Microsoft

Corda Enterprise Scale Deployment on Microsoft Azure

Corda Enterprise - Automated Deployment, Security & Monitoring

Prepared by

Gurpreet Singh (GP), Sergio Figueiredo

Program Manager, Partner CAT

gurpreet.singh@microsoft.com,

sergio.figueiredo@microsoft.com

Table of Contents

1. Overview

This document explains how to automate the deployment and monitoring of Corda Enterprise nodes on Microsoft Azure. We'll briefly explain the high-level architecture before describing in detail the automated deployment of Corda nodes along with the design needed to support monitoring and telemetry requirements of such a solution. This document does not go into the details about the Corda Enterprise architecture/design, although we do provide links to the Corda Enterprise documentation for reference wherever applicable. Major topics covered in this document include –

- How to deploy Corda network on Microsoft Azure?
- How to integrate Corda with Azure Key Vault?
- How to secure Corda network communication?
- How to manage Monitoring and Telemetry on Microsoft Azure?

2. Reference Implementation

2.1. Business Scenario

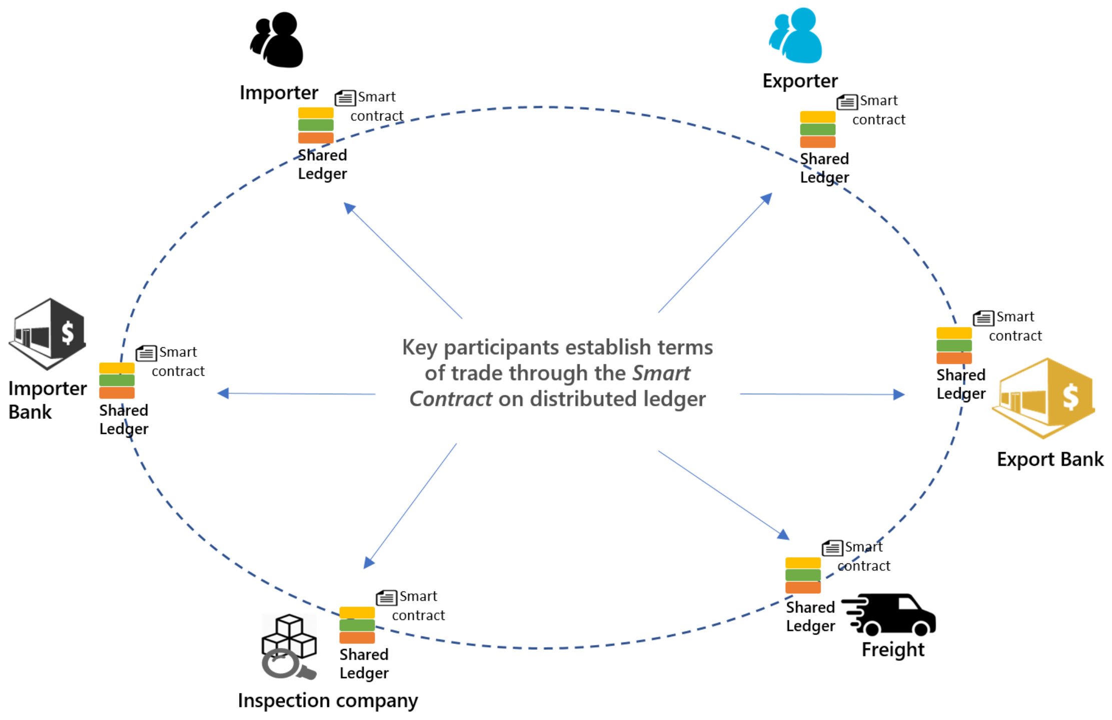
Any business scenario that involves peer-to-peer transactions in a non-trusted environment is a viable candidate for Blockchain technology e.g. *Trade Finance*.

Trade Finance has enabled exchange of goods for centuries. However, the process is prone to delays and frauds. It's largely manual and lacks transparency. It is one of the key business scenario for realizing the benefits of Blockchain, which can help reduce the disputes and errors and bring in transparency by providing a single source of truth.

Note: In this document we do not explain the 'Trade Finance' process in detail. If needed, please refer to the resource below:

https://en.wikipedia.org/wiki/Trade_finance

The diagram below highlights the benefits of using Blockchain for *Trade Finance*



Benefits of using Blockchain/DLT:

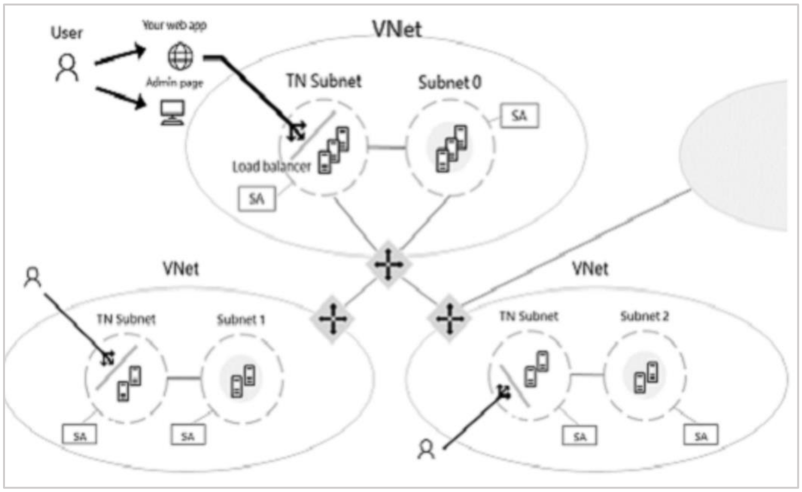
- **Real-time review:** Financial documents linked and accessible through Blockchain are reviewed and approved in real time, reducing the time it takes to initiate shipment
- **Disintermediation:** Banks facilitating trade finance through Blockchain do not require a trusted intermediary to assume risk, eliminating the need for correspondent banks
- **Decentralized contract execution:** As contract terms are met, status is updated on Blockchain in real time, reducing the time and headcount required to monitor the delivery of goods
- **Proof of ownership**: The title available within Blockchain provides transparency into the location and ownership of the goods
- **Automated settlement and reduced transaction fees:** contract terms executed via smart contract eliminate the need for correspondent banks and additional transaction fees

2.2. High-Level Architecture

This section does not provide a step-by-step approach or detailed architecture for implementing the Trade Finance scenario rather the intention is to highlight the products and services needed to enable such a scenario on Microsoft Azure using Corda Enterprise.

