

---

# **Blockchain Automation Framework Documentation**

*Release 0.4.0*

**Blockchain Automation Framework**

**Sep 25, 2020**



---

## Table of Contents:

---

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Release notes</b>	<b>5</b>
<b>3</b>	<b>Key Concepts</b>	<b>7</b>
<b>4</b>	<b>Getting Started</b>	<b>13</b>
<b>5</b>	<b>Operations Guides</b>	<b>15</b>
<b>6</b>	<b>Developer Guide</b>	<b>73</b>
<b>7</b>	<b>Sample Usage</b>	<b>173</b>
<b>8</b>	<b>BAF current roadmap</b>	<b>177</b>
<b>9</b>	<b>Architecture Reference</b>	<b>181</b>
<b>10</b>	<b>Commands Reference</b>	<b>211</b>
<b>11</b>	<b>Frequently Asked Questions</b>	<b>217</b>
<b>12</b>	<b>Glossary</b>	<b>223</b>
<b>13</b>	<b>Contributing</b>	<b>237</b>
<b>14</b>	<b>Maintainers for the Blockchain Automation Framework</b>	<b>239</b>



The Blockchain Automation Framework is an accelerator for delivering production-ready blockchain solutions, enabling teams to deliver without the need to architect the solution from scratch.

**What is the Blockchain Automation Framework?**

The Blockchain Automation Framework is an automation framework for delivering consistent production ready DLT networks on cloud based infrastructures.

The Blockchain Automation Framework (BAF) provides 3 key features:

- **Security:** BAF provides a secure environment for DLT development. BAF has best practices of key management and other security features available by default.
- **Scalability:** BAF has a scalable network implementation, a user can easily scale the environment and resources according to his/her needs.
- **Acceleration:** BAF will help in providing a blockchain solution that drives acceleration up to deployment providing an opportunity to participate in those deliveries and drive more services.

It is an accelerator for all the developers to be able to use a DLT network right away. So with the Blockchain Automation Framework, users are able to create a DLT environment and know that it is something that will continue to be used as project management.



# CHAPTER 1

---

## Introduction

---

At its core, Blockchain is a new type of data system that maintains and records data in a way that allows multiple stakeholders to confidently share access to the same data and information. A blockchain is a type of Distributed Ledger Technology (DLT), meaning it is a data ledger that is shared by multiple entities operating on a distributed network.

This technology operates by recording and storing every transaction across the network in a cryptographically linked block structure that is replicated across network participants. Every time a new data block is created, it is appended to the end of the existing chain formed by all previous transactions, thus creating a chain of blocks called the blockchain. This blockchain format contains records of all transactions and data starting from the inception of that data structure.

Setting up a new DLT/Blockchain network or maintaining an existing DLT/Blockchain network in a production-scale environment is not straightforward. For the existing DLT/Blockchain platforms, each has its own architecture, which means the same way of setting up one DLT/Blockchain network cannot be applied to others.

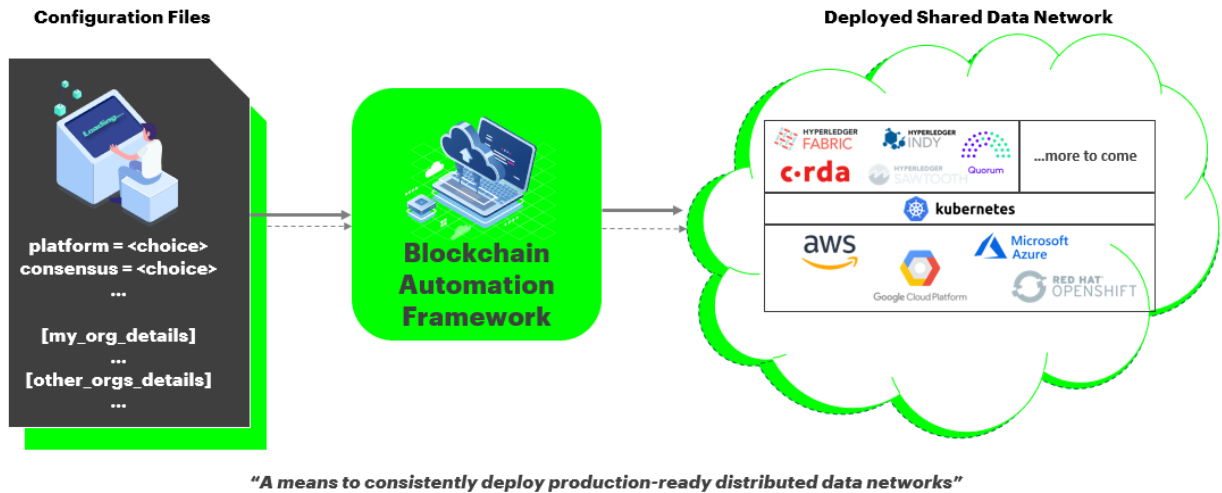
Therefore, when blockchain developers are asked to use an unfamiliar DLT/Blockchain platform, it requires a great deal of effort for even experienced technicians to properly setup the DLT/Blockchain network. This is especially true in large-scale production projects across heterogeneous corporate environments which require other key aspects such as security and service availability.

Being aware of the potential difficulty and complexity of getting a production-scale DLT/Blockchain network ready, cloud vendors such as AWS and Azure have provisioned their own managed Blockchain services (aka Blockchain as a Service or BaaS) to help alleviate various pain-points during the process. However, limitations can still be identified in their BaaS solutions, e.g. limited network size, locked to all nodes on a single cloud provider, or limited choice of DLT/Blockchain platform, etc.

## 1.1 The Blockchain Automation Framework (BAF) Platform

The objective of BAF is to provide a consistent means by which developers can deploy production-ready distributed networks across public and private cloud providers. This enables developers to focus on building business applications quickly, knowing the framework upon which they are building can be adopted by an enterprise IT production operations organization. BAF is not intended solely to quickly provision development environments which can be done more efficiently with other projects/scripts. Likewise, Blockchain Automation Framework is not intended to

replace BaaS offerings in the market, but instead, BAF is an alternative when existing BaaS offerings do not support a consortium's current set of requirements.



## 1.2 How is it different from other BaaS?

- The Blockchain Automation Framework deployment scripts can be reused across cloud providers like AWS, Azure, GCP, and OpenShift
- Can deploy networks and smart contracts across different DLT/Blockchain platforms
- Supports heterogeneous deployments in a multi-cloud, multi-owner model where each node is completely owned and managed by separate organizations
- Bring Your Own Infrastructure (BYOI) - You provide GIT, Kubernetes cluster(s), and Hashicorp Vault services provisioned to meet your specific requirements and enterprise standards
- No network size limit
- Specify only the number of organizations and the number of nodes per organization in a [network.yaml file](#) uniquely designed in the Blockchain Automation Framework for a new DLT/Blockchain network set-up and its future maintenance
- Provides a sample supply chain application which runs on multiple DLT/Blockchain platforms that can be used as a reference pattern for how to safely abstract application logic from the underlying DLT/Blockchain platform

### 1.2.1 What next?

We have been actively searching for partners who need and understand the value of Blockchain Automation Framework, who share the vision of building and owning well architect-ed solutions . We wish to work together so as to identify the market needs for those partners, to further reduce the barriers in adoption.

## CHAPTER 2

---

### Release notes

---

Release notes have been moved to GitHub [here](#).



## CHAPTER 3

---

### Key Concepts

---

This section introduces the key concepts along with their features used within the Blockchain Automation Framework. This section works as step one that will pave the way for new users to understand the key conceptual building blocks used in the Blockchain Automation Framework's architecture design.

### 3.1 Ansible

**Ansible** is an automation command line tool that helps IT technicians easily achieve system configuration, software deployment and other complex tasks in orchestration.

Ansible provisions several types of command line tools such as **ansible**, **ansible-playbook** and **ansible-galaxy** etc. Each serves different scenarios so that a user can choose the most appropriate one or more to be adopted in the chosen scenario(s).

Below gives a simple description of the three mentioned above, and a user can use the link to find more information for each of them.

- **ansible**: it is the simplistic command line tool that enables a user to quickly achieve simple IT tasks, e.g. list one or more local/remote machines' information.
- **ansible-playbook**: it is an advanced command line that will run one or more Ansible playbooks (i.e. YAML files that have all the steps configured to achieve one or more complex tasks). Ansible **roles** are defined to group relevant configurations together that can be reusable in multi playbooks.
- **ansible-galaxy**: it is an advanced command line that can run existing Ansible roles predefined by other users in the Ansible community.

The Blockchain Automation Framework extensively uses Ansible playbooks along with roles to spin up a DLT/Blockchain network. For instance, to issue certificates for each node in the DLT/Blockchain network, and then put the certificates to HashiCorp **Vaults**. In the Blockchain Automation Framework, there are different Ansible playbooks being designed, and the key player that makes the whole DLT/Blockchain network set-up to happen automatically is the roles defined in the playbooks following a specific order.

## 3.2 Kubernetes Services

### 3.2.1 Container

A [Docker Container](#) is an ephemeral running process that have all the necessary package dependencies within it. It differentiates from a [Docker Image](#) that is a multi-layered file. A container is much more light-weighted, standalone and reusable compared to a Virtual Machine (VM).

### 3.2.2 Cluster

A cluster of containers is grouped by one or more running containers serving different purposes with their duplicates that can ensure high availability of services. One example of a cluster is [Docker Swarm](#).

### 3.2.3 Kubernetes

[Kubernetes](#) (K8s) is an open-source system for automating deployment, scaling and maintaining containerized applications. Kubernetes provisions more advanced configurations and features to set up a cluster compared to Docker Swarm, which make it a very strong candidate in any production-scale environments.

### 3.2.4 Managed Kubernetes Services

The open-source K8s services requires technicians to set up an underlying infrastructure and all initial K8s clusters, but the setting-up process is normally time-consuming and error-prone. This is why K8s is well known for its deep learning curves. To alleviate this complex process for users, many Cloud service providers such as [AWS](#), [Azure](#) and [GCP](#) have provisioned their own Managed K8s Services.

The Blockchain Automation Framework leverages Kubernetes's various features for deploying a DLT/Blockchain network along with other required services in one or more K8s clusters. All the current functions have been tested on Amazon K8s Services (AKS) as a managed K8s service, but in theory they should work on a non-managed K8s service as well.

### 3.2.5 Ambassador

[Ambassador](#) is an open-source microservices API gateway designed for K8s.

The Blockchain Automation Framework uses Ambassador to route traffic amongst multiple K8s clusters. For each K8s cluster, an Ambassador Loadbalancer Service will be created to sit inside it. A user has to manually use a DNS server (e.g. AWS Route53) to map the public IP of the Ambassador Service to a DNS name for each cluster. Optionally, you can configure [External-DNS](#) on the cluster and map the routes automatically. Automatic updation of routes via External DNS is supported from BAF 0.3.0.0 onwards.

A simplistic view of how Ambassador works is as follows: If a pod in Cluster1 wants to reach a target pod in Cluster2, it will just use the Domain address or IP in Cluster2 and then Cluster2 Ambassador will route the traffic to the target pod in Cluster2.

---

**NOTE:** If only one cluster is used in a DLT/Blockchain network, Ambassador may not be needed, but it will still be installed (if chosen).

---

### 3.2.6 HAProxy Ingress

[HAProxy Ingress](#) is another way of routing traffic from outside your cluster to services within the cluster. This is implemented in BAF Fabric from Release 0.3.0.0 onwards as we were unable to configure Ambassador to do ssl-passthrough for GRPC.

in BAF, HAProxy Ingress does the same thing as Ambassador does i.e. it routes traffic amongst multiple K8s clusters. For each K8s cluster, an HAProxy Ingress Loadbalancer Service will be created to sit inside it. A user has to manually use a DNS server (e.g. AWS Route53) to map the public IP of the HAProxy Service to a DNS name for each cluster. Optionally, you can configure [External-DNS](#) on the cluster and map the routes automatically. Automatic updation of routes via External DNS is supported from BAF 0.3.0.0 onwards.

---

**NOTE:** If only one cluster is used in a DLT/Blockchain network, HAProxy may not be needed, but it will still be installed (if chosen).

