default value specified. If there's no value, no claim is included in the issued VC.
`mail`	mail	The user's email address. The mail value isn't the same as the UPN. It's also an attribute that doesn't have a value by default.
`revocationId`	userPrincipalName	The UPN of the user is added as a claim named revocationId and gets indexed.
`photo`	photo	The uploaded photo for the user. The image type (JPEG, PNG, etc.), depends on the uploaded image type. When presenting the photo claim to a verifier, the photo claim is in the UrlEncode(Base64Encode(photo)) format. To use the photo, the verifier application has to Base64Decode(UrlDecode(photo)).

See user resource type - Microsoft Graph v1.0 for full Azure AD user profile properties reference.

Please note that if attribute values change in the user's Azure AD profile, the VC isn't automatically reissued. You must reissue it manually. Issuance would be the same as the issuance process when working with the code sample application.

5. Select **Create** to create the credential. A **Verified employee 1** credential of type VerifiedEmployee is now created.



Now that you have a new Verified Employee credential in place, you're going to gather some information about this specific credential and your environment.

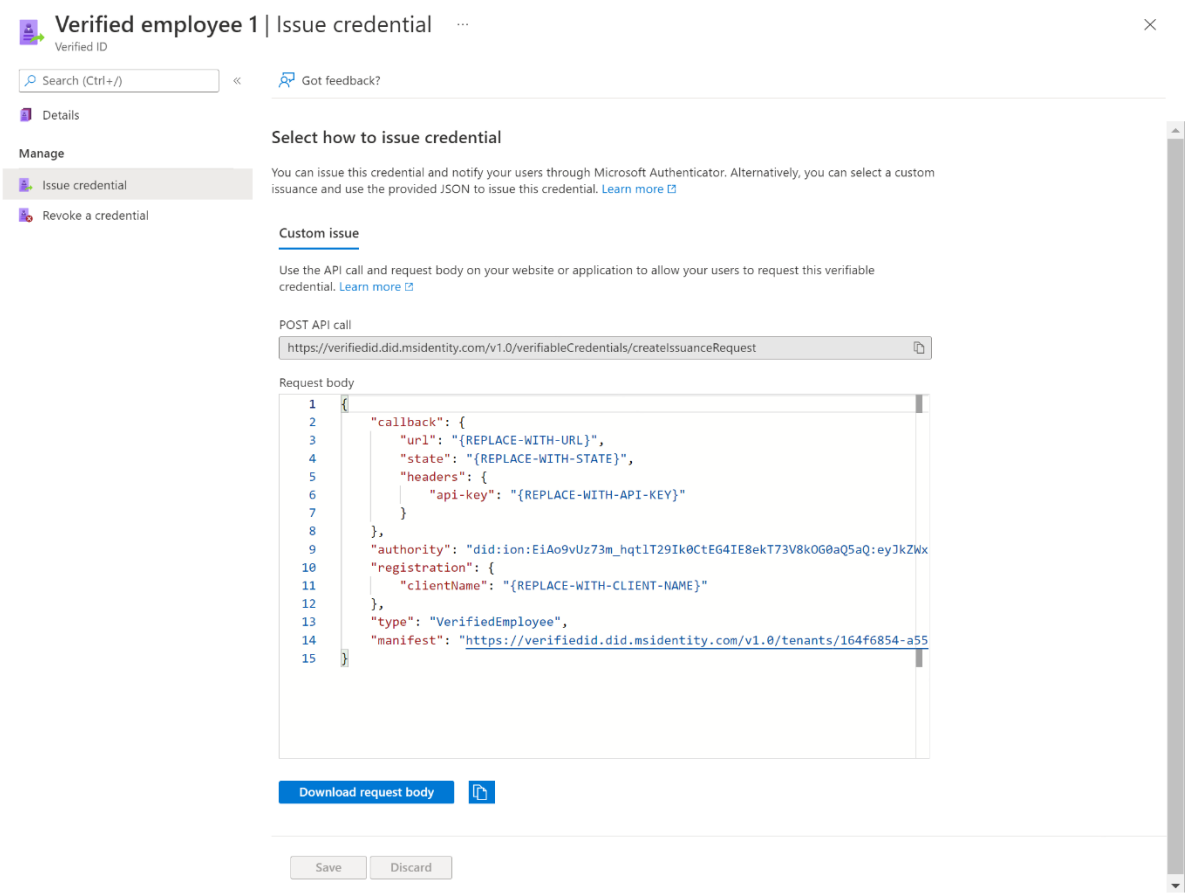
Gather the Verified Employee credential details

Verifiable Credentials for Directory-based claims can be issued and verified just like any other credentials you create. Please refer to the second part of this guide to see how to both issue and verify Custom credentials where you design your own credentials from scratch.

All you need is your issuer DID for your Azure AD tenant, the credential type, and the manifest url to your credential. The easiest way to find these values for a managed credential is to view the credential in the portal.

Proceed with the following steps:

1. Still from the above **Verified employee 1** blade, under **Manage**, select **Issue credential** and switch to **Custom issue**.



These steps bring up a textbox with a skeleton JSON payload for the Request Service REST API:

```
{
    "callback": {
        "url": "{REPLACE-WITH-URL}",
        "state": "{REPLACE-WITH-STATE}",
        "headers": {
            "api-key": "{REPLACE-WITH-API-KEY}"
        }
    },
    "authority":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG
9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1NmsxIiwia3R5Ijoi
RUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LW
VzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3Ay
NTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6Imxpbmtl ZGRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2
lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50
Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5Zj
ZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYy
OERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNk
JPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMlldG5tNGcifX0",
    "registration": {
        "clientName": "{REPLACE-WITH-CLIENT-NAME}"
    },
    "type": "VerifiedEmployee",
    "manifest": "https://verifiedid.did.msidentity.com/v1.0/tenants/164f6854-a553-4084-9dc8-
499819f6fceb/verifiableCredentials/contracts/MTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2VidmVyaWZpZWQgZW1wbG95Z
WUgMQ/manifest"
}
```

2. In this payload, take a note of the following values that you will need to reflect in your code sample application's configuration files :

    a. `authority`: this value corresponds to the issuer's DID (`did:ion..`). You will need to specify this value in the `IssuerAuthority` and `VerifierAuthority` properties of the *config.json* file of the NodeJS code sample application.
    b. `type`: the credential type is always `VerifiedEmployee` when looking at a Verified Employee credential.
    c. `manifest`: the complete URL to the DID manifest used for both issuance and presentation is as follows:

    https://verifiedid.did.msidentity.com/v1.0/tenants/164f6854-a553-4084-9dc8-499819f6fceb/verifiableCredentials/contracts/MTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2VidmVyaWZpZWQgZW1wbG95ZWUgMQ/manifest

    Starting from July '22, the hostname of the Request Service API becomes verifiedid.did.msidentity.com, with no longer distinction for an EU tenant, see section **Check the region of your Azure AD tenant**. Please note that the hostnames beta.did.msidentity and the beta.eu.did.msidentity will continue to work.

    If you copy the (JWT) `token` value returned by the above URL, and decode it with jwt.ms, the content of the DID manifest appears as follows:


```
{
  "id": "MTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2VidmVyaWZpZWQgZW1wbG95ZWUgMQ",
  "display": {
    "locale": "en-US",
    "contract": "https://beta.eu.did.msidentity.com/v1.0/tenants/164f6854-a553-4084-9dc8-
499819f6fceb/verifiableCredentials/contracts/MTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2VidmVyaWZpZWQgZW1wbG95Z
WUgMQ/manifest",
    "card": {
      "title": "Verified Employee",
      "issuedBy": "Litware 369 Inc",
      "backgroundColor": "#2E4053",
      "textColor": "#FFFFFF",
      "logo": {
        "uri": "https://i.postimg.cc/XYDSqxpL/Litware-Logo.png",
        "description": "Default verified employee logo"
      },
      "description": "This verifiable credential is issued to all members of the Litware 369 Inc org."
    },
    "consent": {
      "title": "Do you want to accept the verified employee credential from Litware 369 Inc.",
      "instructions": "Please log in with your Litware 369 Inc account to receive this credential."
    },
    "claims": {
      "vc.credentialSubject.givenName": {
        "type": "String",
        "label": "Name"
      },
      "vc.credentialSubject.surname": {
        "type": "String",
        "label": "Surname"
      },
      "vc.credentialSubject.mail": {
        "type": "String",
        "label": "Email"
      },
      "vc.credentialSubject.jobTitle": {
        "type": "String",
        "label": "Job title"
      },
```

```
      "vc.credentialSubject.photo": {
        "type": "image/jpg;base64url",
        "label": "User picture"
      },
      "vc.credentialSubject.displayName": {
        "type": "String",
        "label": "Display name"
      },
      "vc.credentialSubject.preferredLanguage": {
        "type": "String",
        "label": "Preferred language"
      },
      "vc.credentialSubject.revocationId": {
        "type": "String",
        "label": "Revocation id"
      }
    },
    "id": "display"
  },
  "input": {
    "credentialIssuer": "https://beta.eu.did.msidentity.com/v1.0/tenants/164f6854-a553-4084-9dc8-
499819f6fceb/verifiableCredentials/issue",
    "issuer":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG
9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1NmsxIiwia3R5Ijoi
RUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LW
VzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3Ay
NTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6ImxpbmtlZGRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2
lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50
Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5Zj
ZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYy
OERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNk
JPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0",
    "attestations": {
      "accessTokens": [
        {
          "id": "https://login.microsoftonline.com/164f6854-a553-4084-9dc8-499819f6fceb/v2.0",
          "encrypted": false,
          "claims": [],
          "required": true,
          "configuration": "https://login.microsoftonline.com/164f6854-a553-4084-9dc8-499819f6fceb/v2.0",
          "resourceId": "bb2a64ee-5d29-4b07-a491-25806dc854d3",
          "oboScope": "User.Read.All"
        }
      ]
    },
    "id": "input"
  },
  "iss":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG
9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1NmsxIiwia3R5Ijoi
RUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LW
VzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3Ay
NTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6ImxpbmtlZGRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2
lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50
Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5Zj
ZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYy
OERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNk
JPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0",
  "iat": 1659705176
}
```


You will also later need to specify the above URL in the `CredentialManifest` property of the *config.json* of the NodeJS code sample application.

3. Eventually, you also have specified the URL for the POST API call:

   https://verifiedid.did.msidentity.com/v1.0/verifiableCredentials/createIssuanceRequest

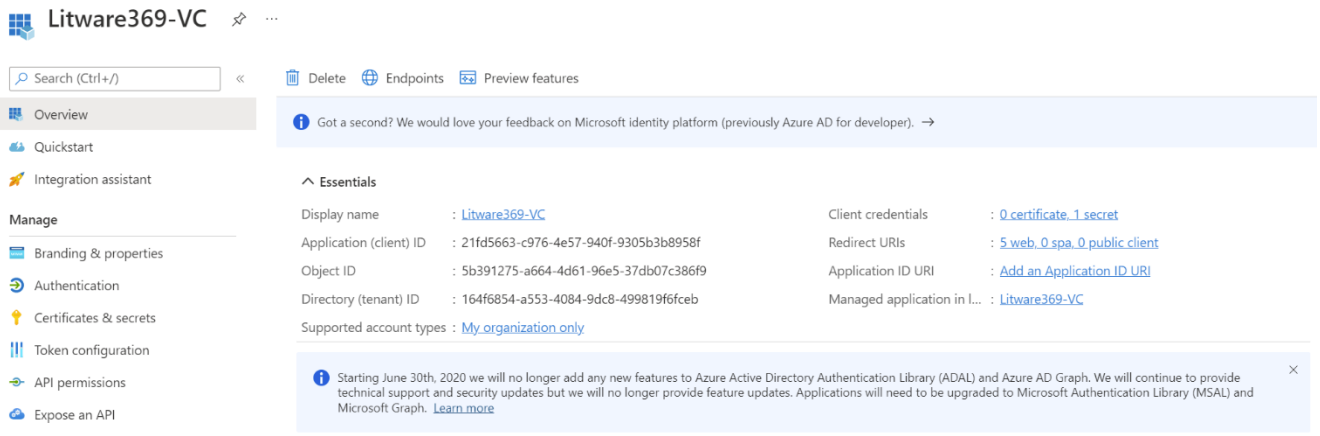
Gather the environment details

Now let's gather some additionnal information about your environment.

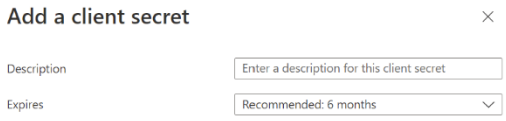
You first need to create a client secret for the registered application that you previoulsy created, see section **Register your application in Azure AD**. The code sample application (see below section) uses the client secret to prove its identity when it requests tokens.