The design follows the '*Multiple Organizations, Private Consortium*' approach (refer here). It is a true consortium setup where each organization/party has its own setup of Azure services (e.g. AD tenant etc.), which is provisioned in its own Azure subscription/region. The Corda node running in respective organization/party's setup is enabled for peer-to-peer communication with other Corda nodes within other organization/party's setup.

A conceptual multi-member network architecture is illustrated below (refer here for more details):



Note: Please refer to below article for details about creating a networking topology for a consortium

- https://azure.microsoft.com/en-us/blog/multi-member-consortium-blockchain-networks-on-azure/
- https://blogs.msdn.microsoft.com/azurecat/2017/12/15/designing-scalable-blockchain-networking-topology-in-azure/

The setup within each of the participating organization's subscription, utilizes multiple Azure services, and follows a tiered layout with subnet for each tier (e.g. Application, Web etc.) as illustrated and described below.

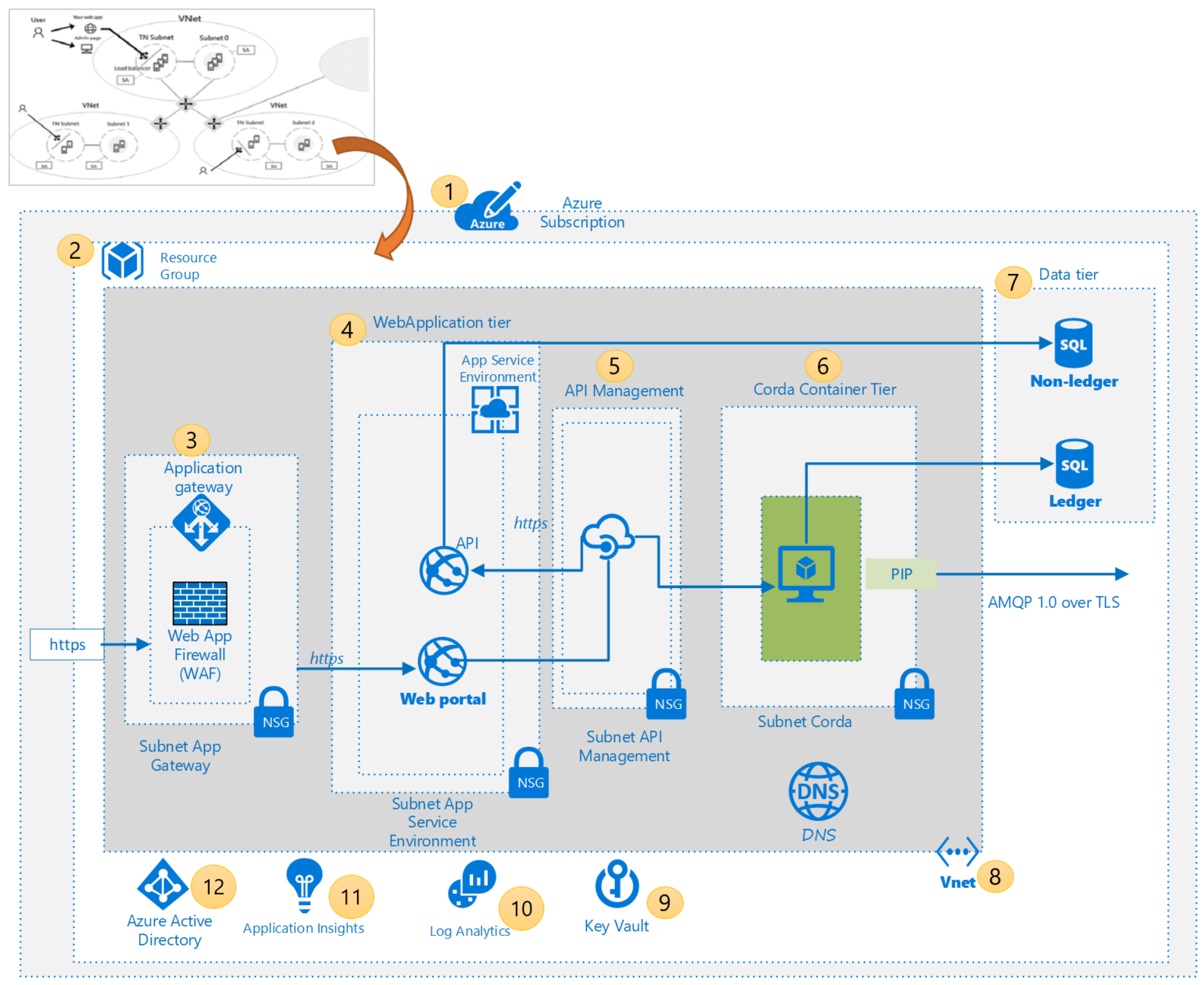


Table below briefly explains the various Azure services along with their usage -

#	Description
1	Azure Subscription governs access to and use of Microsoft Azure services.
2	Resource Group provides a way to provision and manage a collection of Azure services. Please refer here for more detail on Azure Resource Group
3	Application Access Tier constitutes of an Application Gateway. It is a dedicated virtual appliance providing various capabilities like Web Application Firewall(WAF), HTTP load balancing and SSL offloading etc. for the Web Application Tier, which hosts the Web/API apps. Please refer here for more detail on Application Gateway.
4	Web Application Tier hosts the Azure App Services (Web app or API app). Please refer here for more detail on Azure App Service.