## 3.3 Helm

Essentially, [Helm](#) is a package manager for K8s. Helm Charts are configuration files designed for K8s to help define, install and upgrade complex K8s applications.

Helm brings below features:

1. Predictable deployments.
2. Maintains “Bill of Materials” of all the pods that work together to deliver the application.
3. Keeps (forces) a team to stay synchronised.
4. Strong version control.
5. Easier testing and QA.
6. Rollbacks on an application level, not just an one-off pod level.

The Blockchain Automation Framework uses Helm Charts for designing and configuring the architecture of each DLT/Blockchain platform for its own network set-up.

## 3.4 HashiCorp Vault

[HashiCorp Vault](#) provisions a secure approach to store and gain secret information such as tokens, passwords and certificates.

The Blockchain Automation Framework relies on Vaults for managing certificates used in each node of a DLT/Blockchain network during the lifecycle of a deployment, and it is a prerequisite that the Vault is installed and unsealed prior to deployment of a DLT/Blockchain network.

### 3.4.1 Installation

There are two approaches to installing Vault:

- Using a [precompiled binary](#)
- Installing from [source](#)

Downloading a precompiled binary is easiest and provides downloads over TLS along with SHA256 sums to verify the binary. Hashicorp also distributes a PGP signature with the SHA256 sums that should be verified.

### 3.4.2 Securing RPC Communication with TLS Encryption

Securing your cluster with TLS encryption is an important step for production deployments. The recommended tool for vault certificate management is Consul. Hashicorp Consul is a networking tool that provides a fully featured service-mesh control plane, service discovery, configuration, and segmentation.

Consul supports using TLS to verify the authenticity of servers and clients. To enable TLS, Consul requires that all servers have certificates that are signed by a single Certificate Authority (CA). Clients should also have certificates that are authenticated with the same CA.

After generating the necessary client and server certificates, the `values.yaml` file `tls` field can be populated with the `ca.cert` certificates. Populating this field will enable or disable TLS for vault communication if a value present.

The latest documentation on generating tls material with consul can be found at: <https://learn.hashicorp.com/consul/security-networking/certificates>

## 3.5 GitOps

GitOps introduces an approach that can make K8s cluster management easier and also guarantee the latest application delivery is on time.

The Blockchain Automation Framework uses Weavework's Flux for the implementation of GitOps and executes an Ansible role called *setup/flux* defined in its GitHub repo that will

- Scan for existing SSH Hosts
- Install Tiller on cluster
- Authorize client machine as per `kube.yaml`
- Add weavework flux repository in helm local repository
- Install flux

## 3.6 The Blockchain Automation Framework's Features

### 3.6.1 Multi-Cloud service providers support

The Blockchain Automation Framework's scripts do not stick to any one of the Cloud service provider. On the contrary, they can be used on any Cloud platform as long as all the *prerequisites* are met.

### 3.6.2 Multi-DLT/Blockchain platforms support

The Blockchain Automation Framework supports an environment of multi-clusters for the spin-up of a DLT/Blockchain network (e.g. Hyperledger Fabric or R3 Corda). Regardless of unique components (e.g. channels and orderers in Fabric, and Doorman, Notary in Corda) designed in each platform which makes the DLT/Blockchain ecosystems be heterogeneous, the Blockchain Automation Framework does remove this complexity and challenge by leveraging a uniquely-designed *network.yaml* file, which enables set-up of a DLT/Blockchain network on either platform to be consistent.

### **3.6.3 No dependency on managed K8s services**

Setting up a DLT network does not depend on a managed K8s services, which means non-managed K8s clusters can be used to make a DLT network set-up happen.

### **3.6.4 One touch/command deployment**

A single Ansible playbook called `site.yaml` can spin up an entire DLT network and a substantial amount of time can be reduced which is involved in configuring and managing the network components of a Corda or Fabric DLT network.

### **3.6.5 Security through Vault**

HashiCorp Vault is used to provide identity-based security. When it comes to managing secrets with machines in a multi-cloud environment, the dynamic nature of HashiCorp Vault becomes very useful. Vault enables the Blockchain Automation Framework to securely store and tightly control access to tokens, passwords, certificates, and encryption keys for protecting machines, applications, and sensitive data.

### **3.6.6 Sharing a Network.yaml file without disclosing any confidentiality**

The Blockchain Automation Framework allows an organization to use a configured `network.yaml` file to set up an initial DLT/Blockchain network and a first node in the network, and allows this file to be shared by new organizations that will have to join this DLT/Blockchain network to reuse this `network.yaml` file, but without revealing any confidential data of the first organization.



Before we begin, if you haven't already done so, you may wish to check that you have all the *prerequisites* installed on the platform(s) on which you'll be deploying blockchain networks from and/or operating the Blockchain Automation Framework.

Once you have the prerequisites installed, you are ready to fork this repository and start using the Blockchain Automation Framework.

### 4.1 Configure Prerequisites

After installation of the *prerequisites*, some of them will need to be configured as per the Blockchain Automation Framework. Follow *these instructions* to configure the pre-requisites and setting up of your environment.

### 4.2 Update Configuration File

Once all the prerequisites have been configured, it is time to update the Blockchain Automation Framework configuration file. Depending on your platform of choice, there can be some differences in the configuration file. Please follow platform specific links below to learn more on updating the configuration file.

- *R3 Corda Configuration File*
- *Hyperledger Fabric Configuration File*
- *Hyperledger Indy Configuration File*
- *Quorum Configuration File*

### 4.3 Deploy the Network

After the configuration file is updated and saved on the **Ansible Controller**, run the provisioning script to deploy the network using the following command.

```
# go to blockchain-automation-framework
cd blockchain-automation-framework
# Run the provisioning scripts
ansible-playbook platforms/shared/configuration/site.yaml -e "@/path/to/network.yaml"
```

For more detailed instructions to set up a network, read *Setting up a DLT/Blockchain network*.

This section defines the pre-requisites installation and steps for setup of a DLT network. If this is your first time, do refer Key-Concepts, Getting-Started and Architecture-References before moving ahead.

## 5.1 Install Pre-requisites

Before we begin, if you haven't already done so, you may wish to check that you have all the prerequisites below installed on the platform(s) on which you'll be deploying blockchain networks from and/or operating the Blockchain Automation Framework.

### 5.1.1 Git Repository

As you may have read in the *key concepts*, the Blockchain Automation Framework (BAF) uses GitOps method for deployment to Kubernetes clusters. So, a Git repository is needed for BAF (this can be a [GitHub](#) repository as well). Fork or import the [BAF GitHub repo](#) to this Git repository.

The Operator should have a user created on this repo with full access to the Git Repository.

### 5.1.2 Kubernetes

The Blockchain Automation Framework (BAF) deploys the DLT/Blockchain network on [Kubernetes](#) clusters; so to use BAF, at least one Kubernetes cluster should be available. BAF recommends one Kubernetes cluster per organization for production-ready projects. Also, a user needs to make sure that the Kubernetes clusters can support the number of pods and persistent volumes that will be created by BAF.

---

### 5.1.3 NOTE: BAF DLT platforms are tested on Kubernetes Cluster of AWS instances with support on both Kubernetes version 1.16 and 1.14. For the current release BAF has been tested and being actively developed on Amazon EKS with Kubernetes version 1.16. AWS cli credentials are also required.

Please follow [Amazon instructions](#) to set-up your required Kubernetes cluster(s). To connect to Kubernetes cluster(s), you would also need kubectl Command Line Interface (CLI). Please refer [here](#) for installation instructions, although the Blockchain Automation Framework configuration code (Ansible scripts) installs this automatically.

### 5.1.4 HashiCorp Vault

In this current release, [Hashicorp Vault](#) is mandatory for the Blockchain Automation Framework (BAF) as the certificate and key storage solution; so to use BAF, at least one Vault server should be available. BAF recommends one Vault per organization for production-ready projects.

Follow [official instructions](#) to deploy Vault in your environment.

---

**NOTE:** Recommended approach is to create one Vault deployment on one VM and configure the backend as a cloud storage.

---

### 5.1.5 Ansible

The Blockchain Automation Framework configuration is essentially Ansible scripts, so install Ansible on the machine from which you will deploy the DLT/Blockchain network. This can be a local machine as long as Ansible commands can run on it.

Please note that this machine (also called **Ansible Controller**) should have connectivity to the Kubernetes cluster(s) and the Hashicorp Vault service(s). And it is essential to install the [git client](#) on the Ansible Controller.

---

**NOTE:** The current Blockchain Automation Framework requires minimum **Ansible version 2.9.4** and **Python3**

**NOTE (MacOS):** Ansible requires GNU tar. Install it on MacOS through Homebrew `brew install gnu-tar`

---

### Configuring Ansible Inventory file

In the Blockchain Automation Framework, we connect to Kubernetes cluster through the **Ansible Controller** and do not modify or connect to any other machine directly. The Blockchain Automation Framework's sample inventory file is located [here](#).

Add the contents of this file in your Ansible host configuration file (typically in file `/etc/ansible/hosts`).

Read more about Ansible inventory [here](#).

### 5.1.6 Docker

The Blockchain Automation Framework provides pre-built docker images which are available on [Docker Hub](#). If specific changes are needed in the Docker images, then you can build them locally using the Dockerfiles provided. A user needs to install [Docker CLI](#) to make sure the environment has the capability of building these Dockerfiles to generate various docker images. Platform specific docker image details are mentioned [here](#).

---

**NOTE:** The Blockchain Automation Framework uses minimum Docker version 18.03.0

---

You can check the version of Docker you have installed with the following command from a terminal prompt:

```
docker --version
```

For storing private docker images, a private docker registry can be used. Information such as registry url, username, password etc. can be configured in the configuration file like [Fabric configuration file](#).

### Docker Build for dev environments

The Blockchain Automation Framework is targetted for Production systems, but, in case, a developer environment is needed, you can create a containerized Ansible machine to deploy the dev DLT/Blockchain network using docker build.

The details on how to create a containerized Ansible machine is mentioned [here](#).

---

**NOTE:** This containerized machine (also called **Ansible Controller**) should have connectivity to the Kubernetes cluster(s) and the Hashicorp Vault service(s).

---

### 5.1.7 Internet Domain

As you may have read in the [Kubernetes key concepts](#), the Blockchain Automation Framework uses [Ambassador](#) or [HAProxy Ingress Controller](#) for inter-cluster communication. So, for the Kubernetes services to be available outside the specific cluster, at least one DNS Domain is required. This domain name can then be sub-divided across multiple clusters and the domain-resolution configured for each. Although for production implementations, each organization (and thereby each cluster), must have one domain name.

---

**NOTE:** If single cluster is being used for all organizations in a dev/POC environment, then domain name is not needed.

## 5.2 Configure Pre-requisites

- *Ansible Inventory file*
- *Private Key for GitOps*
- *Docker Images*
- *Vault Initialization and unseal*

- *Ambassador*
- *External DNS*
- *HAProxy Ingress*

### 5.2.1 Ansible Inventory file

If not done already, configure the Ansible controller with this sample inventory file is located [here](#).

Add the contents of this file in your Ansible host configuration file (typically in file `/etc/ansible/hosts`).

Read more about Ansible inventory [here](#).

### 5.2.2 Private Key for GitOps

For synchronizing the Git repo with the cluster, the Blockchain Automation Framework configures Flux for each cluster. The authentication is via SSH key, so this key should be generated before you run the playbooks. Run the following command to generate a private-public key pair named **gitops**.

```
ssh-keygen -q -N "" -f ./gitops
```

The above command generates an SSH key-pair: **gitops** (private key) and **gitops.pub** (public key).

Use the path to the private key (**gitops**) in the `gitops.private_key` section of the [configuration file](#).

---

**NOTE:** Ensure that the Ansible host has read-access to the private key file (`gitops`).

---

And add the public key contents (starts with **ssh-rsa**) as an Access Key (with read-write permissions) in your Github repository by following [this guide](#).

### 5.2.3 Docker Images

The Blockchain Automation Framework provides pre-built docker images which are available on [Docker Hub](#). Ensure that the versions/tags you need are available. If not, raise it on our [RocketChat Channel](#).