Proceed with the following steps:

1. From the Azure portal, go to **Azure Active Directory**.
   a. In the left pane, click **App registrations**.
   b. Select the application you previoulsy registred. For example, Litware369-VC in our illustration. An eponym blabe pops up.



   c. On the application overview page, copy and take a note of the **Application (client) ID**.
   d. **Under Manage**, click **Certificates & secrets**. An eponym **Certificates & secrets** page opens up.
   e. **Select Client secrets** and click **+ New client secret**. A new **Add a client secret** blade pops up.



   i. **Description**: type a description. For example, "*Litware369 VC secret*" in our illustration.
   ii. **Expires**: Select a duration.
   iii. Click **Add**.

When you click the Add button, the key value will be displayed. Copy and save the value in a safe location. You'll need this key later to configure the code sample application. **This key value will not be displayed again, nor retrievable by any other means, so record it as soon as it is visible from the Azure portal.**

You will need this key value not only in this part but also for the second and third parts of this guide.

Configure the code sample application to issue and verify your VerifiedEmployee credential

It's time to reflect your own environment configuration in the provided code sample application. This code sample application demonstrates how to use Microsoft Entra Verified ID to issue here and later consume verifiable credentials.

The project is thus divided in 2 parts, i) one for issuing a verifiable crdential and ii) yet another one for verifying a verifiable credential.

Issuance	Description
Public\issuer.html	The basic webpage containing the javascript to call the APIs for issuance.
issuer.js	This is the file which implements express routes which contains the API called from the webpage. It calls the Request Service REST API after getting an access token through the Microsoft Authentication Library (MSAL).
issuance_request_config.json	The sample payload send to the server to start issuing a VC.
Verification	**Description**
Public\verifier.html	The website acting as the verifier of the verifiable credential.
verifier.js	This is the file which implements express routes which contains the API called from the webpage. It calls the Request Service REST API after getting an access token through MSAL and helps verifying the presented verifiable credential.
presentation_request_config.json	The sample payload send to the server to start issuing a vc.

With that, proceed with the following steps:

1. Launch a code editor of your choice. For example Visual Studio Code in our illustration.
2. Click **File**, and select **Open Folder**. Select the directory *part-1* under the folder where you previously cloned the code sample repo, e.g., *C:\>entra-verifiedid-tour\part-1* in our illustration. Click **Select the directory**.
3. At the root of the directory, open the *config.json* file.

```
{
    "azTenantId": "<YOUR-AAD-TENANDID>",
    "azClientId": "<YOUR-AAD-CLIENTID-FOR-API-ACCESS>",
    "azClientSecret": "<YOUR-AAD-CLIENTSECRET-FOR-API-ACCESS>",
    "azCertThumbprint":  "<YOUR-AAD-CERTTHUMBPRINT>",
    "azCertificatePrivateKeyLocation":  "<YOUR-AAD-CERTLOCATION>",
    "CredentialManifest": "<YOUR-MANIFEST-URL>",
    "IssuerAuthority": "<YOUR-did:ion>",
    "VerifierAuthority": "<VCs-did:ion>"
}
```

4. Update the following properties with the informations as the following:
    a. `azTenantId`: specify your tenant ID as per section Take a note of your Tenant ID above.
    b. `azClientId`: specify your application (client) ID as per section Gather the environment details above. This corresponds to the application you priorly registered that has the VC `permission` `VerifiableCredential.Create.All` and that has access to your VC Azure KeyVault.

c. `azClientSecret`: enter the related client secret you took a note earlier as per **section Gather the environment details** above.
d. `azCertificateName`, `azCertThumbprint`, and `azCertificatePrivateKeyLocation`: empty the value string.
e. `CredentialManifest`: specify the complete URL to the DID manifest as per section **Gather the Verified Employee credential details** above. It is used to set the attribute manifest and it is used for both issuance and presentation.
f. `IssuerAuthority`: specify the DID for your Microsoft Entra Verified ID service as per section **Gather the Verified Employee credential details** above.
g. `VerifierAuthority`: specify again the DID for your Microsoft Entra Verified ID service.

You should have a .json file similar to the following one:


```
{
  "azTenantId": "164f6854-a553-4084-9dc8-499819f6fceb",
  "azClientId": "21fd5663-c976-4e57-940f-9305b3b8958f",
  "azClientSecret": "vGa7Q~crZdB8FVk9q6Rn1GxNSTvJuLd4OlR4F",
  "azCertificateName": "",
  "azCertThumbprint": "",
  "azCertificatePrivateKeyLocation": "",
  "CredentialManifest": "https://verifiedid.did.msidentity.com/v1.0/tenants/164f6854-a553-4084-9dc8-
499819f6fceb/verifiableCredentials/contracts/MTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2VidmVyaWZpZWQgZW1wbG95Z
WUgMQ/manifest",
  "IssuerAuthority":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG
9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1NmsxIiwia3R5Ijoi
RUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LW
VzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3Ay
NTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6Imxpbmtlc GRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2
lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50
Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5Zj
ZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYy
OERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNk
JPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0",
  "VerifierAuthority":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG
9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1NmsxIiwia3R5Ijoi
RUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LW
VzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3Ay
NTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6Imxpbmtlc GRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2
lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50
Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5Zj
ZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYy
OERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNk
JPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0"
}
```


5. Save the *config.json* file by pressing CTRL+S.

The Request Service REST API is called with special payloads for both issuing and verifying verifiable credentials. The sample payload file is modified by the sample code by copying the correct values defined in the *config.json* file.

If you want to modify the payload *issuance_request_config.json* file resp. *presentation_request_config.json* file yourself, make sure you comment out the code overwriting the values in the *issuer.js* file resp. the *verifier.js* file. As illustrated hereafter, the code indeed overwrites the `Authority`, `Manifest` and `trustedIssuers` values, and the callback URI is also modified in code to match your hostname.

- *issuer.js* file:

```
[…]

var requestConfigFile = process.argv.slice(2)[1];
if ( !requestConfigFile ) {
  requestConfigFile = process.env.ISSUANCEFILE || './issuance_request_config.json';
}
var issuanceConfig = require( requestConfigFile );
issuanceConfig.registration.clientName = "Verified Employee Issuer Sample";
// get the manifest from config.json, this is the URL to the credential created in the azure portal.
// the display and rules file to create the credential can be found in the credentialfiles directory
// make sure the credentialtype in the issuance payload ma
issuanceConfig.authority = mainApp.config["IssuerAuthority"]
issuanceConfig.issuance.manifest = mainApp.config["CredentialManifest"]

[…]

// modify the callback method to make it easier to debug
// with tools like ngrok since the URI changes all the time
// this way you don't need to modify the callback URL in the payload every time
// ngrok changes the URI
issuanceConfig.callback.url = `https://${req.hostname}/api/issuer/issuance-request-callback`;

[…]
```

- *verifier.js* file:

```
[…]

var requestConfigFile = process.argv.slice(2)[1];
if ( !requestConfigFile ) {
  requestConfigFile = process.env.PRESENTATIONFILE || './presentation_request_config.json';
}
var presentationConfig = require( requestConfigFile );
presentationConfig.registration.clientName = "Verified Employee Verifier Sample";
presentationConfig.authority = mainApp.config["VerifierAuthority"]
// copy the issuerDID from the settings and fill in the acceptedIssuers part of the payload
// this means only that issuer should be trusted for the requested credentialtype
// this value is an array in the payload, you can trust multiple issuers for the same credentialtype
// very common to accept the test VCs and the Production VCs coming from different verifiable credential services
presentationConfig.presentation.requestedCredentials[0].acceptedIssuers[0] = mainApp.config["IssuerAuthority"]

[…]

// modify the callback method to make it easier to debug
// with tools like ngrok since the URI changes all the time
// this way you don't need to modify the callback URL in the payload every time
// ngrok changes the URI
presentationConfig.callback.url = `https://${req.hostname}/api/verifier/presentation-request-callback`;

[…]
```

6. Eventually, to be able to issue verifiable credentials, you need to make some changes in the *app.js* file as you don't need to specify any certificate in the configuration file. This file is also located at the root of the directory.

    a. Comment the following code snippet in the Microsoft Authentication Library (MSAL) configuration by pressing CTRL+K+C:

```
if ( config.azCertificateName !== '') {
  const privateKeyData = fs.readFileSync(config.azCertificatePrivateKeyLocation, 'utf8');
  console.log(config.azCertThumbprint);
  const privateKeyObject = crypto.createPrivateKey({ key: privateKeyData, format: 'pem',
    passphrase: config.azCertificateName.replace("CN=", "") // the passphrase is the appShortName (see
Configure.ps1)
  });
  msalConfig.auth = {
    clientId: config.azClientId,
    authority: `https://login.microsoftonline.com/${config.azTenantId}`,
    clientCertificate: {
      thumbprint: config.azCertThumbprint,
      privateKey: privateKeyObject.export({ format: 'pem', type: 'pkcs8' })
    }
  };
}
```

b. Save the *app.js* file by pressing CTRL+S.

All the required configuration files are now updated.

At this stage, your code sample application is ready to build, issue, and verifiy a Verified Employee credential.

Run the code sample application locally

Now you are ready to issue and then verify your first Verified Employee credential card by running the code sample application.

Run the application

Perform the following steps:

1. From Visual Studio Code, open a new terminal window. In the menu, click **Terminal**, and the select **New terminal**.
2. From the terminal window in Visual Studio Code, go to the folder where you clone the source code of the code sample application. For example, *c:\entra-verifiedid-tour \part-1* in our illustration.
3. Run the following commands to install Express:

```
PS C:\ > npm install
```

The above command will install all dependencies that the code sample application needs - it can takes few minutes -.

4. Launch the code sample application with the following command:

```
PS C:\> .\run.cmd
```

The *run.cmd* file is a template for passing the correct variables and run your NodeJS code sample application. You can also run it manually via this command:

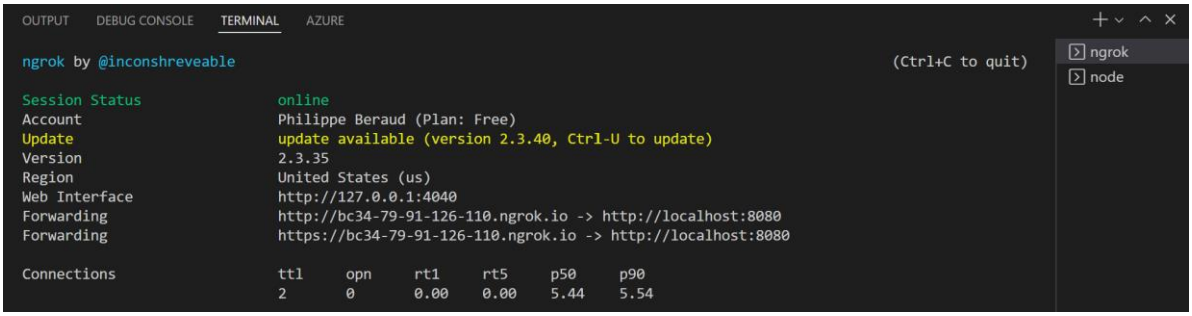
```
PS C:\> node app.js .\config.json .\issuance_request_config.json .\presentation_request_config.json
```

5. Open a second terminal windows in Visual Studio Code, and start a tunnel.

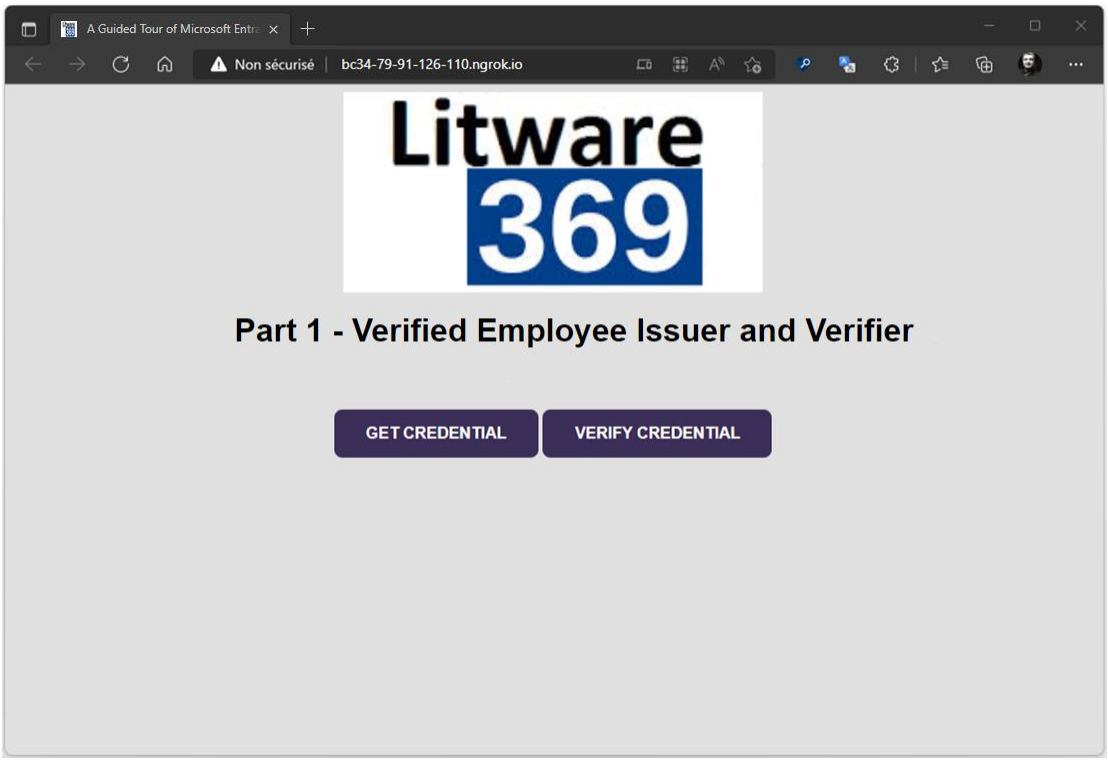
a. Run the following command:

```
PS C:\> ngrok http 8080
```

It's callback endpoint must be publically reachable, and for that reason, you use ngrok as a reverse proxy to reach your application. More specifically, this command runs ngrok to set up a URL on 8080 where the code sample application is listening, and make it publicly available on the Internet.