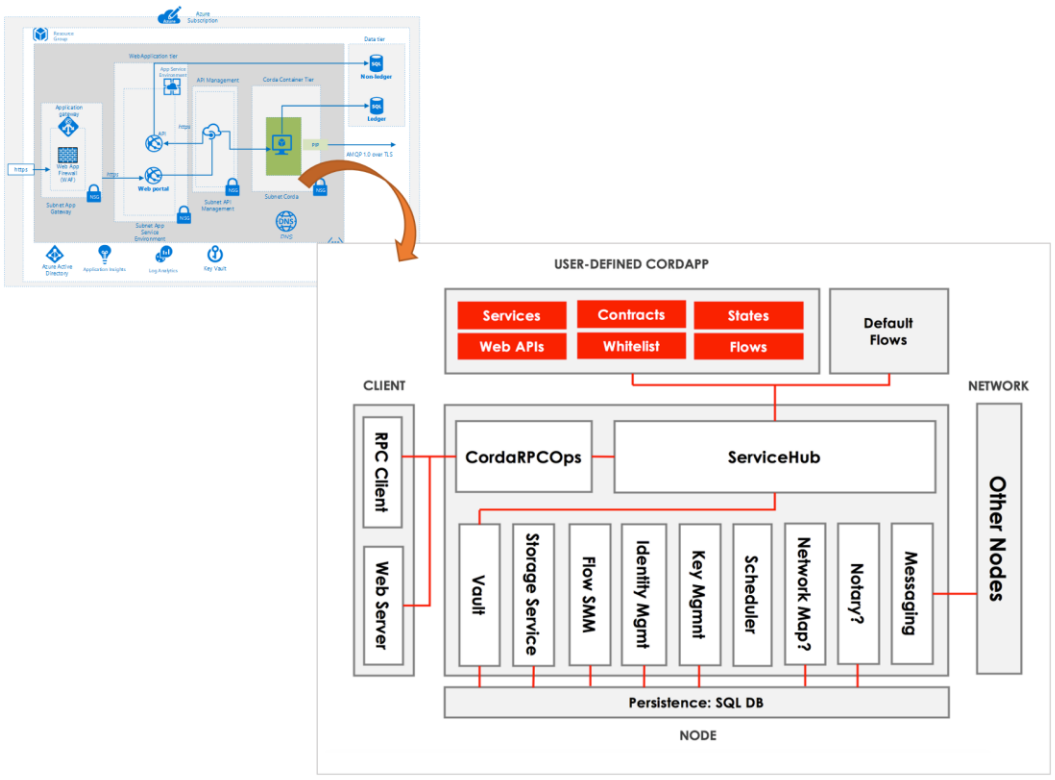
5	API Tier hosts the Azure API Management service, which helps publish APIs providing functions like security, version control, auditing etc. The solution relies on API to connect various components like Corda node, UIs and off-ledger data stores. Please refer here for more detail on Azure API Management.
6	Corda Tier hosts the virtual machines running containerized Corda images using Docker. Please refer here for multiple ways to run Docker in Azure using Virtual Machines.
7	Data Tier hosts Azure SQL Database for Off-Ledger data. Please refer here for more detail on Azure SQL Database service.
8	Virtual Network is a logical isolation of the Azure Corda dedicated to a given subscription enabling Azure resources to securely communicate with each other. Please refer here for more detail on Azure Virtual Network.
9	Azure Key Vault stores the certificates and keys utilized by Apps, Azure Services and Corda network. Please refer here for more detail on Azure Key Vault.
10	Log Analytics provides way to collect telemetry and other data from a variety of sources. Please refer here for more detail on Azure Log Analytics.
11	Application Insights enables instrumentation for the custom code. Please refer here for more detail on Azure Application Insights.
12	Azure Active Directory provides the Identity Management services. Please refer here for more detail on Azure Active Directory.

3. Corda Node Deployment

This section describes automated deployment of Corda nodes using Azure Resource manager (ARM) templates.

3.1. What's a Corda Node?

A Corda network is composed of machines running nodes. A node is a JVM run-time environment running the Corda software. The internals of the node are illustrated below -



Credit: https://docs.corda.net/key-concepts-node.html

The core elements of the architecture are:

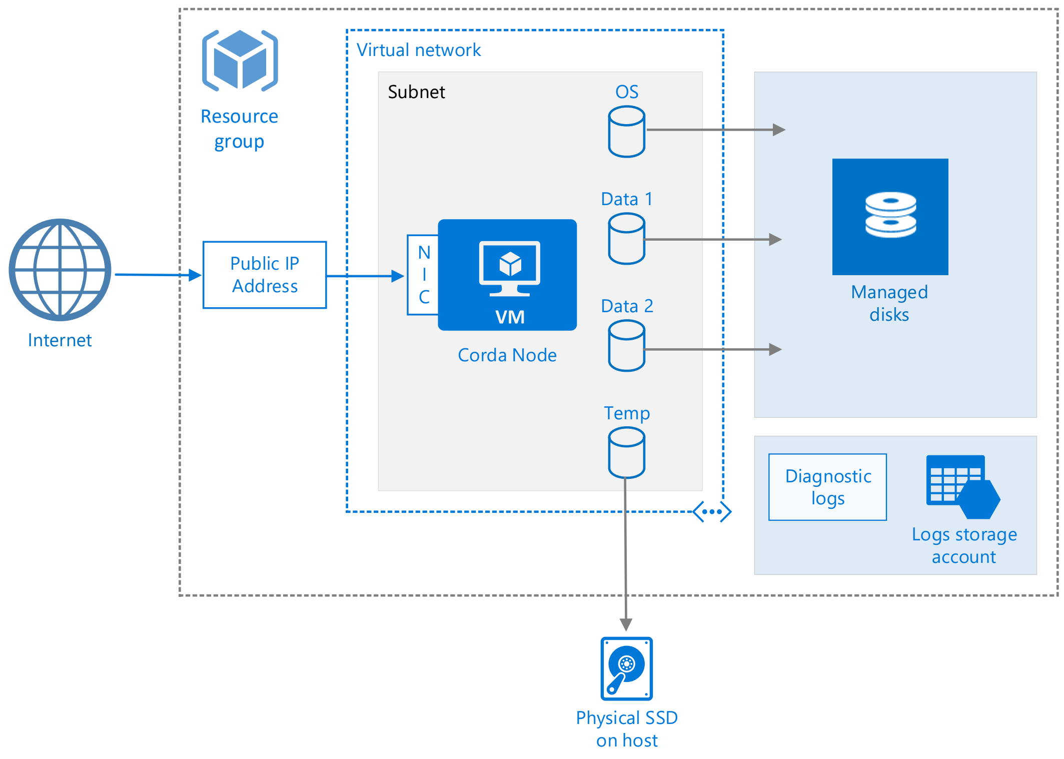
- A persistence layer for storing data
- A network interface for interacting with other nodes
- An RPC interface for interacting with the node's owner
- A service hub for allowing the node's flows to call upon the node's other services
- A CorDapp interface and provider for extending the node by installing CorDapps

Further details can be found at - https://docs.corda.net/key-concepts-node.html

3.2. Which OS to use?

Corda can be deployed on both Windows and Linux. It can also be containerized (Refer https://docs.corda.net/head/deploying-a-node.html for more details)

When using Azure VM, as a best practice we should always use Azure Data Disks and at the Operating System level run the Corda node from the logical data disk which can be configured to aggregate 2 or more disks. (refer diagram below)