For Corda Enterprise, the docker images should be built and put in a private docker registry. Please follow [these instructions](#) to build docker images for Corda Enterprise.

---

**NOTE:** The Blockchain Automation Framework recommends use of private docker registry for production use. The username/password for the docker registry can be provided in a **network.yaml** file so that the Kubernetes cluster can access the registry.

---

### 5.2.4 Unseal Hashicorp Vault

Hashicorp Vault is one of the pre-requisites for the Blockchain Automation Framework. The Vault service should be accessible by the ansible host as well as the kubernetes cluster (proper inbound/outbound rules should be configured). If not initialised and unsealed already, complete the following steps to unseal and access the Vault.

- Install Vault client. Follow the instructions on [Install Vault](#).
- Set the environment Variable **VAULT\_ADDR** to the Vault service. Note that this service should be accessible from the host where you are running this command from, as well as the Ansible controller and the Kubernetes nodes.

```
export VAULT_ADDR=http://my-vault-server:9000
```

- Now execute the following:

```
vault operator init -key-shares=1 -key-threshold=1
```

It will give following output:

```
Unseal Key 1: << unseal key>>
Initial Root Token: << root token>>
```

Save the root token and unseal key in a secure location. This root token is to be updated in the Blockchain Automation Framework's network.yaml file before running the Ansible playbook(s) to deploy the DLT/Blockchain network.

- Unseal with the following command:

```
vault operator unseal << unseal-key-from-above >>
```

- Run this command to login and check if Vault is unsealed:

```
vault login << give the root token >>
```

- Enable secrets engine

```
vault secrets enable -version=1 -path=secret kv
```

You may generate multiple root tokens at the time of initialising the Vault, and delete the one used in the network.yaml file as that is visible in ansible logs.

---

**NOTE:** It is recommended to use Vault auto-unseal using Cloud KMS for Production Systems. And use root token rotation.

---

### 5.2.5 Ambassador

The Blockchain Automation Framework (BAF) uses [Ambassador](#) for inter-cluster communication. To enable BAF Kubernetes services from one Kubernetes cluster to talk to services in another cluster, Ambassador needs to be configured as per the following steps:

- After Ambassador is deployed on the cluster (manually or using `platforms/shared/configuration/kubernetes-env-setup.yaml` playbook), get the external IP address of the Ambassador service.

```
kubectl get services -o wide
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
ambassador	LoadBalancer	172.20.242.205	a975bdcaad9f311e99aa102933d6d36c-493130229.eu-west-2.elb.amazonaws.com
CF,10030:31291/TCP	13d	service=ambassador	
ambassador-admin	LoadBalancer	172.20.70.22	a975450d8d9f311e99aa102933d6d36c-631754403.eu-west-2.elb.amazonaws.com
flux	ClusterIP	172.20.213.13	<none>
flux-memcached	ClusterIP	172.20.43.214	<none>
kubernetes	ClusterIP	172.20.0.1	<none>

The output of the above command will look like this:  
Service Output

- Copy the **EXTERNAL-IP** for **ambassador** service from the output.

**NOTE:** If Ambassador is configured by the playbook, then this configuration has to be done while the playbook is being executed, otherwise the deployment will fail.

- Configure your subdomain configuration to redirect the external DNS name to this external IP. For example, if you want to configure the external domain suffix as **test.corda.blockchaincloudpoc.com**, then update the DNS mapping to redirect all requests to **\*.test.corda.blockchaincloudpoc.com** towards **EXTERNAL-IP** from above as an ALIAS. In AWS Route53, the settings look like below (in Hosted Zones).

Record Set Name	Any Type	Aliases Only	Weighted Only
Displaying 1 to 24 out of 24 Record Sets			
*.test.corda.blockchaincloudpoc.com.	A	ALIAS dualstack.a975bdcaad9f311e99aa102933d6c	No
*.org1ambassador.blockchaincloudpoc.com.	A	ALIAS dualstack.a72b794b3cfe211e99ff60204829e1	No

DNS Configuration

Ambassador

**NOTE:** Ambassador for AWS and AWS-baremetal expose Hyperledger Indy nodes via a TCP Network Load Balancer with a fixed IP address. The fixed IP address is used as EIP allocation ID for all steward public IPs found in the network.yaml. The same public IP is specified for all stewards within one organization. All ports used by Indy nodes in the particular organization have to be exposed.

## 5.2.6 External DNS

In case you do not want to manually update the route configurations every time you change DNS name, you can use [External DNS](#) for automatic updation of DNS routes. Follow the steps as per your cloud provider, and then use `external_dns: enabled` in the `env` section of the BAF configuration file (network.yaml).

**NOTE:** Detailed configuration for External DNS setup is not provided here, please refer the link above.

### 5.2.7 HAProxy Ingress

From Release 0.3.0.0 onwards, Blockchain Automation Framework (BAF) uses [HAProxy Ingress Controller](#) for inter-cluster communication for Fabric network. To enable Fabric GRPC services from one Kubernetes cluster to talk to GRPC services in another cluster, HAProxy needs to be configured as per the following steps:

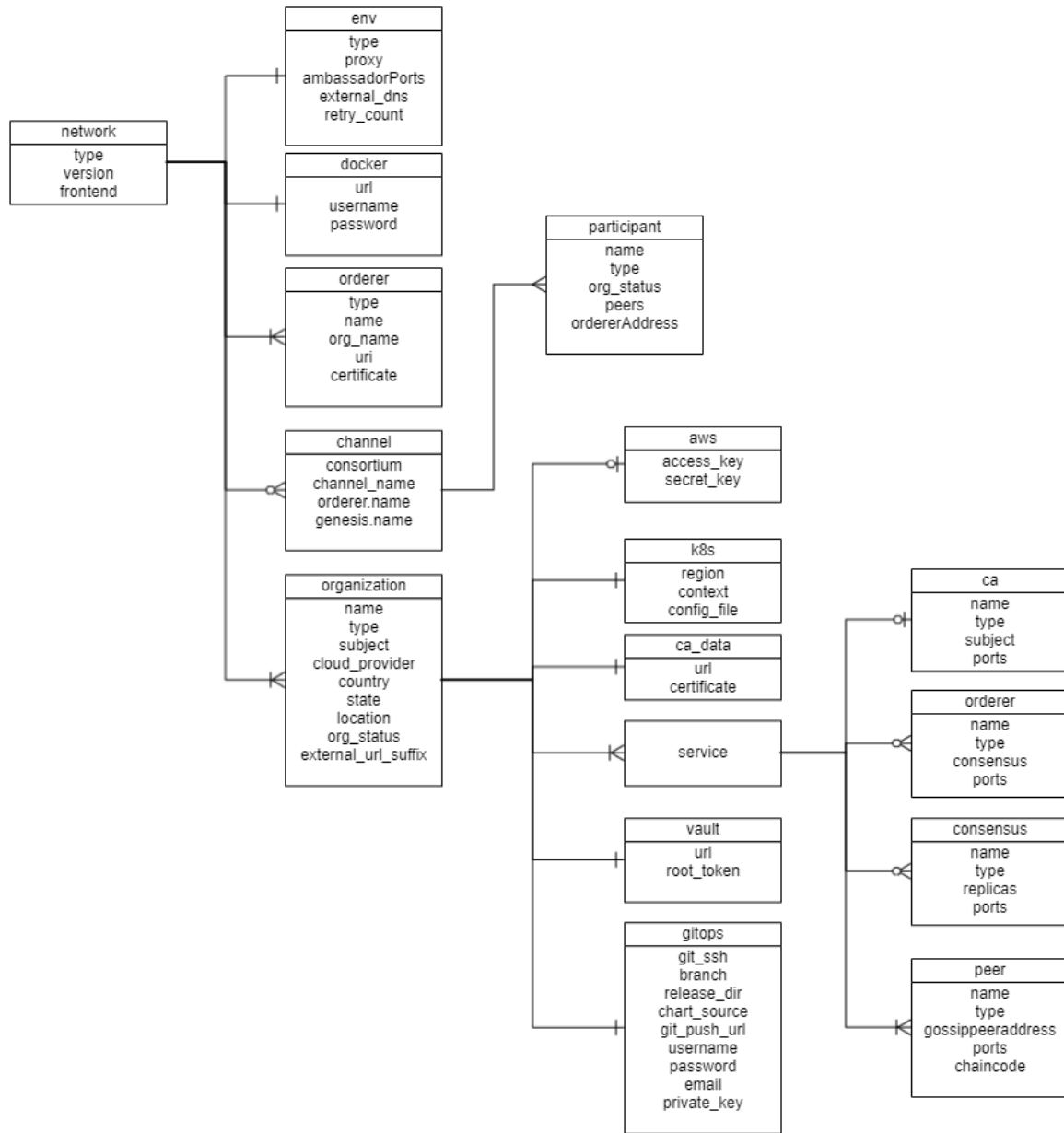
- Use proxy: `haproxy` in the `env` section of the BAF configuration file (`network.yaml`).
- Execute `platforms/shared/configuration/kubernetes-env-setup.yaml` playbook using the BAF configuration file, and then get the external IP address of the HAProxy controller service.

```
kubectl get services --all-namespaces -o wide
```

- Copy the **EXTERNAL-IP** for **haproxy-ingress** service in namespace **ingress-controller** from the output.
- Configure your subdomain configuration to redirect the external DNS name to this external IP. For example, if you want to configure the external domain suffix as **test.corda.blockchaincloudpoc.com**, then update the DNS mapping to redirect all requests to **\*.test.corda.blockchaincloudpoc.com** towards **EXTERNAL-IP** from above as an **ALIAS**.
- Or, you can use *External DNS* above to configure the routes automatically.

## 5.3 Configuration file specification: Hyperledger-Fabric

A `network.yaml` file is the base configuration file designed in the Blockchain Automation Framework for setting up a Fabric DLT network. This file contains all the information related to the infrastructure and network specifications. Below shows its structure.



Before setting up a Fabric DLT/Blockchain network, this file needs to be updated with the required specifications. A sample configuration file is provided in the repo path: `platforms/hyperledger-fabric/configuration/samples/network-fabricv2.yaml`

The configurations are grouped in the following sections for better understanding.

- type
- version
- docker
- frontend
- env

- orderers
- channels
- organizations

Here is the snapshot from the sample configuration file

```
---
# This is a sample configuration file for SupplyChain App on Single K8s Cluster.
# For multiple K8s clusters, there should be multiple configuration files.
network:
  # Network level configuration specifies the attributes required for each organization
  # to join an existing network.
  type: fabric
  version: 1.4.0
  frontend: enabled

  # Docker registry details where images are stored. This will be used to create k8s secrets
  # Please ensure all required images are built and stored in this registry.
  # Do not check-in ****.
  docker: ...

  env: ...