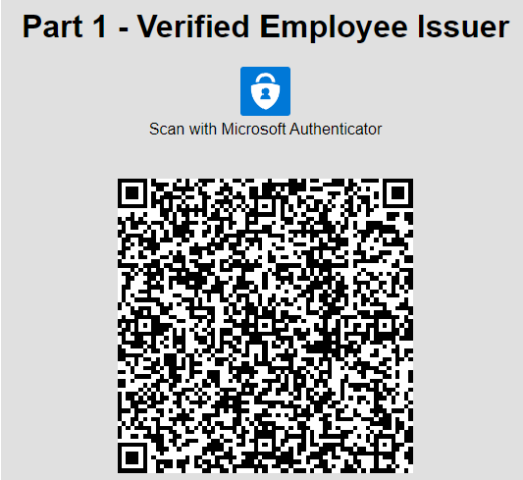
b. Open the HTTPS URL generated by ngrok in a browser session. For example, in our illustration: http://bc34-79-91-126-110.ngrok.io/. As covered above, the code dynamically copies the hostname to be part of the callback URL, this way the Request Service REST API can reach your code sample application to execute the callback method.



Issue a Verified Employee credential

Proceed with the following steps:

1. Select **GET CREDENTIAL** (twice) to issue a Verified Employee credential.



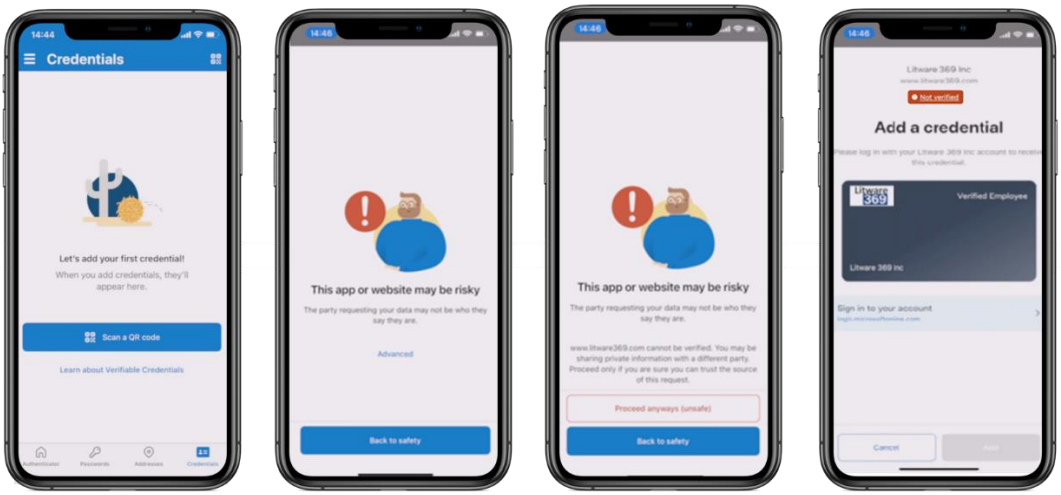
2. Using your mobile device, open Microsoft Authenticator, select **Credentials**, and click **Scan a QR code**. You can also scan the QR code directly from your camera, which will open Authenticator Authenticator for you.

   The QR code informs Microsoft Authenticator where it can retrieve the issuance request. The request defines the Verifiable Credential input type; more on this later. As an alternative to using the QR code, if the website is accessed directly from a mobile broswer, a deep-link will instead trigger Microsoft Authenticator ans the issuance request retrieval.

   If this is the first time you are using Verifiable Credentials (VCs), the Credentials page with the Scan QR button is hidden. You can use the **add account** button. Select **other** and scan the QR code, this will enable the Verifiable Credentials in Microsoft Authenticator.

3. You should now see a **This app or website may be risky** screen, click **Advanced**.
4. On the next **This app or website may be risky** screen, select **Proceed anyways (unsafe)**. The **Add a credential** screen appears.
5. At the top of the screen, you can see a red **Not verified** message. For the local deployment, this is "normal". The domain is effectively not (yet) verified.



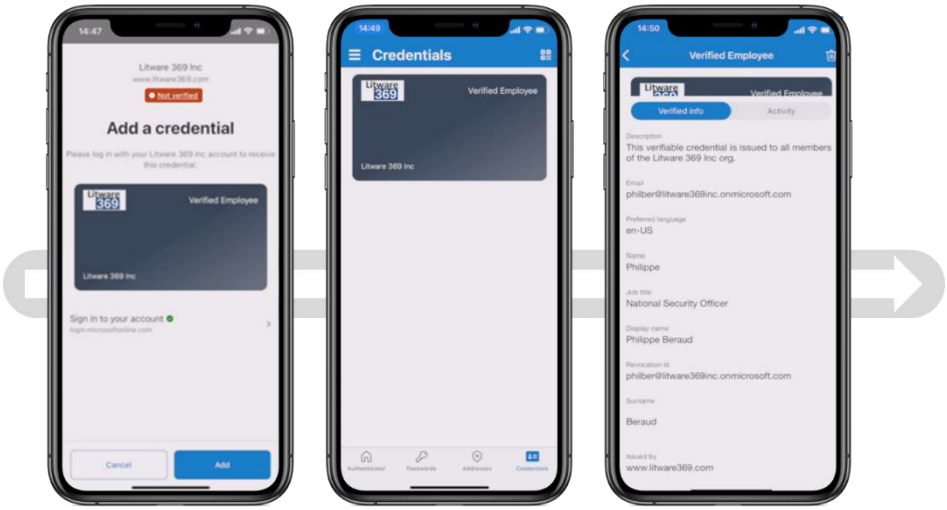


The actions the user will have to performs to obtain a credential depends on how the claims for that credential are both asserted and derived. In this walkthrough, with the managed Verified Employee credential, you need to sign into your Azure AD tenant, and the claims will be obtained from the returned ID token.

6. Click **Sign in to your account**, and follow the instructions
7. Once you have signed in and the ID token is returned, you are prompted to add the credential. So, back to the **Add a credential** screen, click **Add** to accept your new Verified Employee credential.

   Microsoft Authenticator sends the ID token (containing the asserted claims by your Azure AD tenant) and other data to the Microsoft Entra Verified ID service and requests the VC to be issued. The VC is returned and displayed in Microsoft Authenticator.

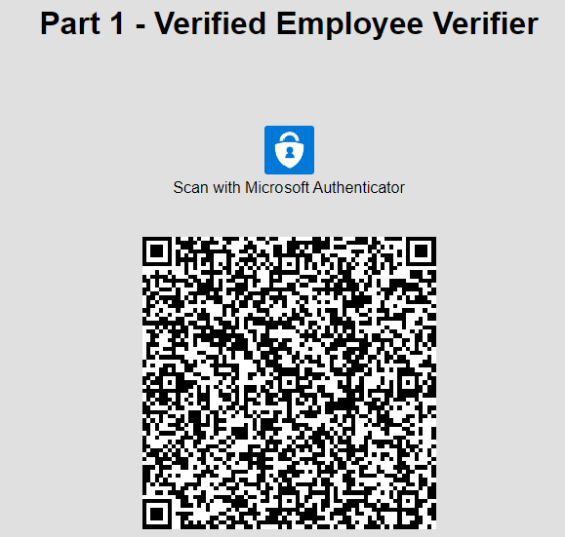
   Congratulations! You now have an Verified Employee credential.
8. Now select your managed Verified employee credential for more details. You can see the related verified information that was asserted by your Azure AD tenant via the issued ID token.



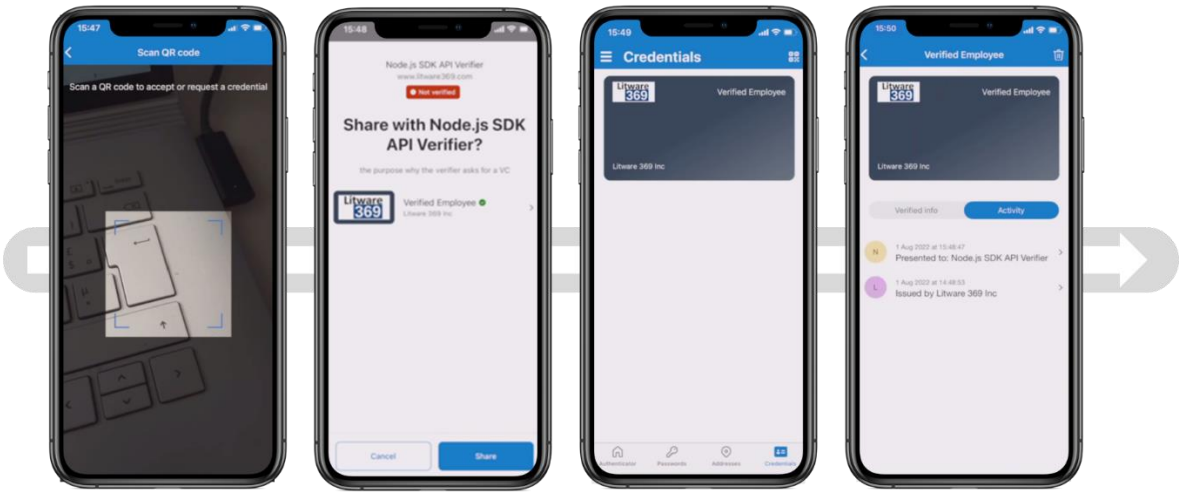
Verify a Verified employee credential

Proceed with the following steps :

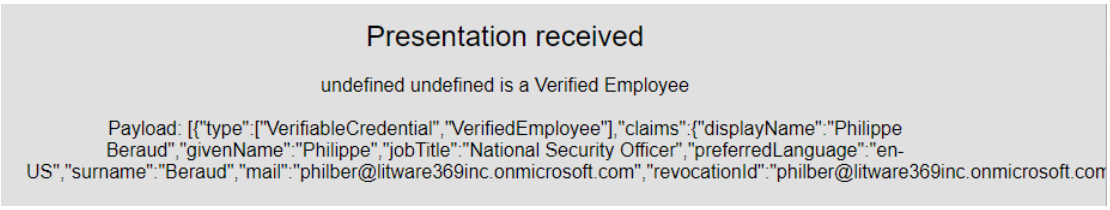
1. Click this time **VERIFY CREDENTIAL** (twice) to verify an issued Verified employee credential. You should see the following screenshot that initiates the Verifiable Credentials verification process.



2. Using Microsoft Authenticator, scan the QR code, or scan it directly from your mobile camera.
3. When you see the warning message, select **Advanced**. Then select **Proceed anyways (unsafe)**.
4. Approve the presentation request by selecting **Share**. After you approve the presentation request, you can see that the request has been approved. You can also check the log. To see the log, select the credential.
5. Select your Verified Employee credential, and then select **Activity**. You can now see the recent activities of your VC.