In Windows we recommend the usage of Storage Spaces and Linux we have the following guidance available to achieve maximum performance - https://docs.microsoft.com/en-us/azure/virtual-machines/linux/optimization

3.3. Can we use Docker?

Corda Enterprise? has made available a docker image which allows to run inside a container and can be found in the following repository - https://github.com/corda/corda-docker. Also, if we run the Corda node in a container, we should also use volumes which are the preferred mechanism for persisting data generated by and used by Docker containers: https://docs.docker.com/storage/volumes/

3.4. CI/CD for Automated Deployment

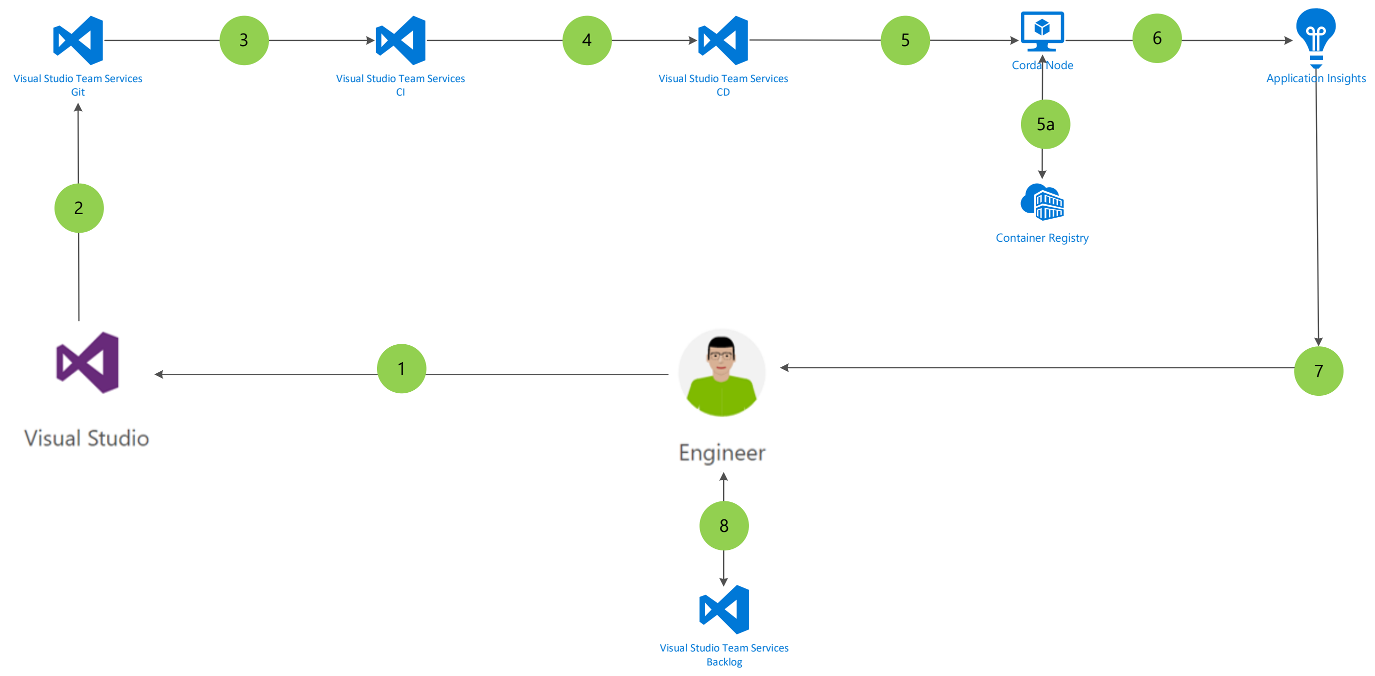
We can use a continuous integration and continuous deployment (CI/CD) pipeline to automate the deployment tasks and push changes to the Corda nodes automatically.

Visual Studio Team Services (VSTS) provides the CI/CD pipeline, starting with a Git repository for managing your application source code and infrastructure code (ARM templates).

Note: Please refer to the resource below for more details on Continuous Integration and Delivery using Visual Studio Team Services

https://www.visualstudio.com/team-services/continuous-integration/

The pipeline can use ARM templates to provision or update the infrastructure as necessary in each subscription, and then deploy the updated build following a workflow as described in the following diagram:


Visual Studio Team Services Git
3
Visual Studio Team Services CI
4
Visual Studio Team Services CD
5
Corda Node
6
Application Insights
5a
Container Registry
2
Visual Studio
1
Engineer
7
8
Visual Studio Team Services Backlog

#	Description
1	Change application source code
2	Commit Application Code and Azure Resource Manager (ARM) Template
3	Continuous integration triggers application build
4	Continuous deployment trigger orchestrates deployment of application artifacts with environment specific parameters
5	Deployment or update of the VM
5a	Deployment or update of the Docker Container which runs the Corda node (either with VSTS or Custom Script Extension). Please refer here for more details on Custom Script Extension.
6	Application Insights collects and analyses health, telemetry, performance and usage data
7	Review health, performance and usage information
8	Update backlog items

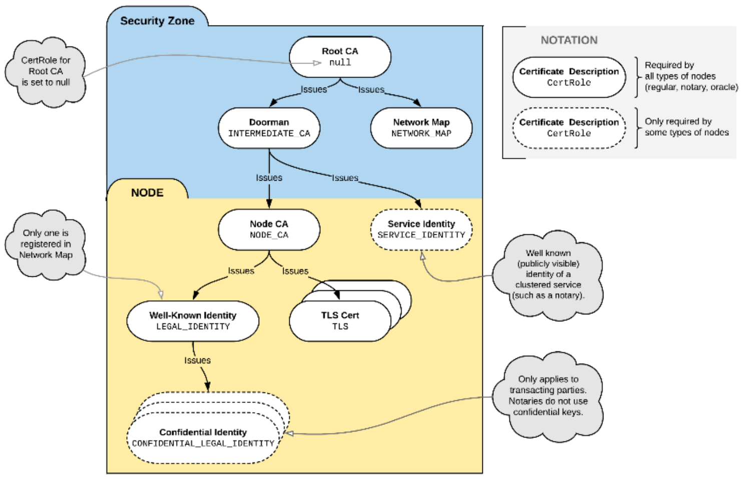
4. Keys Management Using Azure Key Vault

It's critical to be familiar with Corda key/certificate management concepts before understanding the possibilities of integrating with Azure Key Vault.