  # Remote connection information for orderer (will be blank or removed for hosting organization)
  orderers: ...

  # The channels defined for a network with participating peers in each channel
  channels:
  - channel: ...

  # Allows specification of one or many organizations that will be connecting to a network.
  # If an organization is also hosting the root of the network (e.g. doorman, membership service, etc),
  # then these services should be listed in this section as well.
  organizations: ...
```

The sections in the sample configuration file are

`type` defines the platform choice like corda/fabric, here in the example its Fabric

`version` defines the version of platform being used. The current Fabric version support is 1.4.0, 1.4.4 and 2.0.0

`frontend` is a flag which defines if frontend is enabled for nodes or not. Its value can only be enabled/disabled. This is only applicable if the sample Supplychain App is being installed.

`env` section contains the environment type and additional (other than 8443) Ambassador port configuration. Value for `proxy` field under this section can be 'ambassador' or 'haproxy'

The snapshot of the `env` section with example value is below

```
env:
  type: "env_type"           # tag for the environment. Important to run_
  ↪multiple flux on single cluster
  proxy: haproxy             # values can be 'haproxy' or 'ambassador'
  ambassadorPorts: 15010,15020 # is valid only if proxy='ambassador'
  retry_count: 100           # Retry count for the checks
  external_dns: enabled      # Should be enabled if using external-dns for_
  ↪automatic route configuration
```

The fields under `env` section are

`docker` section contains the credentials of the repository where all the required images are built and stored.

The snapshot of the `docker` section with example values is below

```
# Docker registry details where images are stored. This will be used to create k8s_
↪secrets
# Please ensure all required images are built and stored in this registry.
# Do not check-in docker_password.
docker:
  url: "docker_url"
  username: "docker_username"
  password: "docker_password"
```

The fields under docker section are

---

**NOTE:** Please follow [these instructions](#) to build and store the docker images before running the Ansible playbooks.

---

orderers section contains a list of orderers with variables which will expose it for the network.

The snapshot of the orderers section with example values is below

```
# Remote connection information for orderer (will be blank or removed for orderer_
↪hosting organization)
orderers:
  - orderer:
    type: orderer
    name: orderer1
    org_name: supplychain #org_name should match one organization_
↪definition below in organizations: key
    uri: orderer1.orglambassador.blockchaincloudpoc.com:8443 # Can be external or_
↪internal URI for orderer which should be reachable by all peers
    certificate: /home/blockchain-automation-framework/build/orderer1.crt
↪# Ensure that the directory exists
  - orderer:
    type: orderer
    name: orderer2
    org_name: supplychain #org_name should match one organization_
↪definition below in organizations: key
    uri: orderer2.orglambassador.blockchaincloudpoc.com:8443 # Can be external or_
↪internal URI for orderer which should be reachable by all peers
    certificate: /home/blockchain-automation-framework/build/orderer2.crt
↪# Ensure that the directory exists
```

The fields under the each orderer are

The channels sections contains the list of channels mentioning the participating peers of the organizations.

The snapshot of channels section with its fields and sample values is below

```
# The channels defined for a network with participating peers in each channel
channels:
  - channel:
    consortium: SupplyChainConsortium
    channel_name: AllChannel
    orderer:
      name: supplychain
    participants:
      - organization:
        name: carrier
        type: creator # creator organization will create the channel and_
↪instantiate chaincode, in addition to joining the channel and install chaincode
```

(continues on next page)

(continued from previous page)

```

    peers:
      - peer:
          name: peer0
          gossipAddress: peer0.carrier-net.org3ambassador.blockchaincloudpoc.com:8443
↪ # External or internal URI of the gossip peer
          ordererAddress: orderer1.org1ambassador.blockchaincloudpoc.com:8443
↪ # External or internal URI of the orderer
      - organization:
          name: store
          type: joiner          # joiner organization will only join the channel and
↪ install chaincode
          peers:
            - peer:
                name: peer0
                gossipAddress: peer0.store-net.org3ambassador.blockchaincloudpoc.com:8443
                ordererAddress: orderer1.org1ambassador.blockchaincloudpoc.com:8443
            - organization:
                name: warehouse
                type: joiner
                peers:
                  - peer:
                      name: peer0
                      gossipAddress: peer0.warehouse-net.org2ambassador.blockchaincloudpoc.com:8443
                      ordererAddress: orderer1.org1ambassador.blockchaincloudpoc.com:8443
                  - organization:
                      name: manufacturer
                      type: joiner
                      peers:
                        - peer:
                            name: peer0
                            gossipAddress: peer0.manufacturer-net.org2ambassador.blockchaincloudpoc.
↪ com:8443
                            ordererAddress: orderer1.org1ambassador.blockchaincloudpoc.com:8443
                      genesis:
                        name: OrdererGenesis

```

The fields under the channel are

Each organization field under participants field of the channel contains the following fields

The organizations section contains the specifications of each organization. In the sample configuration example, we have five organization under the organizations section

The snapshot of an organization field with sample values is below

```

organizations:
  # Specification for the 1st organization. Each organization maps to a VPC and a
↪ separate k8s cluster
  - organization:
      name: supplychain
      country: UK
      state: London
      location: London
      subject: "O=Orderer,L=51.50/-0.13/London,C=GB"
      type: orderer
      external_url_suffix: org1ambassador.blockchaincloudpoc.com
      cloud_provider: aws # Options: aws, azure, gcp, minikube

```

Each organization under the organizations section has the following fields.

For the aws and k8s field the snapshot with sample values is below

```
aws:
  access_key: "<aws_access_key>"      # AWS Access key, only used when cloud_
↪provider=aws
  secret_key: "<aws_secret>"          # AWS Secret key, only used when cloud_
↪provider=aws

# Kubernetes cluster deployment variables.
k8s:
  region: "<k8s_region>"
  context: "<cluster_context>"
  config_file: "<path_to_k8s_config_file>"
```

The aws field under each organization contains: (This will be ignored if cloud\_provider is not 'aws')

The k8s field under each organization contains

For gitops fields the snapshot from the sample configuration file with the example values is below

```
# Git Repo details which will be used by GitOps/Flux.
gitops:
  git_ssh: "git@github.com:<username>/blockchain-automation-framework.git" #_
↪Gitops ssh url for flux value files
  branch: "<branch_name>" #_
↪Git branch where release is being made
  release_dir: "platforms/hyperledger-fabric/releases/dev" # Relative Path in_
↪the Git repo for flux sync per environment.
  chart_source: "platforms/hyperledger-fabric/charts" # Relative Path_
↪where the Helm charts are stored in Git repo
  git_push_url: "github.com:<username>/blockchain-automation-framework.git" #_
↪without https://
  username: "<username>" # Git Service user who has rights to check-in_
↪in all branches
  password: "<password>" # Git Server user password/personal token
  email: "<git_email>" # Email to use in git config
  private_key: "<path to gitops private key>"
```

The gitops field under each organization contains

The services field for each organization under organizations section of Fabric contains list of services which could be ca/orderers/consensus/peers based on if the type of organization.

Each organization will have a ca service under the service field. The snapshot of ca service with example values is below

```
# Services maps to the pods that will be deployed on the k8s cluster
# This sample is an orderer service and includes a zk-kafka consensus
services:
  ca:
    name: ca
    subject: "/C=GB/ST=London/L=London/O=Orderer/CN=ca.supplychain-net"
    type: ca
    grpc:
      port: 7054
```

The fields under ca service are

Each organization with type as peer will have a peers service. The snapshot of peers service with example values is below

```

peers:
- peer:
  name: peer0
  type: anchor      # This can be anchor/nonanchor. Atleast one peer should be
↪anchor peer.
  gossippeeraddress: peer0.manufacturer-net:7051 # Internal Address of the
↪other peer in same Org for gossip, same peer if there is only one peer
  grpc:
    port: 7051
  events:
    port: 7053
  couchdb:
    port: 5984
  restserver:      # This is for the rest-api server
    targetPort: 20001
    port: 20001
  expressapi:      # This is for the express api server
    targetPort: 3000
    port: 3000
  chaincode:
    name: "chaincode_name" #This has to be replaced with the name of the
↪chaincode
    version: "chaincode_version" #This has to be replaced with the version of
↪the chaincode
    maindirectory: "chaincode_main" #The main directory where chaincode is
↪needed to be placed
    repository:
      username: "git_username"      # Git Service user who has rights to
↪check-in in all branches
      password: "git_password"
      url: "github.com/hyperledger-labs/blockchain-automation-framework.git"
      branch: develop
      path: "chaincode_src" #The path to the chaincode
      arguments: 'chaincode_args' #Arguments to be passed along with the
↪chaincode parameters
      endorsements: "" #Endorsements (if any) provided along with the chaincode

```

The fields under peer service are

The organization with orderer type will have consensus service. The snapshot of consensus service with example values is below

```

consensus:
  name: kafka
  type: broker
  replicas: 4
  grpc:
    port: 9092

```

The fields under consensus service are

The organization with orderer type will have orderers service. The snapshot of orderers service with example values is below

```

orderers:

```

(continues on next page)

(continued from previous page)

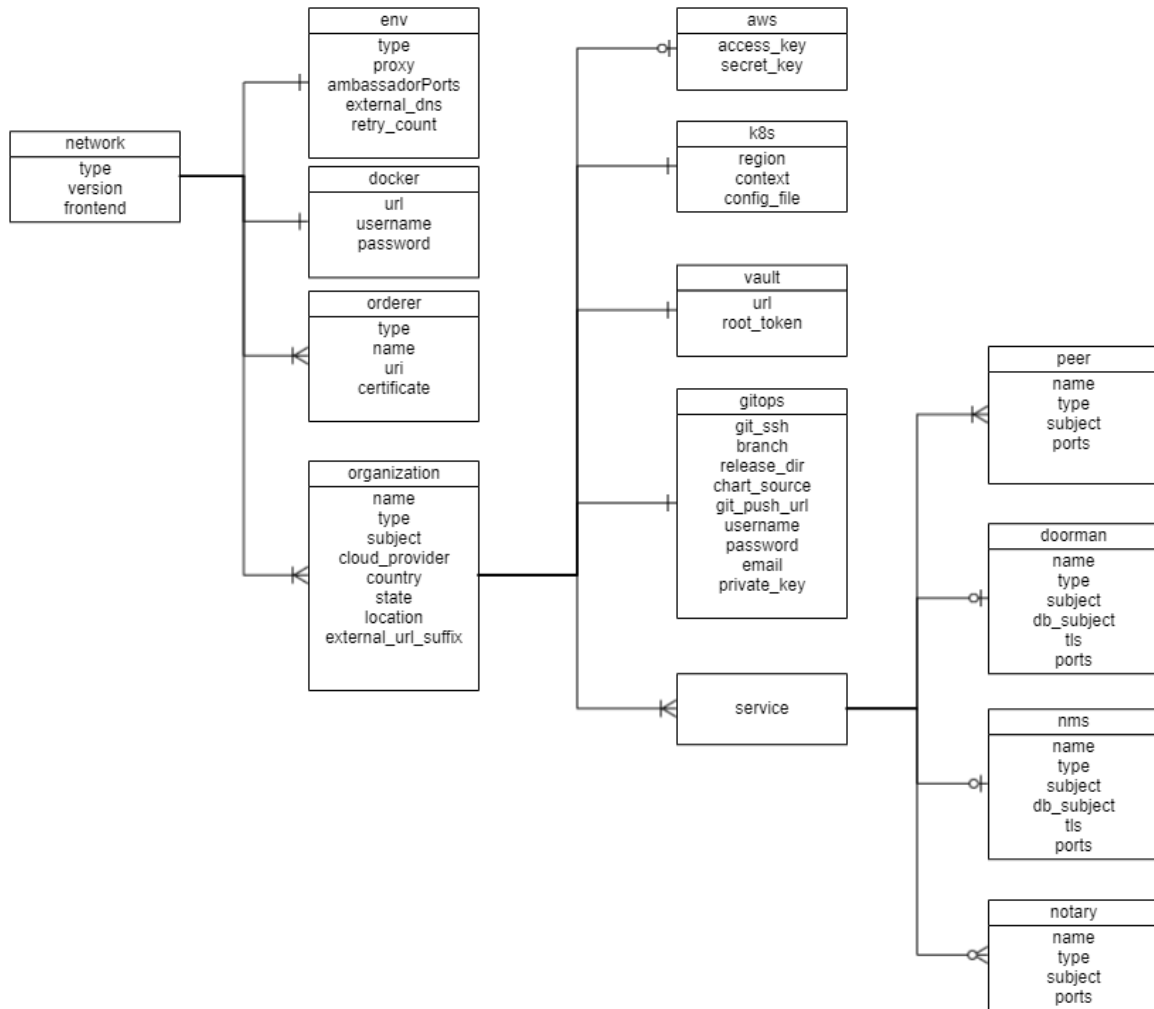
```
# This sample has multiple orderers as an example.
# You can use a single orderer for most production implementations.
- orderer:
  name: orderer1
  type: orderer
  consensus: kafka
  grpc:
    port: 7050
- orderer:
  name: orderer2
  type: orderer
  consensus: kafka
  grpc:
    port: 7050
```

The fields under `orderer` service are

\ \*\* feature is in future scope

## 5.4 Configuration file specification: R3 Corda

A `network.yaml` file is the base configuration file for setting up a Corda DLT network. This file contains all the information related to the infrastructure and network specifications. Here is the structure of it.



Before setting up a Corda DLT/Blockchain network, this file needs to be updated with the required specifications. A sample configuration file is provide in the repo path:platforms/r3-corda/configuration/samples/network-cordav2.yaml

The configurations are grouped in the following sections for better understanding.

- type
- version
- frontend
- env
- docker
- orderers
- organizations

Here is the snapshot from the sample configuration file