6. Go back to the code sample application. It shows you that the presentation of the VC(s) was received.



Deploy the code sample application on Internet

At this stage, you will need to create an App Service app or use an app that you created for another purpose. Please note that the web app's App Service plan must be a paid tier and not Free (F1). See Scale up an app to update the tier if needed.

More specifically, as coverred in the next sections, you will need to:

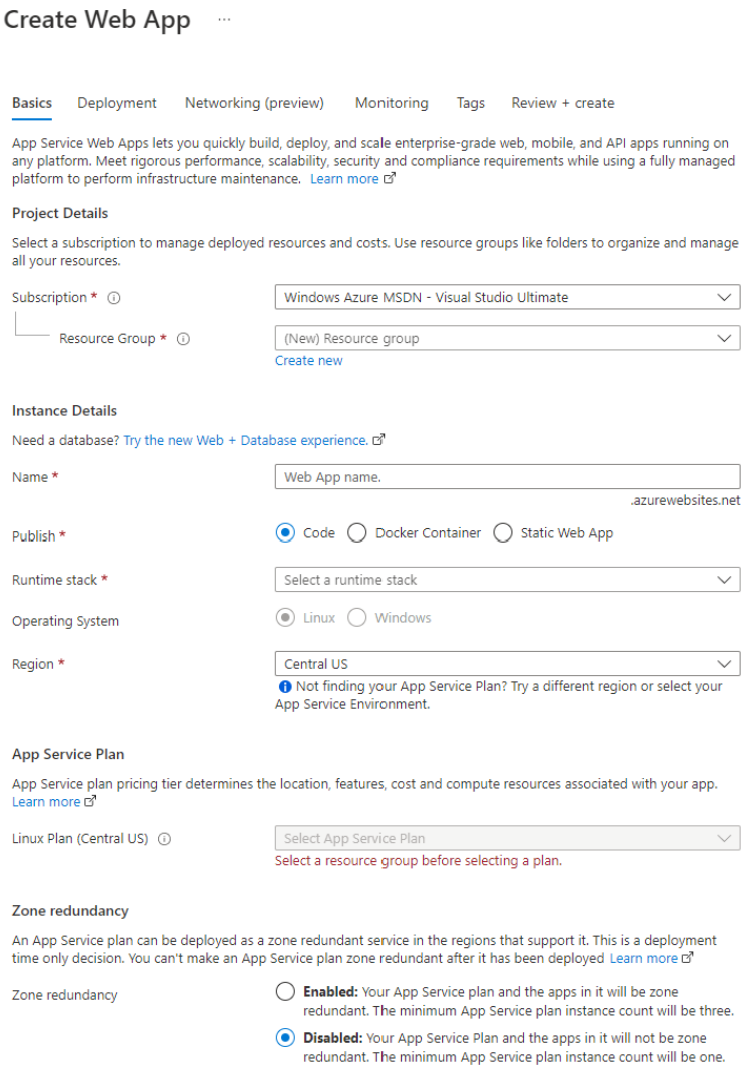
- Create an App Service app.
- Map your custom domain to your App Service app.
- Create a TLS/SSL certificate binding for your App Service app.
- Deploy the code sample application to Azure using Visual Studio code.

Create an App Service app

If you already have a suitable App Service app, you can skip this section.

Otherwise, proceed with the following steps:

1. Open a browser session, navigate to the Azure portal at https://portal.azure.com, and sign in with your Azure administrative credentials.
2. Search for "*app services*", then select **App Services**. An eponym **App Services** page opens up.
3. Click **Create**. A **Create Web App** blabe pops up with the **Basics** tab selected.



4. Under **Project Details**,
    a. **Subscription**: ensure the correct subscription is selected.
    b. **Resource Group**: select the resource group you created previously for the sake of this walktrough. For example, VerifiableCredentialsTest in our illustration.
5. Under **Instance Details**:
    a. **Name**: type a globally unique name for your web app. For example "litware369" in our illustration.
    b. Publish: leave **Code** selected.
    c. **Runtime stack**: select **Node 14 LTS**.
    d. **Operating System**: leave **Linux** selected. A Linux container is used by default.
    e. **Region**: pick the region you want to serve your app from. For example, **France Central** in our illustration.
6. Under **App Service Plan**:
    a. **Linux Plan (<yourRegion>)**: click **Create new**. In the **New App Service Plan** popup, type a name. For example, "*MyAppServicePlanforVC*" in our illustration. Click **OK**.

b. **Sku and size**: click **Change size**. A **Spec Picker** blade pops up. Select any of the non-free tiers (B1, B2, B3, or any tier in the Production category)
c. Select the plan of your choice that includes the **Custom domains / SSL** feature under the **Dev/Test** tab or the **Production** tab.

   Please note that:

   - The **Free (F1)** plans do not support this feature and cannot be thus selected.
   - The first **Basic (B1)** core for Linux is free for the first 30 days.

   Once selected, click **Apply.**

7. Under **Zone redundancy**, leave **Disabled** selected.
8. Click **Review + create**. The deployment process may take few minutes.

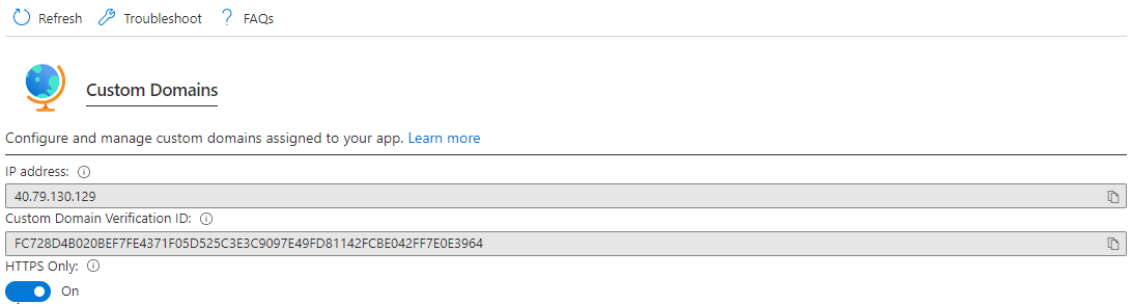
Map your custom domain to your App Service app

See Tutorial: Map existing custom DNS name.

Proceed with the following steps:

1. In the Azure portal, from the **App Services** page, select the name of the App Service app you previoulsy created. You see the management page of the App Service app.
2. To add a custom domain to your app, you need to verify your ownership of the domain by adding a verification ID as a TXT record with your domain provider.

   In the left pane of your app page, select **Custom domains**. A **Custom Domains** page opens up.



   a. **Custom Domain Verification ID**: copy the ID for the next step.
   b. IP address: copy the value of IP address. To map an A record, for your custom domain (top-level domain (TLD) or root domain), you indeed need the app's external IP address.

3. Create the DNS A and TXT records in your domain provider. For example, GoDaddy.com in our illustration for the TLD custom domain litware369.com.

   a. Sign in to the website of your domain provider. For example, https://www.godaddy.com in our illustration.

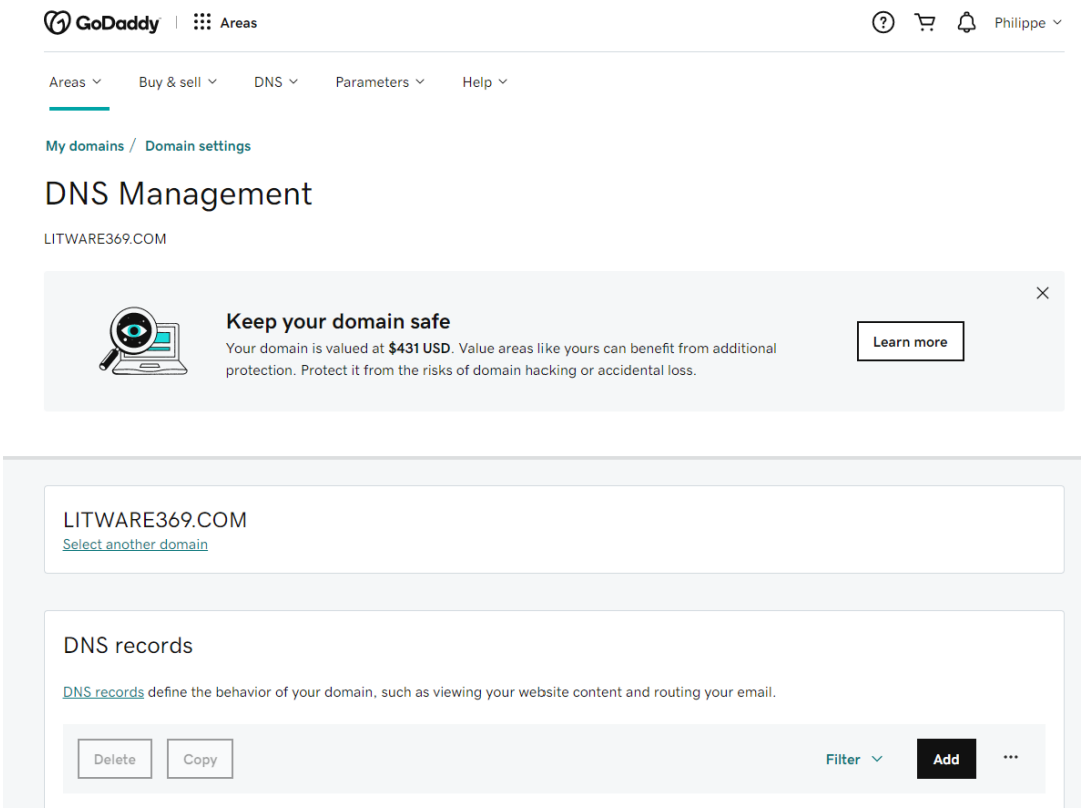
      You can use Azure DNS to manage DNS records for your domain and configure a custom DNS name for Azure App Service. See Tutorial: Host your domain in Azure DNS.

   b. Find the page for managing DNS records.

Every domain provider has its own DNS records interface, so consult the provider's documentation. Look for areas of the site labeled **Domain Name**, **DNS**, or **Name Server Management**.

Often, you can find the DNS records page by viewing your account information and then looking for a link such as **My domains**. Go to that page, and then look for a link that's named something like **Zone file**, **DNS Records**, or **Advanced configuration**.

The following screenshot is the DNS records page for litware369.com at GoDaddy:



i. Select **Add** or the appropriate widget to create a record.
ii. Select the type of record to create and follow the instructions.
iii. For a top level domain like litware369.com, create two records according to the following table:

Record type	Host	Value	Comments
A	@	IP address you got earlier	The domain mapping itself (@ typically represents the top level domain).
TXT	*asuid*	The verification ID you got earlier	For a top-level domain, App Service accesses asuid TXT record to verify your ownership of the custom domain

A record:



TXT record:



For certain providers, such as GoDaddy, changes to DNS records don't become effective until you select a separate **Save Changes** link.

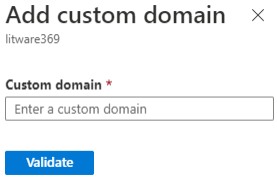
4. In your domain provider, also create a DNS CNAME record for the www subdomain. For example www.litware369.com in our illustration.

   CNAME record:

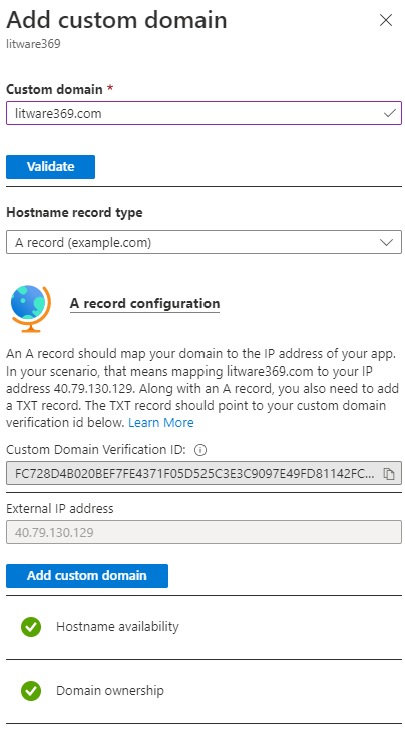


5. After you create DNS records, you enable the mapping in your App Service app. In the Azure portal, from the previous **Custom Domains** page,

   a. Select **Add custom domain**. An eponym **Add custom domain** blade pops up.



   b. In **Custom domain**, type the fully qualified domain name that you configured the A record for, such as litware369.com.
   c. Click **Validate**. The **Add custom domain** blade refreshes.



d. Make sure that **Hostname record type** is set to **A record (example.com)**. Select **Add custom domain**.

   It might take some time for the new custom domain to be reflected in the app's **Custom Domains** page. Refresh the browser to update the data.

   Check if the DNS records are properly configured using an online DNS lookup tool.