4.1. Understanding Corda Keys and Certificates

A Corda network has four types of Certificate Authorities (CAs):

- Root Network CA
- Doorman CA
- Node CAs
- Legal Identity CAs


Security Zone
CertRole for Root CA is set to null
Root CA
null
Issues
Issues
Doorman
INTERMEDIATE_CA
Network Map
NETWORK_MAP
NOTATION
Certificate Description
CertRole
Required by all types of nodes (regular, notary, oracle)
Certificate Description
CertRole
Only required by some types of nodes
Issues
Issues
NODE
Only one is registered in Network Map
Node CA
NODE_CA
Service Identity
SERVICE_IDENTITY
Well known (publicly visible) identity of a clustered service (such as a notary).
Issues
Issues
Well-Known Identity
LEGAL_IDENTITY
TLS Cert
TLS
Issues
Only applies to transacting parties. Notaries do not use confidential keys.
Confidential Identity
CONFIDENTIAL_LEGAL_IDENTITY

Corda's X509Utilities (which uses Bouncy Castle) can be used to create public/private keypairs and certificates. Included below are the steps needed to build the Certificate hierarchy –

1. Root Network CA – Generate keypair, create a self-signed certificate for the keypair
2. Doorman CA – Generate keypair, obtain a certificate for the keypair signed with the root network CA key.
3. Node CA – For each node, generate keypair, obtain a certificate for the keypair signed with the doorman CA key

Note: Please refer https://docs.corda.net/permissioning.html for more details.

Now that we understand the Corda certificate hierarchy and generation process, we can move onto understanding how Azure Key Vault (AKV) can be integrated with it.

4.2. Azure Key Vault (AKV) Integration

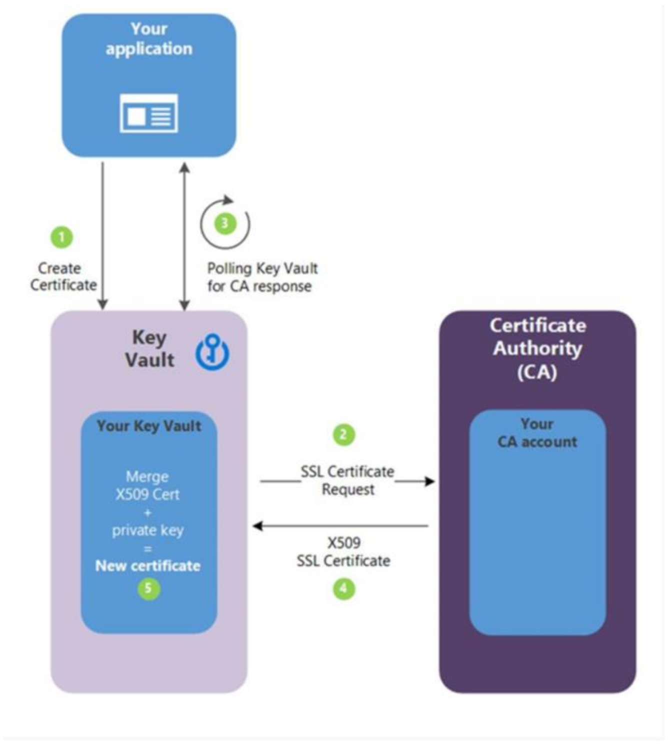
Azure Key Vault (AKV) can be used to store the Keys/Certificates used in a Corda network (please refer ‘Certificate Hierarchy’ in the section above for details). We can also create a certificate using AKV with a known issuer, such as Digicert or GlobalCert (as explained in the next section).

However, it would be challenging to integrate AKV with the existing version of Corda as it requires low-level understanding of the runtime services and enhancements to the source code. This would be easier to implement in the upcoming release of Corda (expected later this year), where the changes needed for the AKV integration would be included at the platform level.

4.2.1. How to create Certificate from within AKV?

The diagram below describes, at a high-level, the certificate creation process involving a given application and AKV (step by step description of the process is available here).

As described in the previous section, it’s expected that Corda’s upcoming release would have the necessary support to integrate with AKV.



5. Securing Corda Network Communication

A Corda network is an authenticated peer-to-peer network of nodes, where each node is a Java Virtual Machine run-time environment hosting Corda services and executing applications. All communication between nodes is direct, with TLS-encrypted messages sent over AMQP/1.0.

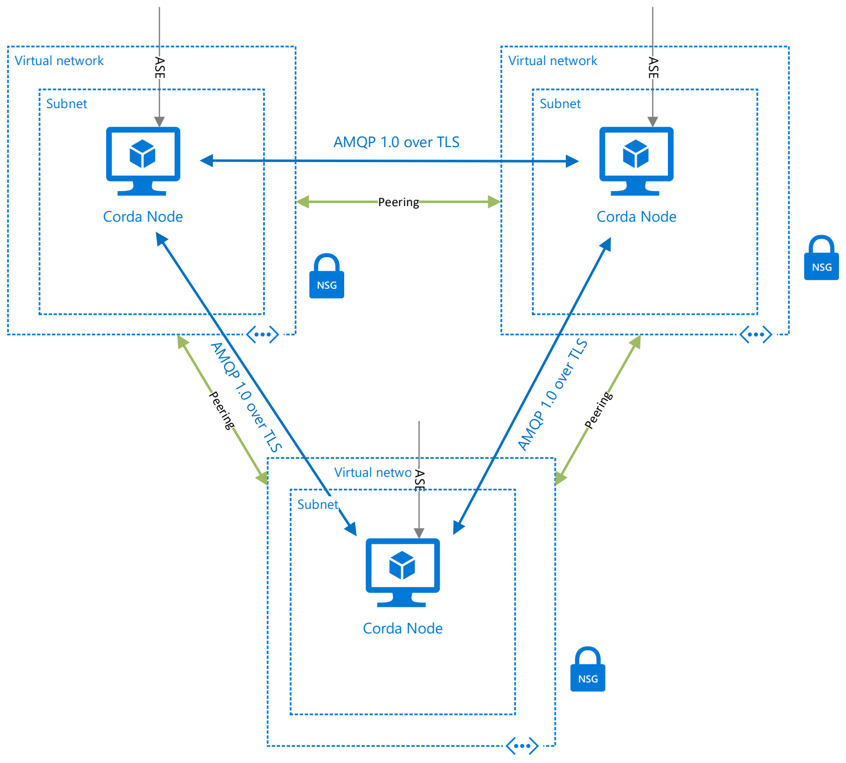
Each Corda network has a *Network Map Service* that publishes the IP addresses through which every node on the network can be reached, along with the identity certificates of those nodes and the services they provide.