```

1  ---
2  # This is a sample configuration file for SupplyChain App on Single K8s Cluster.
3  # For multiple K8s clusters, there should be multiple configuration files.
4  network:
5      # Network level configuration specifies the attributes required for each organization
6      # to join an existing network.
7      type: corda
8      version: 4.0
9
10     frontend: enabled #Flag for frontend to enabled for nodes/peers
11
12     #Environment section to help run multiple applications on same cluster
13     > env: ...
14
15
16     # Docker registry details where images are stored. This will be used to create k8s secrets
17     # Please ensure all required images are built and stored in this registry.
18     # Do not check-in docker_password.
19     > docker: ...
20
21
22
23
24     # Remote connection information for doorman and networkmap (will be blank or removed for hosting organization)
25     > orderers: ...
26
27
28
29
30     # Allows specification of one or many organizations that will be connecting to a network.
31     # If an organization is also hosting the root of the network (e.g. doorman, membership service, etc),
32     # then these services should be listed in this section as well.
33     > organizations: ...

```

The sections in the sample configuration file are

`type` defines the platform choice like `corda/fabric/quorum`. Use `corda` for **Corda Opensource** and `corda-enterprise` for **Corda Enterprise**.

`version` defines the version of platform being used, here in example the Corda version is 4.0, the corda version 4.1 and 4.4 is also supported and should be used. Please note only 4.4 is supported for **Corda Enterprise**.

`frontend` is a flag which defines if frontend is enabled for nodes or not. Its value can only be enabled/disabled. This is only applicable if the sample Supplychain App is being installed.

`env` section contains the environment type and additional (other than 8443) Ambassador port configuration. Value for proxy field under this section has to be 'ambassador' as 'haproxy' has not been implemented for Corda.

The snapshot of the `env` section with example values is below

```

env:
  type: "env-type" # tag for the environment. Important to run_
  ↪multiple flux on single cluster
  proxy: ambassador # value has to be 'ambassador' as 'haproxy' has_
  ↪not been implemented for Corda
  ambassadorPorts: 15010,15020 # Any additional Ambassador ports can be given_
  ↪here, must be comma-separated without spaces, this is valid only if proxy=
  ↪'ambassador'
  retry_count: 20 # Retry count for the checks
  external_dns: enabled # Should be enabled if using external-dns for_
  ↪automatic route configuration

```

The fields under `env` section are

`docker` section contains the credentials of the repository where all the required images are built and stored.

For Opensource Corda, the required instructions are found [here](#).

For **Corda Enterprise**, all Docker images has to be built and stored in a private Docker registry before running the Ansible playbooks. The required instructions are found [here](#).

The snapshot of the `docker` section with example values is below

```
# Docker registry details where images are stored. This will be used to create k8s_
↪secrets
# Please ensure all required images are built and stored in this registry.
docker:
  url: "<url>"
  username: "<username>"
  password: "<password>"
```

The fields under docker section are

**NOTE:** Please follow [these instructions](#) to build and store the docker images before running the Ansible playbooks.

The snapshot of the orderers section with example values is below

```
# Remote connection information for doorman/idman and networkmap (will be blank or_
↪removed for hosting organization)
orderers:
  - orderer:
    name: doorman
    type: doorman
    uri: https://doorman.test.corda.blockchaincloudpoc.com:8443
    certificate: home_dir/platforms/r3-corda/configuration/build/corda/doorman/tls/
↪ambassador.crt
    crlissuer_subject: "CN=Corda TLS CRL Authority,OU=Corda UAT,O=R3 HoldCo LLC,
↪L=New York,C=US"
  - orderer:
    name: networkmap
    type: networkmap
    uri: https://networkmap.test.corda.blockchaincloudpoc.com:8443
    certificate: home_dir/platforms/r3-corda/configuration/build/corda/networkmap/
↪tls/ambassador.crt
    truststore: home_dir/platforms/r3-corda-ent/configuration/build/
↪networkroottruststore.jks #Certificate should be encoded in base64 format
    truststore_pass: rootpassword
```

The orderers section contains a list of doorman/networkmap which is exposed to the network. Each orderer has the following fields:

The organizations section allows specification of one or many organizations that will be connecting to a network. If an organization is also hosting the root of the network (e.g. doorman, membership service, etc), then these services should be listed in this section as well. In the sample example the 1st Organisation is hosting the root of the network, so the services doorman, nms and notary are listed under the 1st organization's service.

The snapshot of an organization field with sample values is below

```
- organization:
  name: manufacturer
  country: CH
  state: Zurich
  location: Zurich
  subject: "O=Manufacturer,OU=Manufacturer,L=Zurich,C=CH"
  type: node
  external_url_suffix: test.corda.blockchaincloudpoc.com
  cloud_provider: aws # Options: aws, azure, gcp
```

Each organization under the `organizations` section has the following fields.

For the `aws` and `k8s` field the snapshot with sample values is below

```
aws:
  access_key: "<aws_access_key>"      # AWS Access key, only used when cloud_
↪provider=aws
  secret_key: "<aws_secret>"          # AWS Secret key, only used when cloud_
↪provider=aws

# Kubernetes cluster deployment variables.
k8s:
  region: "<k8s_region>"
  context: "<cluster_context>"
  config_file: "<path_to_k8s_config_file>"
```

The `aws` field under each organisation contains: (This will be ignored if `cloud_provider` is not 'aws')

The `k8s` field under each organisation contains

For `gitops` fields the snapshot from the sample configuration file with the example values is below

```
# Git Repo details which will be used by GitOps/Flux.
gitops:
  git_ssh: "git@github.com:<username>/blockchain-automation-framework.git" #_
↪Gitops ssh url for flux value files
  branch: "<branch_name>" #_
↪Git branch where release is being made
  release_dir: "platforms/r3-corda/releases/dev" # Relative Path in the Git_
↪repo for flux sync per environment.
  chart_source: "platforms/r3-corda/charts" # Relative Path where the Helm_
↪charts are stored in Git repo
  git_push_url: "github.com:<username>/blockchain-automation-framework.git"
  username: "<username>" # Git Service user who has rights to check-in_
↪in all branches
  password: "<password>" # Git Server user password/personal token
  email: "<git_email>" # Email to use in git config
  private_key: "<path to gitops private key>"
```

The `gitops` field under each organization contains

The `credentials` field under each organization contains

For `cordapps` fields the snapshot from the sample configuration file with the example values is below: This has not been implmented for **Corda Enterprise**.

```
# Cordapps Repository details (optional use if cordapps jar are store in a_
↪repository)
cordapps:
  jars:
    - jar:
        # e.g https://alm.accenture.com/nexus/repository/
        ↪AccentureBlockchainFulcrum_Release/com/supplychain/bcc/cordapp-supply-chain/0.1/
        ↪cordapp-supply-chain-0.1.jar
        url:
    - jar:
        # e.g https://alm.accenture.com/nexus/repository/
        ↪AccentureBlockchainFulcrum_Release/com/supplychain/bcc/cordapp-contracts-states/0.1/
        ↪cordapp-contracts-states-0.1.jar
        url:
```

(continues on next page)

(continued from previous page)

```
username: "cordapps_repository_username"
password: "cordapps_repository_password"
```

The cordapps optional field under each organization contains

The services field for each organization under organizations section of Corda contains list of services which could be doorman/idman/nms/notary/peers for opensource, and additionally idman/networkmap/signer for **Corda Enterprise**.

The snapshot of doorman service with example values is below

```
services:
  doorman:
    name: doormanskar
    subject: "CN=Corda Doorman CA,OU=DLT,O=DLT,L=Berlin,C=DE"
    db_subject: "/C=US/ST=California/L=San Francisco/O=My Company Ltd/OU=DBA/
↪CN=mongoDB"
    type: doorman
    ports:
      servicePort: 8080
      targetPort: 8080
    tls: "on"
```

The fields under doorman service are

The snapshot of nms service example values is below

```
nms:
  name: networkmapskar
  subject: "CN=Network Map,OU=FRA,O=FRA,L=Berlin,C=DE"
  db_subject: "/C=US/ST=California/L=San Francisco/O=My Company Ltd/OU=DBA/
↪CN=mongoDB"
  type: networkmap
  ports:
    servicePort: 8080
    targetPort: 8080
  tls: "on"
```

The fields under nms service are

For **Corda Enterprise**, following services must be added to CENM Support.

The snapshot of idman service with example values is below

```
services:
  idman:
    name: idman
    subject: "CN=Test Identity Manager Service Certificate, OU=HQ, O=HoldCo LLC,
↪ L=New York, C=US"
    crlissuer_subject: "CN=Corda TLS CRL Authority,OU=Corda UAT,O=R3 HoldCo LLC,
↪L=New York,C=US"
    type: cenm-idman
    port: 10000
```

The fields under idman service are

The snapshot of networkmap service with example values is below

```

services:
  networkmap:
    name: networkmap
    subject: "CN=Test Network Map Service Certificate, OU=HQ, O=HoldCo LLC, L=New York, C=US"
    type: cenm-networkmap
    ports:
      servicePort: 10000
      targetPort: 10000

```

The fields under networkmap service are

The snapshot of signer service with example values is below

```

services:
  signer:
    name: signer
    subject: "CN=Test Subordinate CA Certificate, OU=HQ, O=HoldCo LLC, L=New York, C=US"
    type: cenm-signer
    ports:
      servicePort: 8080
      targetPort: 8080

```

The fields under signer service are

The snapshot of notary service with example values is below

```

# Currently only supporting a single notary cluster, but may want to expand in the future
notary:
  name: notaryskar
  subject: "O=Notary, OU=Notary, L=London, C=GB"
  type: notary
  p2p:
    port: 10002
    targetPort: 10002
    ambassador: 15010 #Port for ambassador service (must be from env. ambassadorPorts above)
  rpc:
    port: 10003
    targetPort: 10003
  rpcadmin:
    port: 10005
    targetPort: 10005
  dbtcp:
    port: 9101
    targetPort: 1521
  dbweb:
    port: 8080
    targetPort: 81

```

The fields under notary service are

The snapshot of peer service with example values is below