A warning label for your custom domain means that it's not yet bound to a TLS/SSL certificate. Any HTTPS request from a browser to your custom domain will receive an error or warning, depending on the browser. So let's add a TLS binding for the sake of our configuration!

Create a TLS/SSL certificate binding for your App Service app

See Add and manage TLS/SSL certificates and Secure a custom DNS name with a TLS/SSL binding in Azure App Service.

You will now secure the custom domain in your App Service app by creating a TLS/SSL certificate binding. When you're finished, you can access your App Service app at the https:// endpoint for your custom DNS name. For example, https://www.litware369.com in our illustration.

Securing a custom domain with a certificate is a two steps process:

1. Add a private certificate to App Service app that satisfies all the private certificate requirements.
2. Create a TLS/SSL binding to the corresponding custom domain.

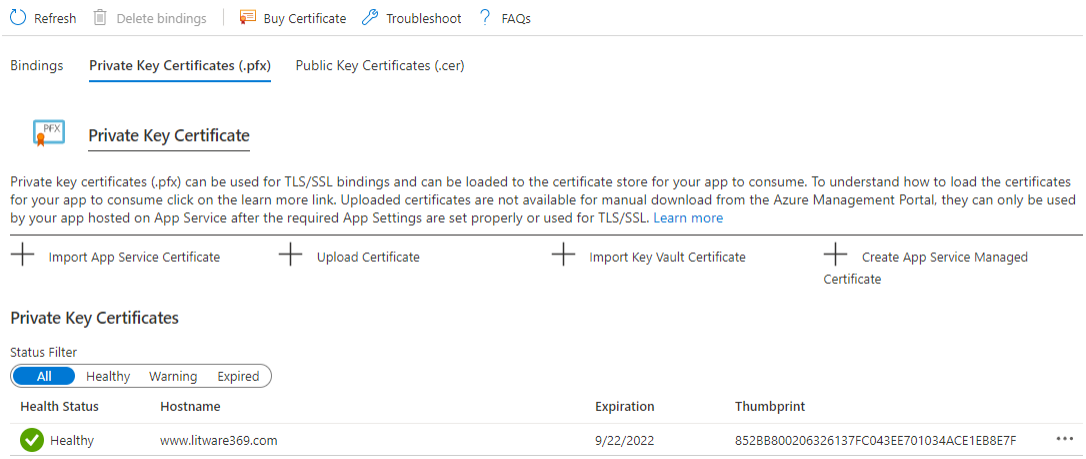
As far as the former is concerned, you will create a free App Service managed certificate. This is a turn-key solution for securing your custom DNS name in App Service. It's a TLS/SSL server certificate that's fully managed by App Service and renewed continuously and automatically in six-month increments, 45 days before expiration, as long as the prerequisites set-up remain the same without any action required from you.

Other options are available to add certificates in App Service: purchase an App Service certificate, import a certificate from Key Vault, upload a private certificate, and upload a public certificate. For the sake of brievety, this walkthrough do not cover them.

For a root domain, (like litware369.com), make sure your App Service app doesn't have any IP restrictions configured. Both certificate creation and its periodic renewal for a root domain depends on your app being reachable from the Internet.

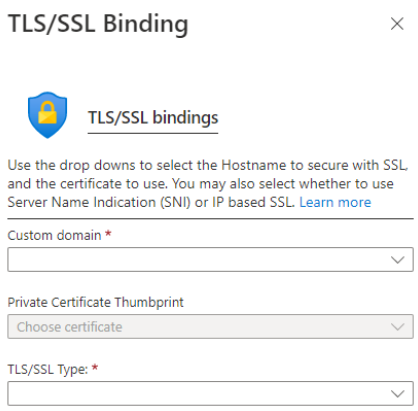
Proceed with the following steps:

1. Create a free managed certificate:
    a. In the Azure portal, from the **App Services** page, select the name of the App Service app you previoulsy created. You see the management page of the App Service app.
    b. From the left pane, select **TLS/SSL settings**, and select the **Private Key Certificates (.pfx)** tab.
        i. Click **Create App Service Managed Certificate**. An eponym **Create a free managed certificate** blade pops up.
        ii. Select the custom domain you created earlier to create a free certificate. For example, litware369.com in our configuration.
        iii. Click **Create**. Note that you can create only one certificate for each supported custom domain.



At this stage, you have fulfilled all prerequisites and are ready to bind the TLS/SSL certificate.

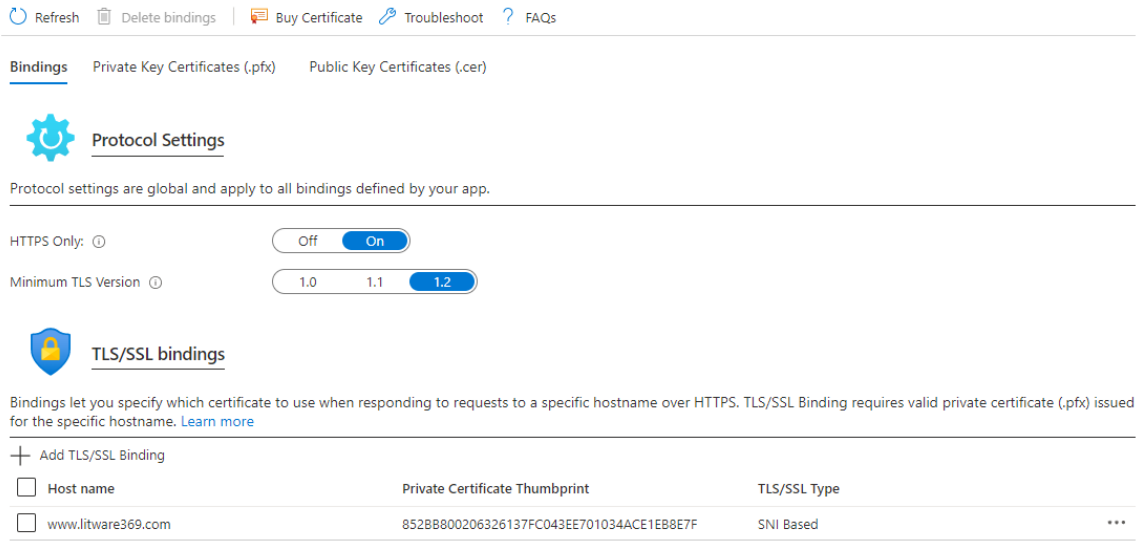
2. Create a TLS/SSL binding to the corresponding custom domain.
    a. From the same the page, now select the **Binding** tab, under **TLS/SSL bindings**, click **+ TLS/SSL Binding**. An eponym **TLS/SSL Binding** blade pops up.



b. Fill the following information:

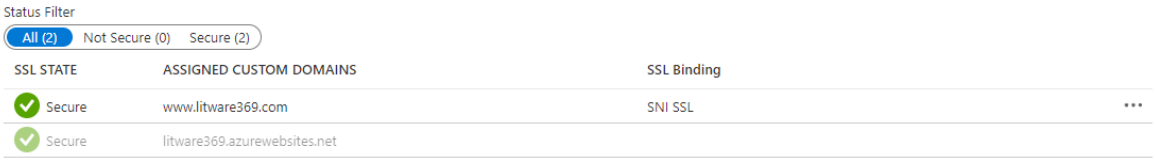
- **Custom domain**: select your custom domain.
- **Private certificate Thumbprint**: choose the certificate you created previously.
- **TLS/SSL Type**: select **SNI SSL**.

c. Click **Add binding**.



As you can see, your App Service app allows TLS 1.2 by default, which is the recommended TLS level by industry standards.

d. Once the operation is complete, the custom domain's TLS/SSL state is changed to Secure.



At this stage, you have successfully deployed your App Service app, mapped it to your custom domain name, and secured by using TLS/SSL certifate binding.

Please note that, in App Service, TLS termination happens at the network load balancers, so all HTTPS requests reach your App Service app as unencrypted HTTP requests. If your app logic needs to check if the user requests are encrypted or not, inspect the `X-Forwarded-Proto` header.

Language specific configuration guides, such as the Linux Node.js configuration guide, shows you how to detect an HTTPS session in your application code logic.

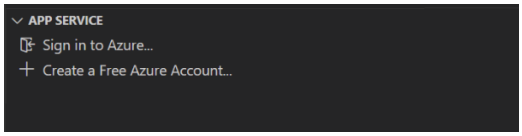
Deploy the code sample application to Azure using Visual Studio code

Azure offers several ways to deploy your apps to App Service. In this section, you will deploy the sample on Azure using Visual Studio Code.

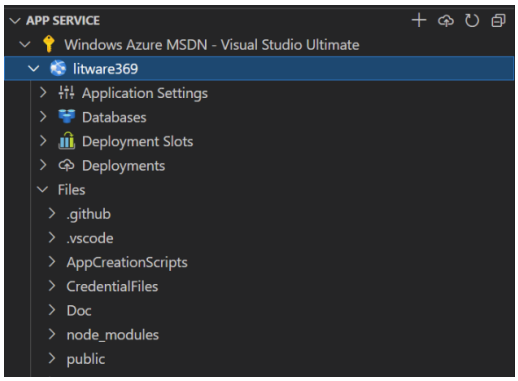
At this stage, and as the prerquisites for this end-to-end walkthrough as per chapter **Fullfil the prerequisites for your testing environment**, we asume that you have an Azure account and active subscription in place, Nodejs and npm installed on your local development machine, Visual Studio Code and App Service extension for Visual Studio Code, etc.

With that, to deploy the code sample application, proceed with the following steps:

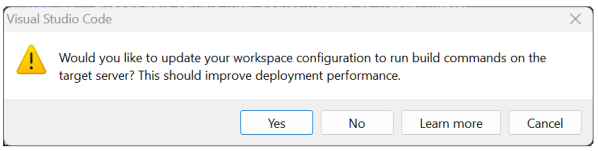
1. Open the sample code application on Visual Studio Code.
2. In Visual Studio Code, in the Activity Bar, select the **Azure** logo.
3. In the **APP SERVICE** explorer, select **Sign in to Azure...** and follow the instructions.



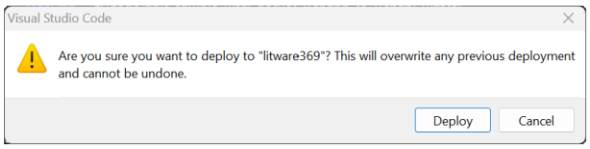
4. Once you are signed in, and have your app open in Visual Studio Code, you should now see in the Status Bar your Azure account, and under the **APP SERVICE** explorer, your subscription with underneath the App Service app you previoulsy created. For example, litware369 in our illustration.



5. Select the **Deploy to Web App** icon.
    a. Select the folder to deploy, i.e., the folder where you previously clone the code sample repo. For example, *C:\entra-verifiedid-tour\part-1* in our illustration.
    b. Select the App Service app your previously created. For example, litware369 in our illustration.
    c. The following Visual Studio Code message box might pop up. If so, click **Yes**.



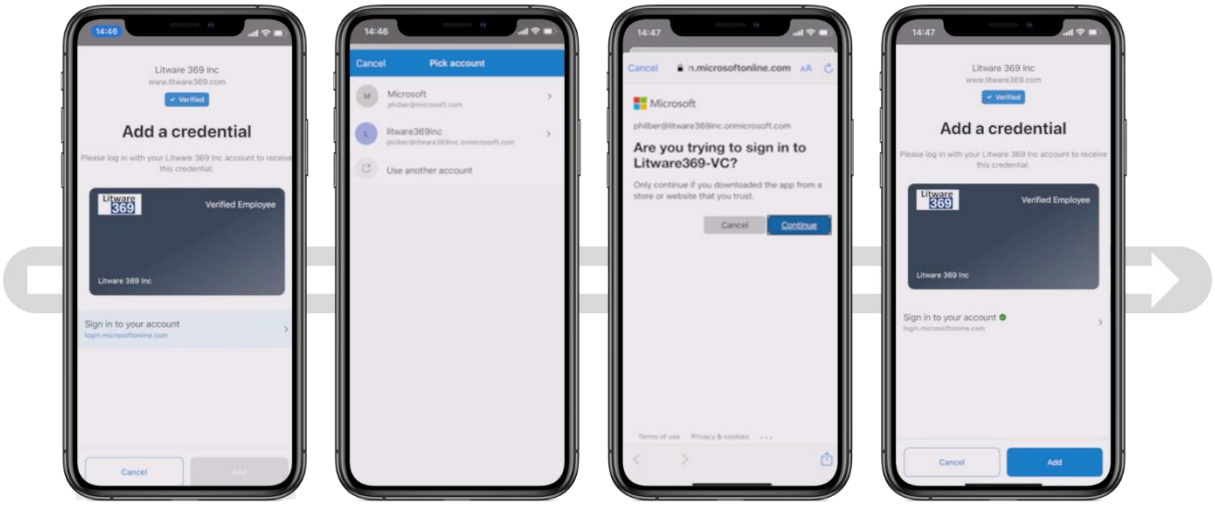
d. When invited, click **Deploy**.