More information is available here: https://docs.corda.net/key-concepts-ecosystem.html

On Azure, a Corda network will constitute of corda nodes running within an Azure Virtual Network (VNET) deployed across multiple Azure subscriptions owned by the respective consortium participants. We have multiple options to interconnect them. These options take in account subscription limits and best practices for each of the connectivity methods used to connect corda network nodes across the different VNETs.

5.1. Solution #1 - VNET peering

The first option to interconnect a Corda network would be using VNET peering, which would allow for higher number of transactions along with enabling better performance. VNET peering is illustrated in the diagram below -



Currently, there are limits which need to be taken into account when it comes to the usage of VNET peering per region. Nonetheless, if the goal performance and keeping configuration to a minimum this is the best option. The limits per subscription for VNET peering are documented here - https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits

In terms of scalability in the future when a Corda network grows, the best option is *either a mesh topology or Service Chaining*: https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-peering-overview

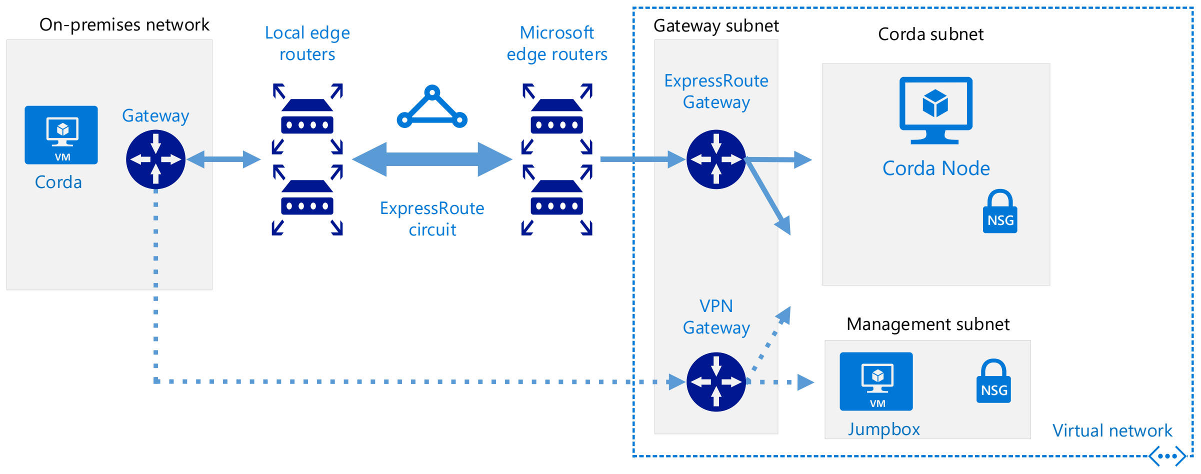
In blockchain, we'll typically have each member belonging to a different company, each with its own subscription, Identity management requirements and restrictions. So peering works when all subscriptions are associated with the same Azure AD tenant. If this is a constraint, then we should look at solution #2.

5.2. Solution #2 - Site-to-Site VPN

In case each member of the consortium wants to have its own Azure AD tenant then the next solution is setting up the network through either a *VPN Gateway* or an Network Virtualization Appliance (*NVA)* which would establish the connections between both networks.

Please refer - https://docs.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-about-vpngateways?toc=%2fazure%2fvirtual-network%2ftoc.json#V2V

Additional possibility which comes with Site-to-Site VPN is the option of having a hybrid environment, where one could have a Corda node running on premises as illustrated below -



5.3. Solution #3 - Nginx with DDoS Standard

The third solution is when customers have a public IP address which is associated with the Corda node and we need to secure this endpoint with the means which we have available for Layers 3, 4. The Azure DDoS Standard Protection service protects your application from a comprehensive set of network layer (Layer 3, 4) attacks.

This service mitigates the following types of attacks:

- Volumetric attacks
- Protocol attacks
- Application layer attacks

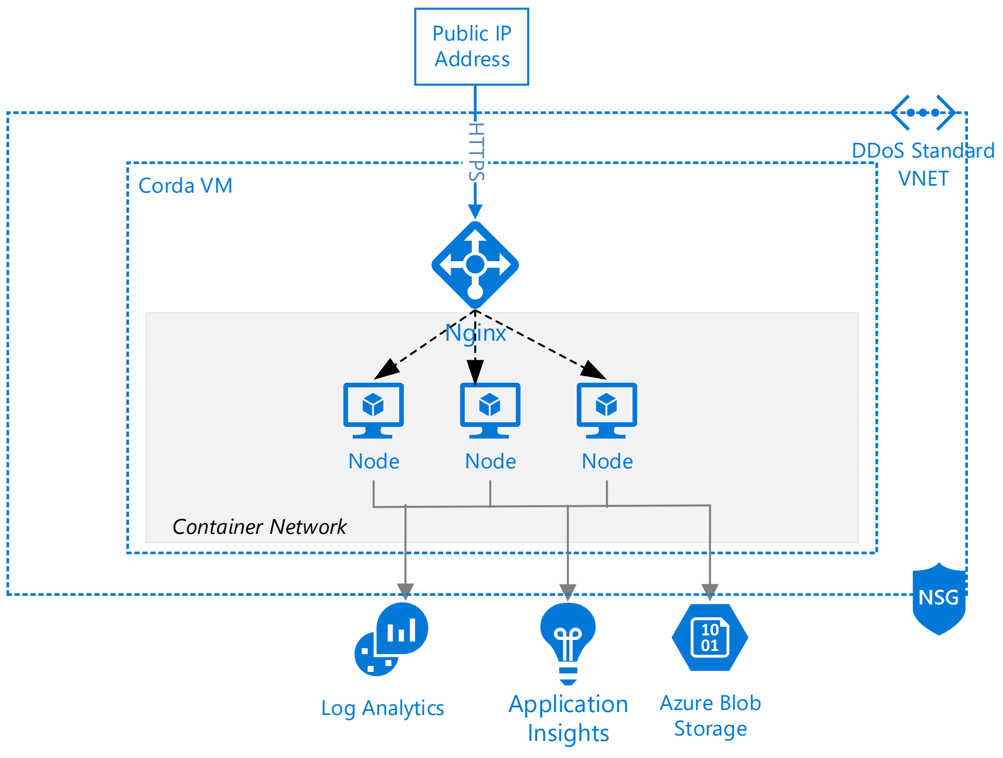
More information can be found here:

https://docs.microsoft.com/en-us/azure/virtual-network/ddos-protection-overview

We still need to protect the traffic which flows at Layer 7 though.