```

# The participating nodes are named as peers
services:

```

(continues on next page)

(continued from previous page)

```

peers:
- peer:
  name: manufacturerskar
  subject: "O=Manufacturer,OU=Manufacturer,L=47.38/8.54/Zurich,C=CH"
  type: node
  p2p:
    port: 10002
    targetPort: 10002
    ambassador: 15010      #Port for ambassador service (must be from env.
↪ ambassadorPorts above)
  rpc:
    port: 10003
    targetPort: 10003
  rpcadmin:
    port: 10005
    targetPort: 10005
  dbtcp:
    port: 9101
    targetPort: 1521
  dbweb:
    port: 8080
    targetPort: 81
  springboot:             # This is for the springboot server
    targetPort: 20001
    port: 20001
  expressapi:             # This is for the express api server
    targetPort: 3000
    port: 3000

```

The fields under each peer service are

For **Corda Enterprise**, following additional fields have been added under each peer.

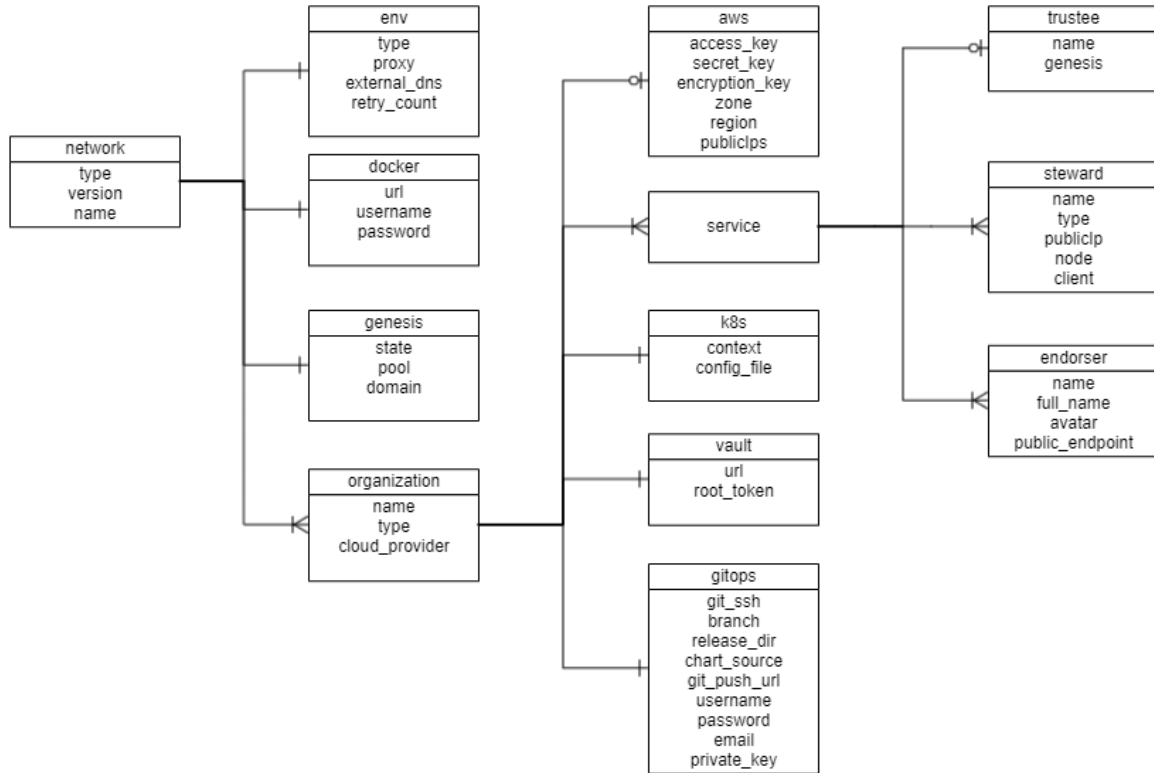
```

firewall:
  enabled: true             # true if firewall components are to be_
↪ deployed
  dmz: true                 # true if a separate k8s cluster is being_
↪ used for Corda Float. false if same cluster is used.
  dmz_k8s:                  # DMZ cluster details. Must be under the_
↪ same Cloud provider as the main cluster.
    context: "dmz_cluster_context"
    config_file: "dmz_cluster_config"
  float:
    name: float
    port: 39999
    subject: "CN=Float Local,O=Local Only,L=London,C=GB"
  bridge:
    name: bridge
    subject: "CN=Bridge Local,O=Local Only,L=London,C=GB"
    port: 40000
    tunnelport: 40000
  hsm:                     # hsm support for future release
    enabled: false

```

## 5.5 Configuration file specification: Indy

A network.yaml file is the base configuration file for setting up a Indy network. This file contains all the information related to the infrastructure and network specifications. Here is the structure of it.



Before setting up a Indy network, this file needs to be updated with the required specifications. A sample configuration file is provide in the repo path:platforms/hyperledger-indy/configuration/samples/network-indyv3.yaml

The configurations are grouped in the following sections for better understanding.

- type
- version
- env
- docker
- name
- genesis
- organizations

Here is the snapshot from the sample configuration file

```
# This is a sample configuration file for hyperledger indy which can reused for a sample indy network of 9 nodes.
# It has 3 organizations:
# 1. organization "authority" with 1 trustee
# 2. organization "provider" with 1 trustee, 2 stewards and 1 endorser
# 3. organization "partner" with 1 trustee, 2 stewards and 1 endorser

network:
  # Network level configuration specifies the attributes required for each organization
  # to join an existing network.
  type: indy
  version: 1.9.2

  #Environment section for Kubernetes setup
  env:
    type: "env_type"          # tag for the environment. Important to run multiple flux on single cluster
    proxy: ambassador         # value has to be 'ambassador' as 'haproxy' has not been implemented for Indy
    retry_count: 20           # Retry count for the checks
    external_dns: disabled    # Should be enabled if using external-dns for automatic route configuration

  # Docker registry details where images are stored. This will be used to create k8s secrets
  # Please ensure all required images are built and stored in this registry.
  # Do not check-in docker_password.
  docker:
    url: "docker_url"
    username: "docker_username"
    password: "docker_password"

  # It's used as the Indy network name (has impact e.g. on paths where the Indy nodes look for crypto files on their local filesystem)
  name: baf

  # Information about pool transaction genesis and domain transactions genesis
  genesis:
    state: absent
    pool: genesis/pool_transactions_genesis
    domain: domain/domain_transactions_genesis

  # Allows specification of one or many organizations that will be connecting to a network.
  organizations:
    # Specification for the 1st organization. Each organization maps to a VPC and a separate k8s cluster
```

The sections in the sample configuration file are

`type` defines the platform choice like corda/fabric/indy, here in example its Indy

`version` defines the version of platform being used, here in example the Indy version is 1.9.2.

`env` section contains the environment type and additional configuration. Value for `proxy` field under this section has to be ‘ambassador’ as ‘haproxy’ has not been implemented for Indy.

The snapshot of the `env` section with example values is below

```
env:
  type: "env_type"          # tag for the environment. Important to run_
  ↪multiple flux on single cluster
  proxy: ambassador         # value has to be 'ambassador' as 'haproxy' has_
  ↪not been implemented for Indy
  # Any additional Ambassador ports can be given below, must be comma-separated_
  ↪without spaces.
  # Must be different from all steward ambassador ports specified in the rest of_
  ↪this network yaml
  ambassadorPorts: 15010,15020,15030,15040
  retry_count: 20           # Retry count for the checks
  external_dns: disabled    # Should be enabled if using external-dns for_
  ↪automatic route configuration
```

The fields under `env` section are

`docker` section contains the credentials of the repository where all the required images are built and stored.

The snapshot of the `docker` section with example values is below

```
# Docker registry details where images are stored. This will be used to create k8s_
↪secrets
# Please ensure all required images are built and stored in this registry.
# Do not check-in docker_password.
docker:
  url: "docker_url"
  username: "docker_username"
  password: "docker_password"
```

The fields under docker section are

---

**NOTE:** Please follow [these instructions](#) to build and store the docker images before running the Ansible playbooks.

---

name is used as the Indy network name (has impact e.g. on paths where the Indy nodes look for crypto files on their local filesystem)

The snapshot of the genesis section with example values is below

```
# Information about pool transaction genesis and domain transactions genesis
genesis:
  state: absent
  pool: genesis/pool_transactions_genesis
  domain: domain/domain_transactions_genesis
```

The genesis section contains Information about pool transaction genesis and domain transactions genesis. genesis contains the following fields:

The organizations section allows specification of one or many organizations that will be connecting to a network. If an organization is also hosting the root of the network (e.g. membership service, etc), then these services should be listed in this section as well.

The snapshot of an organization field with sample values is below

```
- organization:
  name: authority
  type: peer
  external_url_suffix: indy.blockchaincloudpoc.com # Provide the external dns_
↪suffix. Only used when Indy webserver/Clients are deployed.
  cloud_provider: aws # Values can be 'aws-baremetal', 'aws' or
↪'minikube'
```

Each organization under the organizations section has the following fields.

For the aws and k8s field the snapshot with sample values is below

```
aws:
  access_key: "aws_access_key" # AWS Access key
  secret_key: "aws_secret_key" # AWS Secret key
  encryption_key: "encryption_key_id" # AWS encryption key. If present, it
↪'s used as the KMS key id for K8S storage class encryption.
  zone: "availability_zone" # AWS availability zone
  region: "region" # AWS region

  publicIps: ["1.1.1.1", "2.2.2.2"] # List of all public_
↪IP addresses of each availability zone
```

(continues on next page)

(continued from previous page)

```

# Kubernetes cluster deployment variables. The config file path has to be_
↪provided in case
# the cluster has already been created.
k8s:
  config_file: "cluster_config"
  context: "kubernetes-admin@kubernetes"

```

The aws field under each organisation contains: (This will be ignored if cloud\_provider is not 'aws')

The publicIps field under each organisation contains:

**NOTE:** Network.yaml file consists of more organizations, where each organization can be under different availability zone. It means, that each organization has different IP. The field publicIps holds list of all IPs of all organizations in the same cluster. This should be in JSON Array format like ["1.1.1.1","2.2.2.2"] and must contain different IP for each availability zone on the K8s cluster i.e. If the K8s cluster is in two AZ, then two IP addresses should be provided here.

The k8s field under each organisation contains

For the vault field the snapshot with sample values is below

```

# Hashicorp Vault server address and root-token. Vault should be unsealed.
# Do not check-in root_token
vault:
  url: "vault_addr"
  root_token: "vault_root_token"

```

The vault field under each organisation contains:

For gitops fields the snapshot from the sample configuration file with the example values is below

```

# Git Repo details which will be used by GitOps/Flux.
# Do not check-in git_password
gitops:
  git_ssh: "gitops_ssh_url" # Gitops ssh url for flux value_
↪files like "ssh://git@github.com/hyperledger-labs/blockchain-automation-framework.
↪git"
  branch: "gitops_branch" # Git branch where release is_
↪being made
  release_dir: "gitops_release_dir" # Relative Path in the Git repo_
↪for flux sync per environment.
  chart_source: "gitops_charts" # Relative Path where the Helm_
↪charts are stored in Git repo
  git_push_url: "gitops_push_url" # Gitops https URL for git push_
↪like "github.com/hyperledger-labs/blockchain-automation-framework.git"
  username: "git_username" # Git Service user who has rights_
↪to check-in in all branches
  password: "git_password" # Git Server user password
  email: "git_email" # Email to use in git config
  private_key: "path_to_private_key" # Path to private key file which_
↪has write-access to the git repo

```

The gitops field under each organization contains

The services field for each organization under organizations section of Indy contains list of services which could be trustee/steward/endorser

The snapshot of trustee service with example values is below

```
services:
  trustees:
  - trustee:
    name: provider-trustee
    genesis: true
    server:
      port: 8000
      ambassador: 15010
```

The fields under `trustee` service are (find more about differences between trustee/steward/endorser [here](#))

The snapshot of steward service example values is below

```
services:
  stewards:
  - steward:
    name: provider-steward-1
    type: VALIDATOR
    genesis: true
    publicIp: 3.221.78.194
    node:
      port: 9711
      targetPort: 9711
      ambassador: 9711          # Port for ambassador service
    client:
      port: 9712
      targetPort: 9712
      ambassador: 9712        # Port for ambassador service
```

The fields under `steward` service are

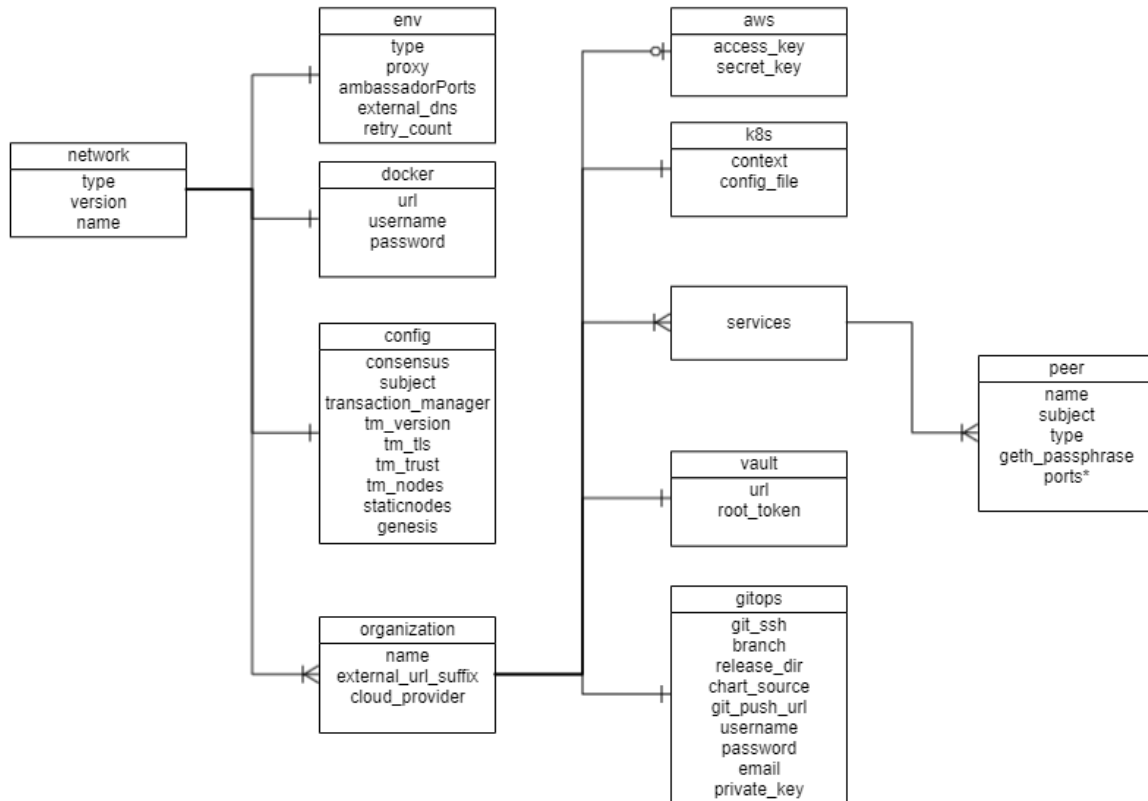
The snapshot of endorser service with example values is below