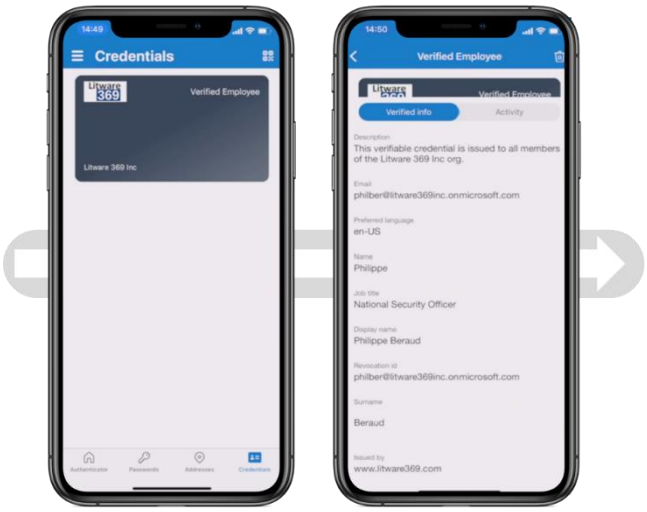
e. While Visual Studio Code provisions the Azure resources and deploys the code, it shows progress notifications.
f. Once deployment completes, select **Browse Website** in the notification popup. The browser should display the code sample application's default page.
g. As before, Select **GET CREDENTIAL** to issue a VC, and proceed with the exact same steps.
    i. Using your mobile device, scan the QR code with Microsoft Authenticator. You no longer have any warning. The **Add a credential** screen appears right away. At the top of the screen, you can see this time a blue **Verified** message.

    The issuer's DID is contained in the issuance request retrieve by Microsoft Authenticator, and the associated public key, contained in the DID document, is used to verify the signature. That DID can now be verified as belonging to our fictitious company Litware369 Inc. via the DNS binding you previouly established. The Authenticator has checked the signature and the DNS binding and has added the blue **Verified** message for www.litware369.com.

    ii. Click **Sign in to your account**. A **Pick an account** screen opens up this time. Select the (test) account you previously used or pick a new one that you also created in your Azure AD tenant. Click **Continue**. For example, you can pursue with the same account.
    iii. An **Are you trying to sign in to Litware369-VC** screen opens up. Click **Continue**.
    iv. Back to the **Add a credential** screen, click **Add** to accept your new verifiable credential.



h. As before, select your Verified Employee credential for more details. You can see the related verified information that was asserted by your Azure AD tenant via the issued ID token.



See [Create a Node.js web app in Azure] for more details.

If needed, you can deploy changes to this app by making edits in Visual Studio Code, saving your files, and then redeploy to your Azure app.

For example, proceed with the following steps:

1. Change the h1 element from:

```
<h1 style="color: #fff;">Litware 369 Verifiable Credential Employee Issuance and Verifier</h1>
```

to :

```
<h1 style="color: #fff;">Your Company Verifiable Credential Employee Issuance and Verifier</h1>
```

2. Like you did previously, in the **APP SERVICE** explorer, select the **Deploy to Web App** icon again. Confirm by clicking **Deploy** again.
3. Once deployment completes, select **Browse Website** in the notification popup. You should see the above change reflected in the default page.

Congratulations, you have successfully deployed your code sample application on Azure. You can use it in the same way as illustrated above.

Verify your own Directory-based employees' credentials as another organization

See How to use the Microsoft Entra Verified ID Network.

To conclude this chapter, let's suppose that the Woodgrove (fictitious) company with whom our company, i.e., Litware369 Inc. for the sake of our illustration, has established a business relationship would like to verify the priorly issued Directory-based employees' credentials before granted your full-time employees (FTEs) an access.

Woodgrove as the verifier needs to know the issuer DID of our company as the issuer of these Verifiable Employee credentials, and the related Verifiable Credential (VC) type that represents consequently your employees before it can create a suitable presentation request for these VCs for Litware369 Inc. FTEs. The necessary information may come from some kind of manual exchange between the two companies, but such an approach would be both a manual and a complex.

The already introduced Entra Verified ID Network makes this process much easier. Litware369 Inc., as an issuer, can indeed publish VC types to the Entra Verified ID Network and Woodgrove, as the verifier, can search for published VC types and related schemas in the Entra Verified ID Network. Using this information, Woodgrove can then create a presentation request and easily invoke the Request Service API to verify an issued Verified Employee credential by Litware369 Inc..

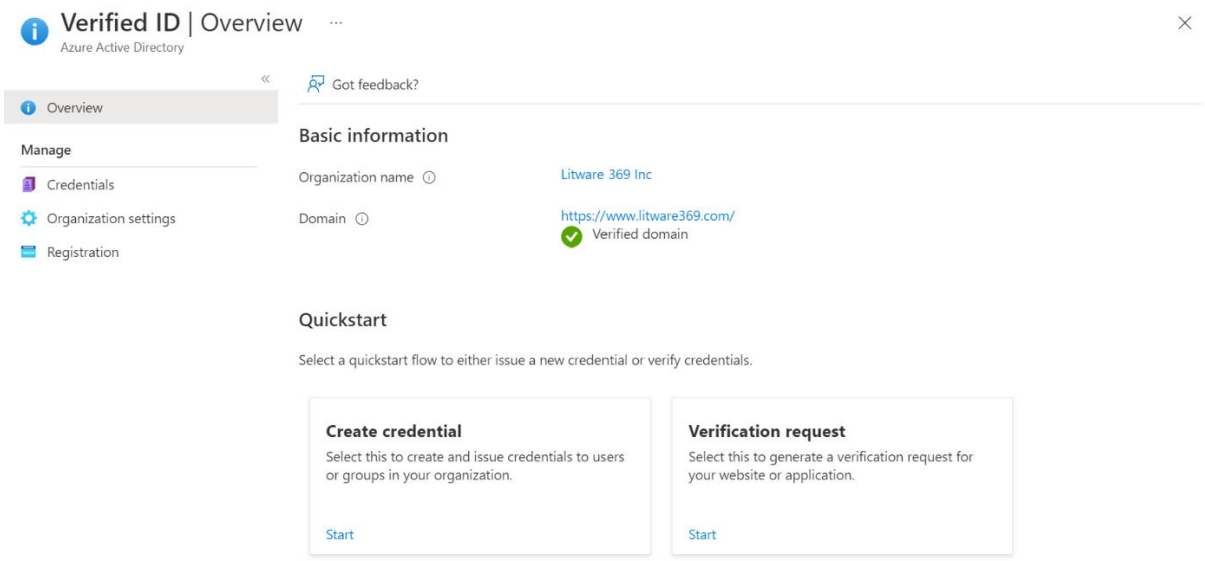
Each credential type that is created has an attribute named `availableInVcDirectory` that makes it visible in the list. You can update this attribute to make the credential type visible or not. See Admin API for managing Microsoft Entra Verified ID.

To use the Entra Verified ID Network, proceed with the following steps:

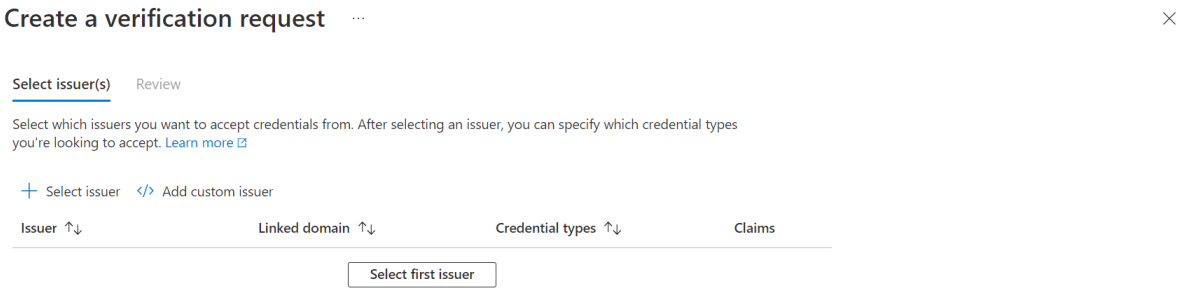
1. Open a browser session, navigate to the Azure portal at https://portal.azure.com, and sign-in as a global administrator.
2. Search for "*verified id*", and then click **Verified ID** to open the eponym blade. Please note that you can start using the **unified Microsoft Entra admin center** instead at https://entra.microsoft.com/.

   For the sake of the simplicity, we use here the Litware369 Inc. Azure AD tenant but for the above scenario, it would be normally the one of the Woodgrove company where a Verified ID instance would have setup.

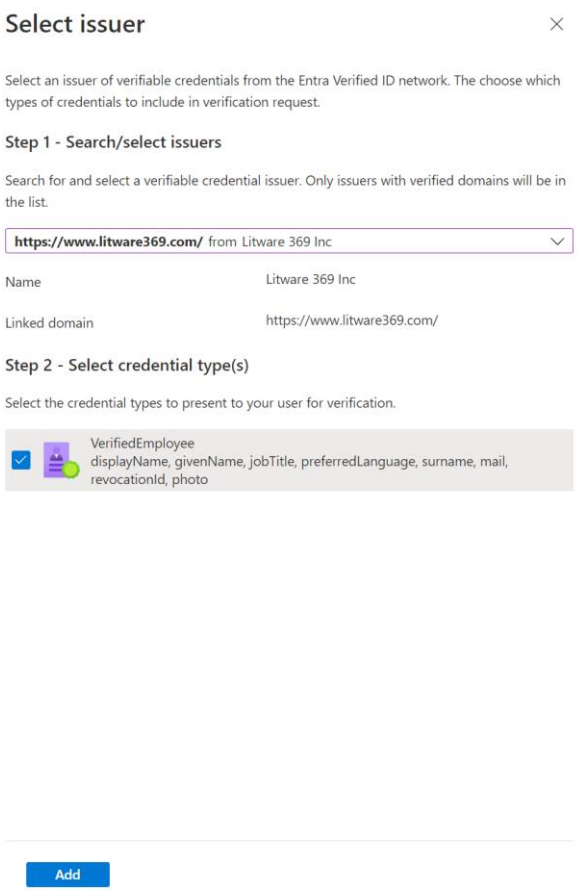
   In the start page of Microsoft Entra Verified ID in the Azure portal, you have a Quickstart named Verification request. Clicking on start will take you to a page where you can browse the Verifiable Credentials Network.
3. In the start page, you have a **Quickstart** named **Verification request**.



4. Click **Start**. This takes you to a page where you can browse the Entra Verified ID Network.

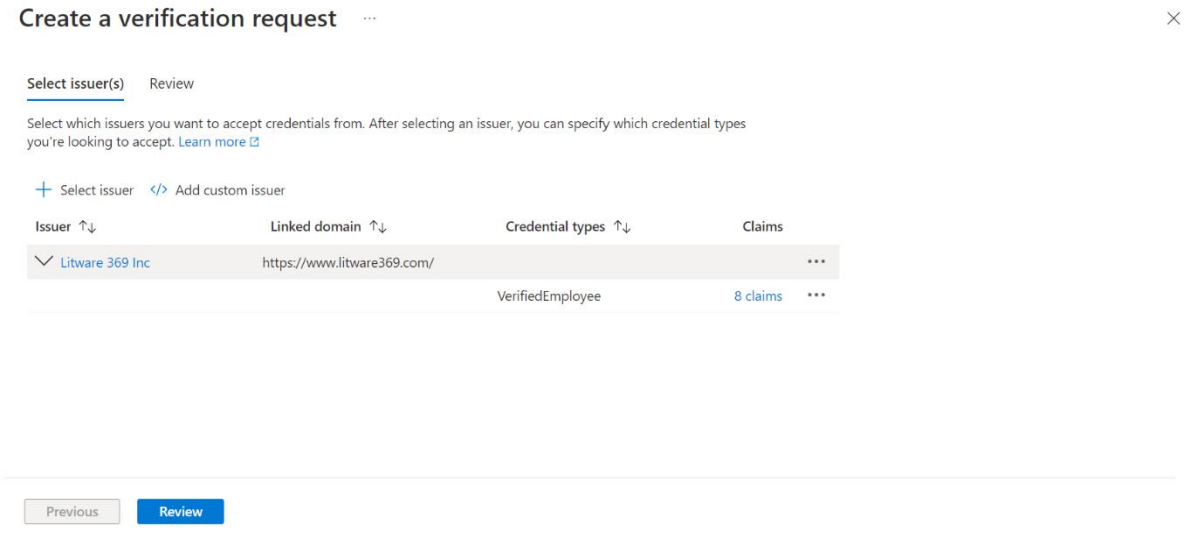


5. Click **Select first issuer**. A **Select issuer** blade opens up.
    a. In **Step 1 - Search/select issuers**, search for "*litware*" and select **https://www.litware369.com/**.
    b. In **Step 2 - Select credential type(s)**, now select **VerifiedEmployee**.