One option would be to go with an approach where we can run the Corda node application coupled with Nginx. In which we would terminate the SSL tunnel:

https://docs.microsoft.com/en-us/azure/virtual-machines/linux/tutorial-secure-web-server



5.4. How to Secure Access to the Corda Node

When securing access, we should consider using a layered approach in which we include a remote access option and use Azure logon restrictions to constrain source IP addresses such as Just-in-Time(JIT) which is provided by Azure Security Center Standard.

Note:

Just in time virtual machine (VM) access can be used to lock down inbound traffic to Corda nodes VMs, reducing exposure to attacks while providing easy access to connect to VMs when needed.

More information here:

https://docs.microsoft.com/en-us/azure/security/azure-security-management

We also absolutely need to think about the other mechanisms which are available to help secure client connections to the virtual network.

Take for instance a stand-alone hardened workstation that does not connect to Azure through an RD Gateway should use the SSTP-based point-to-site VPN to create the initial connection to the Azure Virtual Network, and then establish the remote connection to the individual nodes from with the VPN tunnel.

https://docs.microsoft.com/en-us/azure/security/azure-security-management#stand-alone-hardened-workstation-for-management

6. Monitoring and Telemetry

For monitoring and telemetry related requirements on Microsoft Azure, it's recommended to utilize a combination of Operations Management Suite(OMS), Log Analytics, Application Insights and Azure Monitor.

For a conceptual view of the components that work together to provide monitoring of Azure resources, please refer - https://docs.microsoft.com/en-us/azure/monitoring-and-diagnostics/monitoring-overview

6.1. What's provided out-of-the-box with Corda installation?

Corda node exports various metrics via the industry-standard JMX Infrastructure. However, as highlighted here, as of Corda M11, Java serialization in the Corda node has been restricted, meaning MBeans access via the JMX port will no longer work and Jolokia is one of the recommended options.

Note: JMX (Java Management eXtensions) is a technology for monitoring and managing Java applications. JMX uses objects called MBeans (Managed Beans) to expose data and resources from the application.

Jolokia allows one to access the raw data and operations without connecting to the JMX port directly.

For monitoring purposes, Corda core node exposes the specific metrics:

- net.corda:name=Attachments
- net.corda:name=Started,type=Flows
- net.corda:name=Finished,type=Flows
- net.corda:name=InFlight,type=Flows
- net.corda:name=Checkpointing Rate,type=Flows

The following JMX statistics are also exported:

- Apache Artemis metrics: queue information for P2P and RPC services
- JVM statistics: classloading, garbage collection, memory, runtime, threading, operating system
- Hibernate statistics (only when node is started-up in devMode due to to expensive run-time costs)

More information can be found here:
https://docs.corda.net/node-administration.html
https://github.com/corda/corda/blob/master/docs/source/node-administration.rst

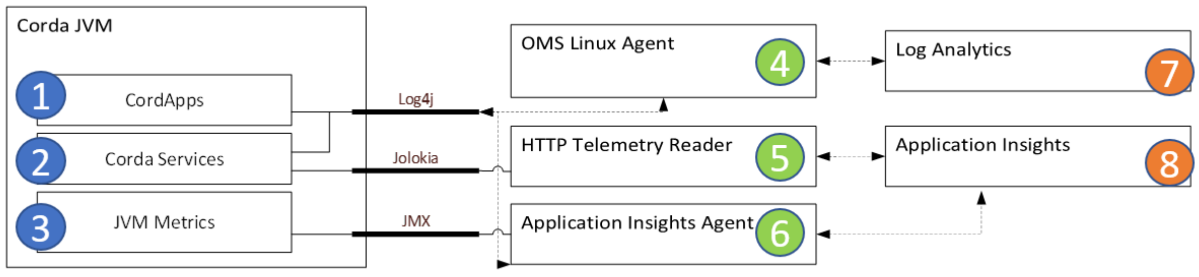
Corda logging is standard log4j2 and can be configured to trace additional information. These logs are by default redirected to files in NODE_DIRECTORY/logs/ but they can also be sent to syslog.

Telemetry from a Corda solution is available at two levels

- Corda core Services (e.g. Network Map etc.)
- Corda Distributed Applications (CorDapps)

6.2. How to manage logs from Corda using Azure Services?

The following diagram illustrates how we can manage the logs generated by Corda using Azure Services:



From a JVM perspective we have 3 sources of logs that can feed into different components, which we can configure to trace telemetry into either Log Analytics or Application Insights.

Inside the Virtual Machine or Container which runs the CorDapps, Corda Services and Java Virtual Machine (JVM) we have three points of data collection which we need to look into:

1. **CorDapps** – here we will need to instrument the code with the Application Insights SDK and also configure Log4j2. This involves user-defined CorDapps such as:

    - Services
    - Contracts
    - States
    - Web APIs
    - Flows

    As CorDapps involve custom code we can use Application Insights for instrumentation and telemetry. Please refer to the article below for more details - https://docs.microsoft.com/en-us/azure/application-insights/app-insights-java-trace-logs

2. **Corda Services** - this refers to Corda core services which are highlighted in the internal architecture outlined in section 3.1. We can leverage two options for tracing in this source of logging:

    - Log4j2 which we configure to trace into the OMS Linux Agent. The following location has configuration specific guidance: https://docs.microsoft.com/en-us/azure/application-insights/app-insights-java-trace-logs

    - JMX metrics exposed by Corda and send the required telemetry through Jolokia – the node exports them over HTTP on the /monitoring/json HTTP endpoint, using a program called Jolokia. Jolokia defines the JSON and REST formats for accessing MBeans and provides client libraries to work with that protocol as well.