```
services:
  endorsers:
  - endorser:
    name: provider-endorser
    full_name: Some Decentralized Identity Mobile Services Provider
    avatar: https://provider.com/avatar.png
    # public endpoint will be {{ endorser.name }}.{{ external_url_suffix }}:{{
    ↪ {endorser.server.httpPort}}
    # Eg. In this sample https://provider-endorser.indy.blockchaincloudpoc.
    ↪ com:15020/
    # For minikube: http://<minikubeip>:15020
    server:
      httpPort: 15020
      apiPort: 15030
```

The fields under `endorser` service are

## 5.6 Configuration file specification: Quorum

A `network.yaml` file is the base configuration file designed in the Blockchain Automation Framework for setting up a Quorum DLT network. This file contains all the configurations related to the network that has to be deployed. Below shows its structure.



Before setting up a Quorum DLT/Blockchain network, this file needs to be updated with the required specifications. A sample configuration file is provided in the repo path: `platforms/quorum/configuration/samples/network-quorum.yaml`

The configurations are grouped in the following sections for better understanding.

- type
- version
- env
- docker
- config
- organizations

Here is the snapshot from the sample configuration file

```

---
# This is a sample configuration file for Quorum network which has 4 nodes.
# All text values are case-sensitive
network:
  # Network level configuration specifies the attributes required for each organization
  # to join an existing network.
  type: quorum
  version: 2.1.1 #this is the version of Quorum docker image that will be deployed

#Environment section for Kubernetes setup
env:
  type: "env_type"          # tag for the environment. Important to run multiple flux on single cluster
  proxy: ambassador         # value has to be 'ambassador' as 'haproxy' has not been implemented for Quorum
  ## Any additional Ambassador ports can be given below, must be comma-separated without spaces, this is valid only if proxy='ambassador'
  # These ports are enabled per cluster, so if you have multiple clusters you do not need so many ports
  # This sample uses a single cluster, so we have to open 4 ports for each Node. These ports are again specified for each organization below
  ambassadorPorts: 15010,15011,15012,15013,15020,15021,15022,15023,15030,15031,15032,15033,15040,15041,15042,15043
  retry_count: 20           # Retry count for the checks on Kubernetes cluster
  external_dns: enabled     # Should be enabled if using external-dns for automatic route configuration

# Docker registry details where images are stored. This will be used to create k8s secrets
# Please ensure all required images are built and stored in this registry.
# Do not check-in docker_password.
docker:
  url: "docker_url"
  username: "docker_username"
  password: "docker_password"

# Following are the configurations for the common Quorum network
config: ...

# Allows specification of one or many organizations that will be connecting to a network.
organizations:
  # Specification for the 1st organization. Each organization should map to a VPC and a separate k8s cluster for production deployments
  - organization:
    name: carrier
    external_url_suffix: test.quorum.blockchaincloudpoc.com # This is the url suffix that will be added in DNS recordset. Must be different for different clusters

```

The sections in the sample configuration file are

type defines the platform choice like corda/fabric/indy/quorum, here in the example its **quorum**.

version defines the version of platform being used. The current Quorum version support is only for **2.5.0**

**NOTE:** Use Quorum Version 2.5.0 if you are deploying Supplychain smartcontracts from examples.

env section contains the environment type and additional (other than 8443) Ambassador port configuration. Vaule for proxy field under this section can be ‘ambassador’ or ‘haproxy’

The snapshot of the env section with example value is below

```

env:
  type: "env-type"          # tag for the environment. Important to run
  ↪multiple flux on single cluster
  proxy: ambassador         # value has to be 'ambassador' as 'haproxy' has
  ↪not been implemented for Quorum
  ## Any additional Ambassador ports can be given below, must be comma-separated
  ↪without spaces, this is valid only if proxy='ambassador'
  # These ports are enabled per cluster, so if you have multiple clusters you do
  ↪not need so many ports
  # This sample uses a single cluster, so we have to open 4 ports for each Node.
  ↪These ports are again specified for each organization below
  ambassadorPorts: 15010,15011,15012,15013,15020,15021,15022,15023,15030,15031,
  ↪15032,15033,15040,15041,15042,15043
  retry_count: 50           # Retry count for the checks
  external_dns: enabled     # Should be enabled if using external-dns for
  ↪automatic route configuration

```

The fields under env section are

docker section contains the credentials of the repository where all the required images are built and stored.

The snapshot of the docker section with example values is below

```
# Docker registry details where images are stored. This will be used to create k8s_
↪ secrets
# Please ensure all required images are built and stored in this registry.
# Do not check-in docker_password.
docker:
  url: "docker_url"
  username: "docker_username"
  password: "docker_password"
```

The fields under docker section are

config section contains the common configurations for the Quorum network.

The snapshot of the config section with example values is below

```
config:
  consensus: "raft" # Options are "raft" and "ibft"
  ## Certificate subject for the root CA of the network.
  # This is for development usage only where we create self-signed certificates_
↪ and the truststores are generated automatically.
  # Production systems should generate proper certificates and configure_
↪ truststores accordingly.
  subject: "CN=DLT Root CA,OU=DLT,O=DLT,L=London,C=GB"
  transaction_manager: "tessera" # Options are "tessera" and "constellation"
  # This is the version of "tessera" or "constellation" docker image that will be_
↪ deployed
  # Supported versions #
  # constellation: 0.3.2 (For all versions of quorum)
  # tessera: 0.10.4 (for quorum 2.5.0)
  tm_version: "0.10.4" # This is the version of "tessera" and
↪ "constellation" docker image that will be deployed
  tm_tls: "strict" # Options are "strict" and "off"
  tm_trust: "tofu" # Options are: "whitelist", "ca-or-tofu", "ca",
↪ "tofu"
  ## Transaction Manager nodes public addresses should be provided.
  # For "tessera", all participating nodes should be provided
  # For "constellation", only one is bootnode should be provided
  #
  # For constellation, use following. This will be the bootnode for all nodes
  # - "http://carrier.test.quorum.blockchaincloudpoc.com:15012/" #NOTE the end /_
↪ is necessary and should not be missed
  # The above domain name is formed by the http://(peer.name).(org.external_url_
↪ suffix):(ambassador constellation port)/
  # In the example (for tessera ) below, the domain name is formed by the https://
↪ (peer.name).(org.external_url_suffix):(ambassador default port)
  tm_nodes:
    - "https://carrier.test.quorum.blockchaincloudpoc.com:8443"
    - "https://manufacturer.test.quorum.blockchaincloudpoc.com:8443"
    - "https://store.test.quorum.blockchaincloudpoc.com:8443"
    - "https://warehouse.test.quorum.blockchaincloudpoc.com:8443"
  staticnodes: "/home/user/blockchain-automation-framework/build/quorum_staticnodes
↪ " # Location where staticnodes will be saved
  genesis: "/home/user/blockchain-automation-framework/build/quorum_genesis" #_
↪ Location where genesis file will be saved
  # NOTE for the above paths, the directories should exist
  ##### Following keys are only used when adding new Node(s) to existing network_
↪ and should NOT be used to create new network.
```

(continues on next page)

(continued from previous page)

```

bootnode:
  #name of the bootnode that matches one from existing node
  name: carrier
  #ambassador url of the bootnode
  url: carrier.test.quorum.blockchaincloudpoc.com
  #rpc port of the bootnode
  rpcport: 15011
  #id of the bootnode
  nodeid: 1

```

The fields under config are

The organizations section contains the specifications of each organization. In the sample configuration example, we have four organization under the organizations section.

The snapshot of an organization field with sample values is below

```

organizations:
  # Specification for the 1st organization. Each organization maps to a VPC and a
  ↪ separate k8s cluster
  - organization:
    name: carrier
    external_url_suffix: test.quorum.blockchaincloudpoc.com # This is the url
    ↪ suffix that will be added in DNS recordset. Must be different for different clusters
    cloud_provider: aws # Options: aws, azure, gcp, minikube

```

Each organization under the organizations section has the following fields.

For the aws and k8s field the snapshot with sample values is below

```

aws:
  access_key: "<aws_access_key>" # AWS Access key, only used when cloud_
  ↪ provider=aws
  secret_key: "<aws_secret>" # AWS Secret key, only used when cloud_
  ↪ provider=aws

  # Kubernetes cluster deployment variables.
  k8s:
    context: "<cluster_context>"
    config_file: "<path_to_k8s_config_file>"

```

The aws field under each organization contains: (This will be ignored if cloud\_provider is not aws)

The k8s field under each organization contains

For gitops fields the snapshot from the sample configuration file with the example values is below

```

  # Git Repo details which will be used by GitOps/Flux.
  gitops:
    git_ssh: "git@github.com:<username>/blockchain-automation-framework.git" #
    ↪ Gitops ssh url for flux value files
    branch: "<branch_name>" #
    ↪ Git branch where release is being made
    release_dir: "platforms/Quorum/releases/dev" # Relative Path in the Git repo
    ↪ for flux sync per environment.
    chart_source: "platforms/Quorum/charts" # Relative Path where the Helm
    ↪ charts are stored in Git repo
    git_push_url: "github.com:<username>/blockchain-automation-framework.git" #
    ↪ without https://

```

(continues on next page)

(continued from previous page)

```

    username: "<username>"           # Git Service user who has rights to check-in_
    ↪in all branches
    password: "<password>"           # Git Server user password/personal token
    email: "<git_email>"             # Email to use in git config
    private_key: "<path to gitops private key>"

```

The gitops field under each organization contains

The services field for each organization under organizations section of Quorum contains list of services which could be only peers as of now.

Each organization with type as peer will have a peers service. The snapshot of peers service with example values is below

```

peers:
- peer:
  name: carrier
  subject: "O=Carrier,OU=Carrier,L=51.50/-0.13/London,C=GB" # This is the_
  ↪node subject. L=lat/long is mandatory for supplychain sample app
  type: validator      # value can be validator or non-validator, only_
  ↪applicable if consensus = 'ibft'
  geth_passphrase: 12345 # Passphrase to be used to generate geth account
  p2p:
    port: 21000
    ambassador: 15010 #Port exposed on ambassador service (use one port_
  ↪per org if using single cluster)
  rpc:
    port: 8546
    ambassador: 15011 #Port exposed on ambassador service (use one port_
  ↪per org if using single cluster)
  transaction_manager:
    port: 8443 # use port: 9001 when transaction_manager =
  ↪"constellation"
    ambassador: 8443 # use ambassador: 15012 when transaction_manager =
  ↪"constellation"
  raft:
    # Only used if consensus = 'raft'
    port: 50401
    ambassador: 15013
  db:
    # Only used if transaction_manager = "tessera"
    port: 3306

```

The fields under peer service are

\*\*\* feature is in future scope

## 5.7 Configuration file specification: Hyperledger Besu

A network.yaml file is the base configuration file designed in the Blockchain Automation Framework for setting up a Hyperledger Besu DLT/Blockchain network. This file contains all the configurations related to the network that has to

operations/../../\_static/TopLevelClass-Besu.png

be deployed. Below shows its structure.

Before setting up a Hyperledger Besu DLT/Blockchain network, this file needs to be updated with the required specifications. A sample configuration file is provided in the repo path: `platforms/hyperledger-besu/configuration/samples/network-besu.yaml`

The configurations are grouped in the following sections for better understanding.

- type
- version
- env
- docker
- config
- organizations

Here is the snapshot from the sample configuration file

```
# This is a sample configuration file for Hyperledger Besu network which has 4 nodes.
# All text values are case-sensitive
network:
  # Network level configuration specifies the attributes required for each organization
  # to join an existing network.
  type: besu
  version: 1.4.4 #this is the version of Besu docker image that will be deployed.