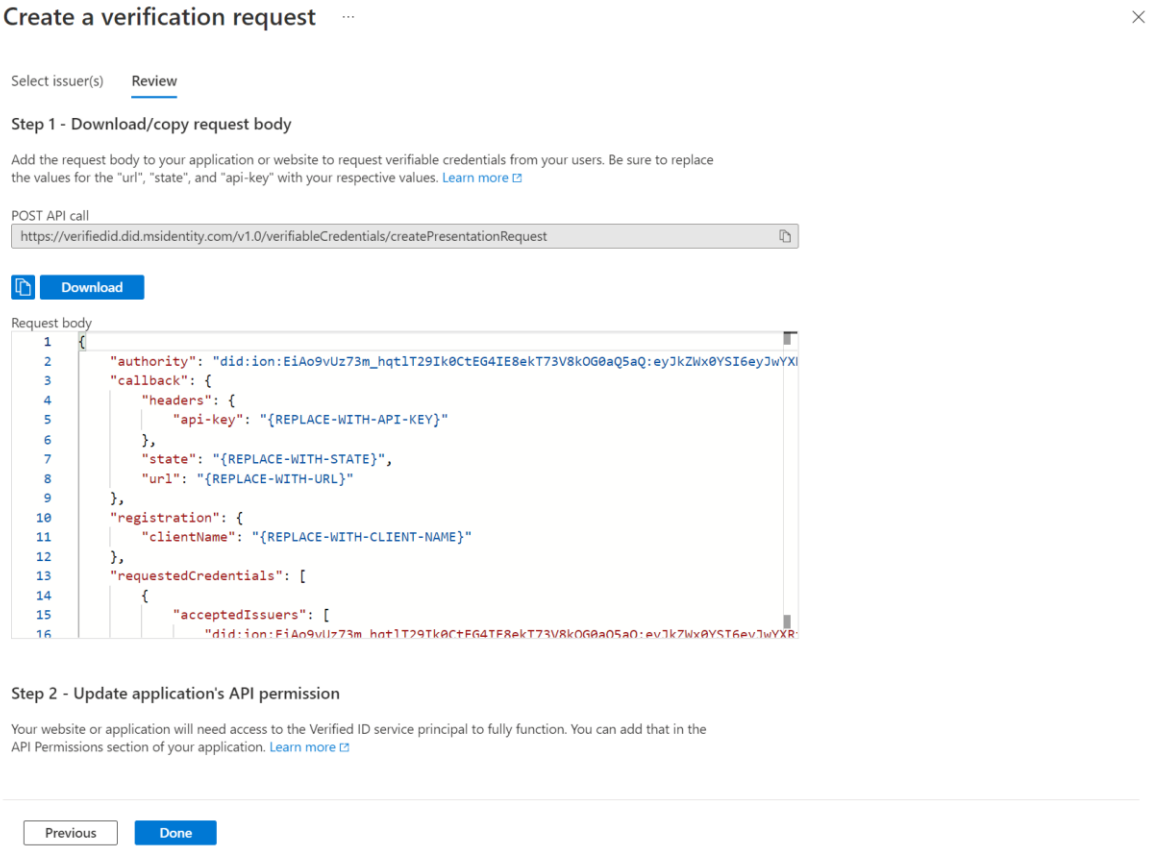
c. Click **Add**.

6. In the first screen, you now have Litware369 Inc. in the issuer list.



Click **Review**.

7. This screen now displays a skeleton presentation request JSON payload for the Request Service API.



8. Click **Download**. A *verifiedRequestPayload.json* file is downloaded.
9. Open the file.


```
{
    "authority":
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG
9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1NmsxIiwia3R5Ijoi
RUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LW
VzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3Ay
NTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6ImxpbmtlZGRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2
lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6Ikxpbmtl ZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50
Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5Zj
ZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYy
OERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNk
JPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMlldG5tNGcifX0",
    "callback": {
        "headers": {
            "api-key": "{REPLACE-WITH-API-KEY}"
        },
        "state": "{REPLACE-WITH-STATE}",
        "url": "{REPLACE-WITH-URL}"
    },
    "registration": {
        "clientName": "{REPLACE-WITH-CLIENT-NAME}"
    },
    "requestedCredentials": [
        {
            "acceptedIssuers": [
```

```
"did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZXBsYWNlIiwiZG
9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijoic2VjcDI1NmsxIiwia3R5Ijoi
RUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInkiOiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LW
VzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uIiwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3Ay
NTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIjpbeyJpZCI6ImxpbmtlZGRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2
lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50
Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5Zj
ZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYy
OERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNk
JPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0"
            ],
            "purpose": "{REPLACE-WITH-PURPOSE}",
            "type": "VerifiedEmployee"
        }
    ]
}
```


The important pieces of information are the DID inside the `acceptedIssuers` collection and the type value. This information is needed to create a presentation request. The request prompts the user for a credential of a certain type issued by a trusted organization, here the expected Verified Employee credential you created earlier.

You will see in the second part how to use such a payload.

With that, and for the moment, it's now high time to see how you can export and import your issued Verified Employee credential.

Export and import your employees' credentials

Microsoft Authenticator users can recover their Verifiable Credentials (VCs) in the event of a loss of device.

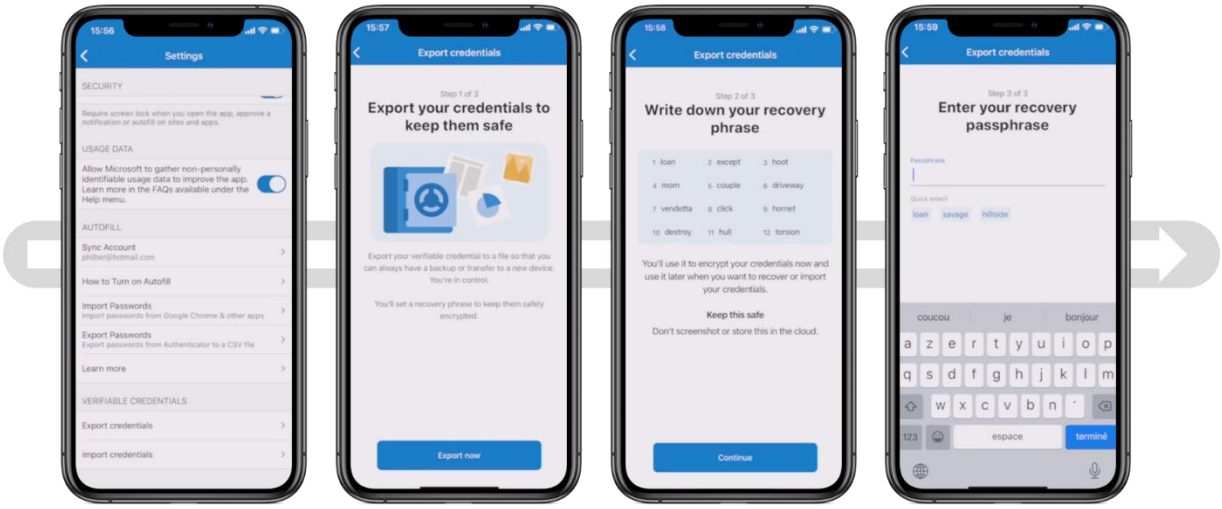
So let's see how you can set up credentials and securely recover them if your device is lost.

Export your employees's credentials

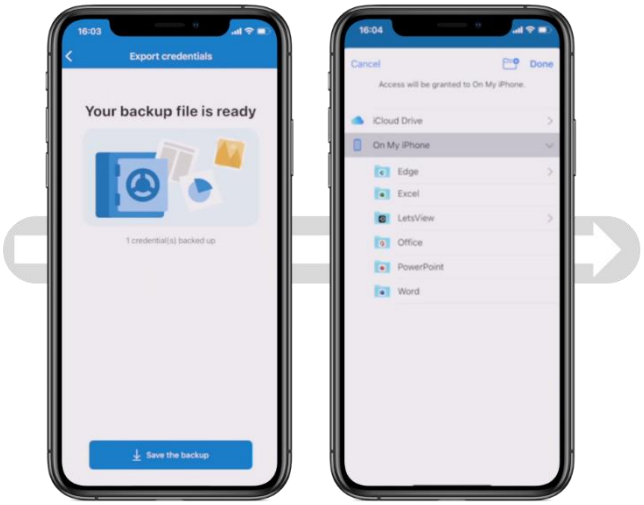
As a user, you can export your employees' credentials and secure the exported file using a pass phrase to ensure that only you can access your encrypted credentials.

To export your employee's credentials, proceed with the following steps:

1. Open Microsoft Authenticator on your device and select **Settings**.
2. Scroll down to **Verifiable Credentials**, and then select **Export Credentials** underneath.
3. On the **Export your credentials to keep them safe** page, click **Export now**.
4. As invited, write down the displayed recovery phrase. Click **Continue**.
5. Now enter your recovery phrase, and press ENTER.



6. On the **Your backup file is ready** page, click **Save the backup**.
7. Specify a location where to save the backup and click **Done**.

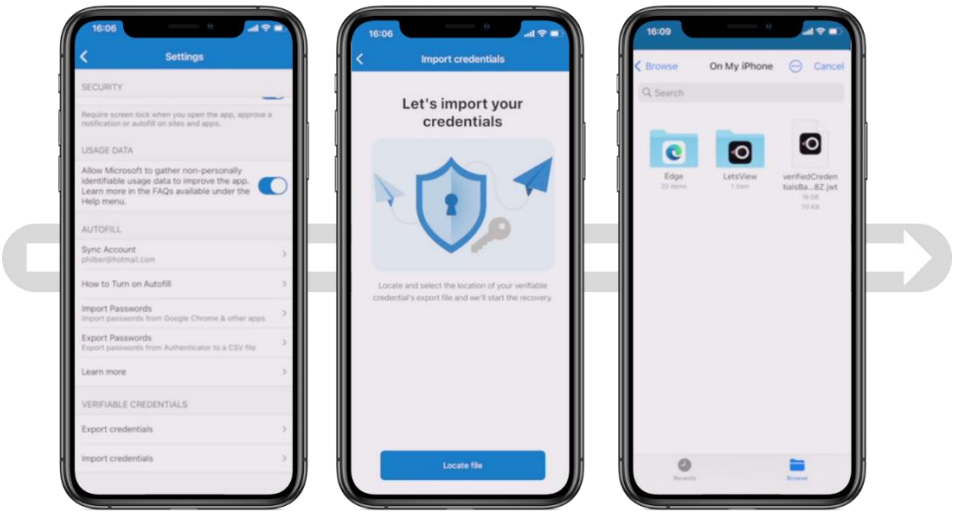


Import your employee's credentials

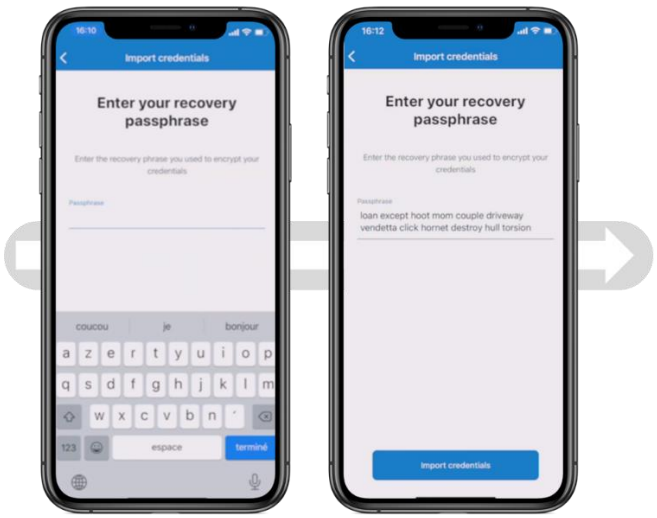
If you lose your phone, you can use Microsoft Authenticator on the new device to import credentials using the file, along with the passphrase.