3. **Java Virtual Machine Metrics** – these are JVM specific metrics which are exposed through JMX and can be consumed by the Application Insights Agent. In the following location we have guidance on how to configure the Application Insights Agent to collect this tracing: https://docs.microsoft.com/en-us/azure/application-insights/app-insights-java-eclipse

Having identified the sources of telemetry, we now have three collectors of telemetry as highlighted in the diagram above which are:

1. **Linux Operations Management Studio (OMS) agent**

    We can use Linux Operations Management Studio (OMS) agent to push logs from Corda core services to Log Analytics. Included below are articles to understand the agent and associated process better -

    - Following link has the description of how the buffers of the OMS agent works on Linux and which parameters we can tweak in order to adapt the ingestion rate with what you expect from the Corda node: https://docs.fluentd.org/v0.12/articles/buffer-plugin-overview

    - The configuration file of the agent and more documentation can be found here - https://github.com/Microsoft/OMS-Agent-for-Linux

    - The Linux OMS agent is based on fluentD, so we can make sure that the right format is displayed in Log Analytics by applying regex patterns which fit the desired view: https://docs.fluentd.org/v0.12/articles/parser_syslog#message_format

    - For quick reference, the following would get the desired format: https://github.com/Microsoft/OMS-Agent-for-Linux/blob/master/docs/Security-Events-Preview-Configuration.md

2. **HTTP Telemetry reader** – this is where we can consume the information which is exported by the Corda services.

   With additional instrumented code (use app insights agent SDK and include link), we can consume the information from Jolokia over HTTP: https://github.com/Microsoft/ApplicationInsights-Java

   One potential implementation for this consumer of logs would be using an instance of TelemetryClient: https://docs.microsoft.com/en-us/azure/application-insights/app-insights-api-custom-events-metrics

3. **Application Insights Agent** – provide full tracing facilities where we can collect telemetry from instrumented code at the CordApps level and also with additional configuration we can trace JMX counters which relate to the Java Virtual Machine (to name a few: Heap Memory Usage, CPU, Garbage Collection).

   This is one option that we can use to perform active probing when monitoring a Corda node.

   More details here: https://docs.microsoft.com/en-us/azure/application-insights/app-insights-java-get-started

Finally, we have two points of telemetry ingestion which are Log Analytics and Application Insights.

With these two points, telemetry from both the Corda core services and Corda Distributed Applications (CorDapps) can be centralized into a single dashboard. However, that dashboard would be specific to the respective organization/party subscription, as the Application-Insight/Log-Analytics account is at the subscription level.

A cross-subscription dashboard will need a custom implementation, where the logs will have to be shipped over to a centralized location, collated and rendered on a single centralized dashboard.

1. **Log Analytics** - If we're looking to Log Analytics as source of Alerts for when we're monitoring the Corda node, we should be aware that data ingestion might have associated latency or simply take time to be consumed.

   Following are the items to consider when going the Log Analytics route -

   - *Data collection frequency*: OMS has a solution for monitoring containers, where the data collection frequency is 3 minutes - https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-add-solutions#data-collection-details. This will be one of the points to keep in mind when looking into the Log Analytics workspace and query results.

   - *Configuration of buffering in the Linux agent:* Given that OMS agent is based on fluentD, here we have the description of how the buffers of the OMS agent works on Linux and which parameters we can tweak in order to adapt the ingestion rate with what you expect from the Corda node application: https://docs.fluentd.org/v0.12/articles/buffer-plugin-overview

     You can find the configuration file of the agent and more documentation here - https://github.com/Microsoft/OMS-Agent-for-Linux

   - *Ingestion of data:* typically takes a few minutes to an hour. After the service processes the data, you can view it in Log Analytics. More information may be found here - https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-add-solutions#data-collection-details

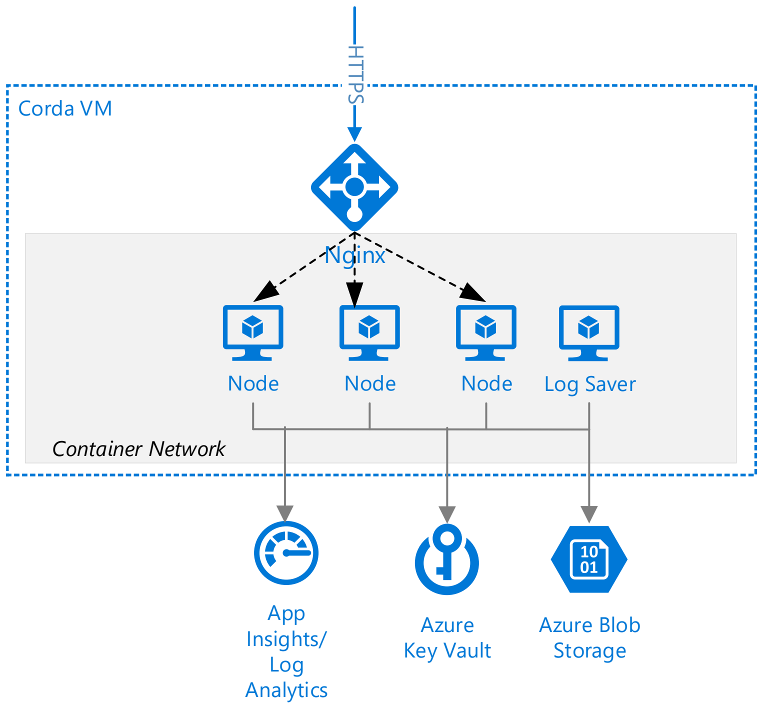
     More specifically, there's a delay when ingesting data which is a combination of data collection frequency and the time that data takes to be ingested into Log Analytics -
     https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-alerts

2. **Application Insights** – when leveraging Application Insights completely, we will be able to automatically detect performance anomalies, correlate telemetry and have the ability to trace the request flow from client (application, browser) through all the services involved in the processing at the subscription level (i.e. between the web application which will interact with the CordApps running in the Corda node).

   More details here: https://docs.microsoft.com/en-us/azure/application-insights/app-insights-overview

6.3. Introducing Sidecar Pattern for telemetry

Given that we can have multiple roles running in the same node, this allows room to introduce a known pattern which we can use for telemetry purposes as shown below:



The pattern which can use to monitor the Corda node application is a *sidecar* pattern. We can achieve this by pairing the Corda node application, which can run either in an IaaS Virtual Machine instance or in a container, next to another instance where we run the "log saver agent" which will then send syslog information into Log Analytics.

More information about this pattern can be found here:
https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar

This way we'll benefit from the separation of both units and fully utilize the resources that each may need without interfering. Configuring it this way, we'll also have one container packaged just for logging which makes it easier to divide configuration tasks such as parsing rules which need to be applied for the OMS agent, redeploy it if needed and without affecting Corda.

In the event of a failure in the log saver container, we'll still have the application healthy.

Additionally, as we scale more containers per different role we'll be able to have a single container which can be used by all the other containers for log saving.