To import your employee's credentials, proceed with the following steps:

1. Open Microsoft Authenticator on your device and select **Settings**.
2. Scroll down to **Verifiable Credentials**, and then select this time **Import Credentials** underneath.
3. Click **Locate file**, and then select the file *verifiedCredentials[…omitted…].jwt* from the above location where you priorly exported your employees' credentials.



4. Now enter your recover phrase, press ENTER.
5. Click **Import credentials** to complete the process. Et voilà!



With that, let's now consider from an admin prespective how you can suspend or revoke or suspend an issued Verified Employee credential. This is the purpose of the next chapter.

Suspend or revoke your employees' credentials

Credentials remain in the individual's possession, but issuers can suspend or revoke the active status of these credentials, meaning they'll no longer be verified as valid by the issuer.

Each customer will have their own unique reason's for wanting to suspend or revoke a verifiable credential, but there are some common themes we have heard thus far: more particulalry, as far as our Employee journey is concerned, the employee is no longer an active employee.

Understand how revocation works

From a standard perspective

Let's first go over the **Status** property part of the Verifiable Credentials Data Model v1.1 (W3C) specification and take a closer look at the revocation process, and why you may want to revoke priorly issued credentials along with some data and privacy implications.

This specification references the **status** property in section 4.9:

§ 4.9 Status

> This specification defines the following `credentialStatus` property for the discovery of information about the current status of a verifiable credential, such as whether it is suspended or revoked.

However, the above W3C specification doesn't define a format on how status check should be implemented.

> Defining the data model, formats, and protocols for status schemes are out of scope for this specification. A Verifiable Credential Extension Registry [VC-EXTENSION-REGISTRY] exists that contains available status schemes for implementers who want to implement verifiable credential status checking.

For now, Microsoft's status check implementation is proprietary but we are actively working with the DID community to align on a standard.

Microsoft status check implementation

Every verifiable credential issued with Microsoft Entra Verified ID contains an attribute called `credentialStatus`. This attributes is populated with a status API that Microsoft manages on your behalf.

The following .json file illustrates what it looks like:

```
"credentialStatus": {
      "id": "https://portableidentitycards.azure-api.net/v1.0/7952032d-d1f3-4c65-993f-
1112dab7e191/portableIdentities/card/status",
      "type": "PortableIdentityCardServiceCredentialStatus2020"
}
```

Once the API is called and provided the right information, the API will return either a `true` or `false`: `true` being the verifiable credential is still active with the issuer, and conversaly, `false` signifying the verifiable credential has been actively revoked by the issuer.

All verifiable credential data is not stored with Microsoft by default. Therefore, we do not have any data to reference to revoke a specific verifiable credential ID. You as the issuer needs to specify a specific field from the above-mentioned verifiable credential attribute for Microsoft to index and subsequently salt and hash.

You can tell Microsoft which attribute of the VC you would like to index. The implication of indexing is that indexed values may be used to search your verifiable credentials for the VCs you want to revoke. For example, an email field in our illustration.

It's now time to see how this works in actions!

Revoke a previously issued Verified Employee credential

See How to Revoke a Verifiable Credential as an Issuer.

Please note that you have to complete all the steps outlined in the previous chapter before starting this one.

For the sake of the illustration, consider the following example :

Dan is a Litware369 Inc. employee. Dan left Litware369 Inc. to work at Contoso. Jane, the IT admin for Litware369 Inc., searches for Dan's email in the Verifiable Credentials Revoke search query. In this example, Jane, indexed the email field of the Litware369 Inc. Verified Employee credential.

Lets' see how to configure your environment to enable this example.

Revoke a credential

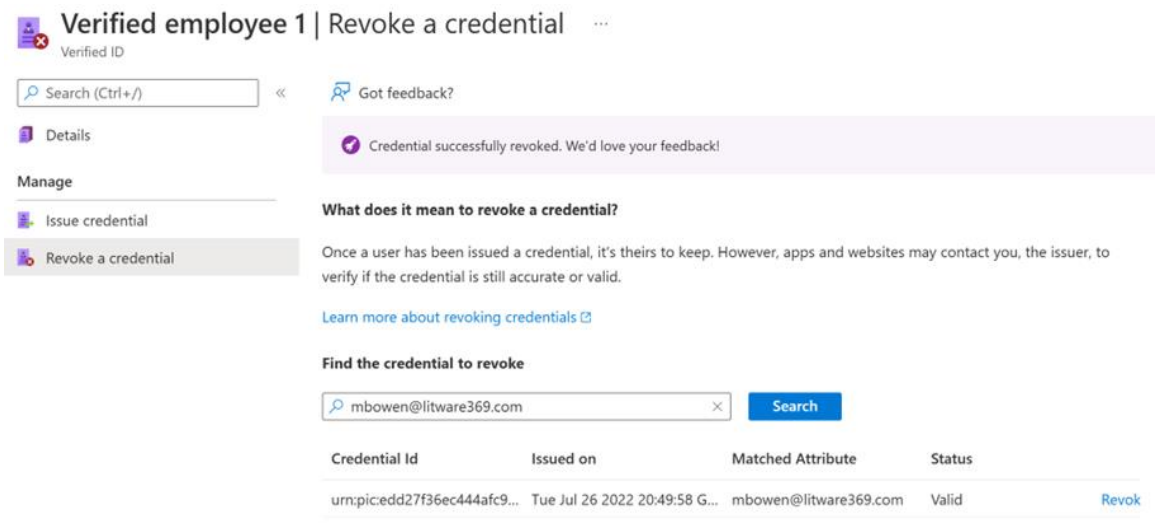
Once an index claim has been set and verifiable credentials have been issued to your users, it's now time to see how you can revoke a verifiable credential in the VC blade.

Proceed with the following steps:

1. Open a browser session, navigate to the Azure portal, sign in with your administratrive credentials, search for "*verified id*", and then click **Verified ID** to open the eponym blade.

   As already mentioned, you can use instead the newly introduced the unified Microsoft Entra admin center. In this case, also click **Verified ID**.

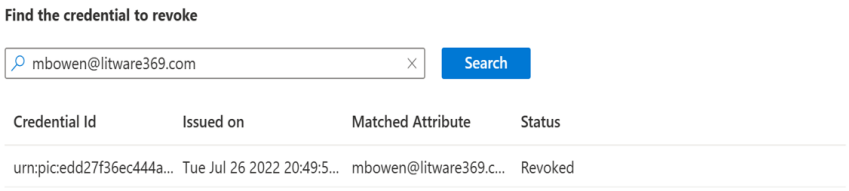
2. In the left pane, click **Credentials** under **Manage**.
3. Now select the Verified Employee credential.
4. On the left pane, select **Revoke a credential** under **Manage**.
5. Search for the index attribute of the user you want to revoke.



Since we are only storing a hash of the indexed claim from the Verified Employee credential, only an exact match will populate the search results.

1. Once you've found a match, select the **Revoke** option to the right of the credential you want to revoke.

   After a successful revocation, you should see the status being updated. Now whenever a relying party calls to check the status of this specific verifiable credential, Microsoft's status API, acting on behalf of the tenant, will return a `false` response.



Enhanced privacy support for credential revocation

Microsoft Entra Verified ID customers can benefit from a higher degree of privacy. The Microsoft Entra Verified ID service indeed supports the W3C Status List 2021 Privacy-preserving status information for Verifiable Credentials standard. See Credential Revocation with Enhanced Privacy.

Each Azure AD issuer tenant has an Identity Hub endpoint used by verifiers to check on the status of a verifiable credential using a privacy-respecting mechanism. An Identity Hub is yet another flavor of a verifiable data repository. The identity hub endpoint for that tenant is published in the DID document.

To see this endpoint in the DID document, proceed with the following steps:

1. Navigate to Azure portal, and go to the **Verifiable Credentials Service**.
2. Navigate to **Organization Settings**.
3. Copy your organization's **Decentralized Identifier (DID)**. For example, in our illustration:


```
did:ion:EiAo9vUz73m_hqtlT29Ik0CtEG4IE8ekT73V8kOG0aQ5aQ:eyJkZWx0YSI6eyJwYXRjaGVzIjpbeyJhY3Rpb24iOiJyZX
BsYWNlIiwiZG9jdW1lbnQiOnsicHVibGljS2V5cyI6W3siaWQiOiJzaWdfMWVlZDk3MTMiLCJwdWJsaWNLZXlKd2siOnsiY3J2Ijo
ic2VjcDI1NmsxIiwia3R5IjoiRUMiLCJ4IjoiazN2QU8xTEI4eENPMGwwaEY1MTA2ZjczemNGa3BoLXl1aGMtOXB2MGN5OCIsInki
OiJpQk40aUh6QjF1RGNoTC1tZFYzRTNBU3Y2LWVzLTlFUG9GTDRLWlJ6ZWxRIn0sInB1cnBvc2VzIjpbImF1dGhlbnRpY2F0aW9uI
iwiYXNzZXJ0aW9uTWV0aG9kIl0sInR5cGUiOiJFY2RzYVNlY3AyNTZrMVZlcmlmaWNhdGlvbktleTIwMTkifV0sInNlcnZpY2VzIj
```

pbeyJpZCI6ImxpbmtlZGRvbWFpbnMiLCJzZXJ2aWNlRW5kcG9pbnQiOnsib3JpZ2lucyI6WyJodHRwczovL2xpdHdhcmUzNjkuY29tLyJdfSwidHlwZSI6IkxpbmtlZERvbWFpbnMifSx7ImlkIjoiaHViIiwic2VydmljZUVuZHBvaW50Ijp7Imluc3RhbmNlcyI6WyJodHRwczovL2JldGEuaHViLm1zaWRlbnRpdHkuY29tL3YxLjAvMTY0ZjY4NTQtYTU1My00MDg0LTlkYzgtNDk5ODE5ZjZmY2ViIl19LCJ0eXBlIjoiSWRlbnRpdHlIdWIifV19fV0sInVwZGF0ZUNvbW1pdG1lbnQiOiJFaUFnbWhNa3NTT1pOWnBubDJWUENJNXc4RFpPOGYyOERnQ3JjMnlqeFlYSUtBIn0sInN1ZmZpeERhdGEiOnsiZGVsdGFIYXNoIjoiRWlBN1F4bkdCVFg1UHJwYWtPNk9jNm9MeDNUeVNMZVdGakV4OHhHNkJPNVhjZyIsInJlY292ZXJ5Q29tbWl0bWVudCI6IkVpQ0t1VnltbVU3eFBlaUVSdzlna25kZ0czaEFhVG5KaVJqUm1hMllldG5tNGcifX0

4. Open a new tab, navigate to the ION Explorer at https://identity.foundation/ion/explorer/, paste your DID in the search box, and click **Search**.
5. Inspect your DID document and search for the `"#hub"` node.

```
{
   "id": "#hub",
       "type": "IdentityHub",
       "serviceEndpoint": {
         "instances": [
           "https://beta.hub.msidentity.com/v1.0/164f6854-a553-4084-9dc8-499819f6fceb"
         ],
         "origins": []
       }
 }
```

As already outlined, for the verifiable credentials contracts your create, in the rules, file you must add the `"credentialStatusConfiguration": "anonymous"` property to start using the feature in combination with the Identity Hub endpoint.

As a conclusion

This concludes this end-to-end walkthrough and our first part of the guide.

We hope you have enjoyed the guided tour and you are not too tired with all those technical details.

As part of this first part of the guided tour, we have outlined all the steps required to setup your own demo environment in Azure with the Microsoft Entra Verified ID, issue, verify, and revoke Directory-based employees's Verifiable Credentials (VCs) along with some additionnal explanations and related rational.

Furthermore, we have also some system integration partners ready to help you quickly do (further) prototyping, proof of concepts, pilots, etc.

In the second part of the guide, we will cover the custom credentials than you can also issue and verify thanks to Microsoft Entra Verified ID.

Going beyond

In the meantime, to continue learning about the passionate subject of verifiable credentials and more generally speaking of decentralized identity, please check out these additional resources:

- Understand the basics of decentralized identity via our five-part blog series.
- Learn more about the decentralized identity movement at aka.ms/ownyouridentity.
- Get involved with http://identity.foundation , the industry working group for all things Decentralized ID (DID).
- Quick overview: http://aka.ms/didexplained.
- Documentation for developers: http://aka.ms/didfordevs.





Verify once. use everywhere. Start your decentralized identity journey today with verifiable credentials in Microsoft Entra

aka.ms/verifyonce



Let’s start building a more trustworthy Internet together

and realize the value that accompany it

We think this is the right problem for us to solve as both a technical leader, an industry and an open community.

We've kind of been held up on the centralized way of doing it, and, at Microsoft, we think this kind of opening up identity systems may unlock a more trustworthy, scalable way of solving these problems.

Copyright © 2022 Microsoft France. All right reserved.

Microsoft France
39 Quai du President 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.