#Environment section for Kubernetes setup
env:
  type: "dev" # tag for the environment. Important to run multiple flux on single cluster
  proxy: ambassador # value has to be 'ambassador' as 'haproxy' has not been implemented for Besu
  ## Any additional Ambassador ports can be given below, must be comma-separated without spaces, this is valid only
  # These ports are enabled per cluster, so if you have multiple clusters you do not need so many ports
  # This sample uses a single cluster, so we have to open 4 ports for each Node. These ports are again specified for
  ambassadorPorts: 15010,15011,15012,15013,15020,15021,15022,15023,15030,15031,15032,15033,15040,15041,15042,15043
  retry_count: 20 # Retry count for the checks on Kubernetes cluster
  external_dns: enabled # Should be enabled if using external-dns for automatic route configuration

# Docker registry details where images are stored. This will be used to create k8s secrets
# Please ensure all required images are built and stored in this registry.
# Do not check-in docker_password.
docker:
  url: "index.docker.io/hyperledgerlabs"
  username: "docker_username"
  password: "docker_password"

# Following are the configurations for the common Besu network
config: ...

# Allows specification of one or many organizations that will be connecting to a network.
organizations:
  # Specification for the 1st organization. Each organization should map to a VPC and a separate k8s cluster for prod
  - organization:
    name: carrier
    external_url_suffix: test.besu.blockchaincloudpoc.com # This is the url suffix that will be added in DNS records
```

The sections in the sample configuration file are

`type` defines the platform choice like corda/fabric/indy/quorum/besu, here in the example its **besu**.

`version` defines the version of platform being used. The current Hyperledger Besu version support is only for **1.4.4**.

`env` section contains the environment type and additional (other than 8443) Ambassador port configuration. Value for `proxy` field under this section can be 'ambassador' as 'haproxy' has not been implemented for Besu.

The snapshot of the env section with example value is below

```
env:
  type: "env-type"          # tag for the environment. Important to run
  ↪multiple flux on single cluster
  proxy: ambassador         # value has to be 'ambassador' as 'haproxy' has
  ↪not been implemented for Hyperledger Besu
  ## Any additional Ambassador ports can be given below, must be comma-separated
  ↪without spaces, this is valid only if proxy='ambassador'
  # These ports are enabled per cluster, so if you have multiple clusters you do
  ↪not need so many ports
  # This sample uses a single cluster, so we have to open 4 ports for each Node.
  ↪These ports are again specified for each organization below
  ambassadorPorts: 15010,15011,15012,15013,15020,15021,15022,15023,15030,15031,
  ↪15032,15033,15040,15041,15042,15043
  retry_count: 50           # Retry count for the checks
  external_dns: enabled     # Should be enabled if using external-dns for
  ↪automatic route configuration
```

The fields under env section are

docker section contains the credentials of the repository where all the required images are built and stored.

The snapshot of the docker section with example values is below

```
# Docker registry details where images are stored. This will be used to create k8s
↪secrets
# Please ensure all required images are built and stored in this registry.
# Do not check-in docker_password.
docker:
  url: "docker_url"
  username: "docker_username"
  password: "docker_password"
```

The fields under docker section are

config section contains the common configurations for the Hyperledger Besu network.

The snapshot of the config section with example values is below

```
config:
  consensus: "ibft"         # Options is "ibft" only
  ## Certificate subject for the root CA of the network.
  # This is for development usage only where we create self-signed certificates
  ↪and the truststores are generated automatically.
  # Production systems should generate proper certificates and configure
  ↪truststores accordingly.
  subject: "CN=DLT Root CA,OU=DLT,O=DLT,L=London,C=GB"
  transaction_manager: "orion" # Option is orion only
  # This is the version of "orion" docker image that will be deployed
  # Supported versions #
  # orion: 1.5.2 (for besu 1.4.4)
  tm_version: "1.5.2"       # This is the version of "orion" docker image
  ↪that will be deployed
  # TLS can be True or False for the orion tm
  tm_tls: True
  # Tls trust value
  tm_trust: "ca-or-tofu"    # Options are: "whitelist", "ca-or-tofu",
  ↪"ca", "tofu"
```

(continues on next page)

(continued from previous page)

```
## File location for saving the genesis file should be provided.
genesis: "/home/user/blockchain-automation-framework/build/besu_genesis" #_
↳ Location where genesis file will be saved
```

The fields under config are

The organizations section contains the specifications of each organization. In the sample configuration example, we have four organization under the organizations section.

The snapshot of an organization field with sample values is below

```
organizations:
  # Specification for the 1st organization. Each organization maps to a VPC and a_
  ↳ separate k8s cluster
  - organization:
    name: carrier
    type: member
    # Provide the url suffix that will be added in DNS recordset. Must be different_
    ↳ for different clusters
    external_url_suffix: test.besu.blockchaincloudpoc.com
    # List of all public IP addresses of each availability zone from all_
    ↳ organizations in the same k8s cluster
    # The Ambassador will be set up using these static IPs. The child services will_
    ↳ be assigned the first IP in this list.
    publicIps: ["3.221.78.194","21.23.74.154"] # List of all public IP addresses of_
    ↳ each availability zone from all organizations in the same k8s cluster
    cloud_provider: aws # Options: aws, azure, gcp, minikube
```

Each organization under the organizations section has the following fields.

For the aws and k8s field the snapshot with sample values is below

```
aws:
  access_key: "<aws_access_key>" # AWS Access key, only used when cloud_
  ↳ provider=aws
  secret_key: "<aws_secret>" # AWS Secret key, only used when cloud_
  ↳ provider=aws
  region: "<aws_region>" # AWS Region where cluster and EIPs are_
  ↳ created
  # Kubernetes cluster deployment variables.
  k8s:
    context: "<cluster_context>"
    config_file: "<path_to_k8s_config_file>"
```

The aws field under each organization contains: (This will be ignored if cloud\_provider is not aws)

The k8s field under each organization contains

For gitops fields the snapshot from the sample configuration file with the example values is below

```
# Git Repo details which will be used by GitOps/Flux.
gitops:
  git_ssh: "git@github.com:<username>/blockchain-automation-framework.git" #_
  ↳ Gitops ssh url for flux value files
  branch: "<branch_name>" #_
  ↳ Git branch where release is being made
  release_dir: "platforms/hyperledger-besu/releases/dev" # Relative Path in the_
  ↳ Git repo for flux sync per environment.
```

(continues on next page)

(continued from previous page)

```

    chart_source: "platforms/hyperledger-besu/charts"           # Relative Path where
↳the Helm charts are stored in Git repo
    git_push_url: "github.com/<username>/blockchain-automation-framework.git" #
↳without https://
    username: "<username>"           # Git Service user who has rights to check-in
↳in all branches
    password: "<password>"          # Git Server user password/personal token
    email: "<git_email>"            # Email to use in git config
    private_key: "<path to gitops private key>"

```

The gitops field under each organization contains

The services field for each organization under organizations section of Hyperledger Besu contains list of services which could be peers or validators.

Each organization with type as member will have a peers service. The snapshot of peers service with example values is below

```

peers:
- peer:
  name: carrier
  subject: "O=Carrier,OU=Carrier,L=51.50/-0.13/London,C=GB" # This is the
↳node subject. L=lat/long is mandatory for supplychain sample app
  geth_passphrase: 12345 # Passphrase to be used to generate geth account
  p2p:
    port: 30303
    ambassador: 15010 #Port exposed on ambassador service (use one port
↳per org if using single cluster)
  rpc:
    port: 8545
    ambassador: 15011 #Port exposed on ambassador service (use one port
↳per org if using single cluster)
  ws:
    port: 8546
  tm_nodeport:
    port: 8888
    ambassador: 15013 # Port exposed on ambassador service (Transaction
↳manager node port)
  tm_clientport:
    port: 8080

```

The fields under peer service are

Each organization with type as validator will have a validator service. The snapshot of validator service with example values is below

```

validators:
- validator:
  name: validator1
  bootnode: true # true if the validator node is used also a
↳bootnode for the network
  p2p:
    port: 30303
    ambassador: 15010 #Port exposed on ambassador service (use one port
↳per org if using single cluster)
  rpc:
    port: 8545

```

(continues on next page)

(continued from previous page)

```
    ambassador: 15011      #Port exposed on ambassador service (use one port_
↪per org if using single cluster)
    ws:
      port: 8546
```

The fields under `validator` service are

\*\*\* feature is in future scope

## 5.8 Setting up a DLT/Blockchain network

### 5.8.1 Pre-requisites

To create a Production DLT/Blockchain network, ensure you have the following:

1. One running Kubernetes Cluster and the Config file (`kubeconfig.yaml`) per Organization.
2. One running Hashicorp Vault server per Organization. Unsealed and configured as per [guidance here](#).
3. Domain Name(s) configured as per [guidance here](#).
4. Private key file per Organization for GitOps with write-access to the Git repo as per [guidance here](#).
5. Git user details per Organization as per [pre-requisites](#).
6. Ansible controller configured as per [guidance here](#).

---

**NOTE:** All commands are executed from the `blockchain-automation-framework` directory which is the default directory created when you clone our Git repo.

---

### 5.8.2 Prepare build folder

If not already done, clone the git repository on your Ansible controller.

```
git clone https://github.com/<your username>/blockchain-automation-framework.git
```

Create a folder called `build` inside `blockchain-automation-framework`.

```
cd blockchain-automation-framework
mkdir build
```

Copy the following files inside `build` folder:

- All the Kubernetes config files (`kubeconfig.yaml`).
- All the private key files.

### 5.8.3 Edit the configuration file

Depending on your choice of DLT/Blockchain Platform, select a `network.yaml` and copy it to `build` folder.

```
# eg for Fabric
cp platforms/hyperledger-fabric/configuration/samples/network-fabricv2.yaml build/
↪network.yaml
```

Open and update the `network.yaml` according to the following Platform specific guides.

#### Platform-specific configuration files

- Hyperledger-Fabric
- R3-Corda
- Hyperledger-Indy
- Quorum
- Hyperledger-Besu

In summary, you will need to update the following:

1. `docker` url, username and password.
2. `external_url_suffix` depending on your Domain Name(s).
3. All DNS addresses depending on your Domain Name(s).
4. `cloud_provider`
5. `k8s` section depending on your Kubernetes zone/cluster name/config filepath.
6. `vault`
7. `gitops` section depending on your git username, tokens and private key filepath.

### 5.8.4 Executing provisioning script

After all the configurations are updated in the `network.yaml`, execute the following to create the DLT network

```
# Run the provisioning scripts
ansible-playbook platforms/shared/configuration/site.yaml -e "@./build/network.yaml"
```

The `site.yaml` playbook, in turn calls various playbooks depending on the configuration file and sets up your DLT/Blockchain network.

### 5.8.5 Verify successful configuration of DLT/Blockchain network

To verify if the network is successfully configured or not check if all the kubernetes pods are up and running or not. Below are some commands to check the pod's status:

- `Kubectl get pods --all-namespaces` : To get list of all the pods and their status across all the namespaces. It will look as below -

kubectl get pods --all-namespaces					
NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	ambassador-57756cf686-9s8zn	1/1	Running	0	23h
default	ambassador-57756cf686-cdjwt	1/1	Running	0	23h
default	ambassador-57756cf686-sb595	1/1	Running	0	23h
default	flux-7bd6c79f76-6nlm8	1/1	Running	0	47m
default	flux-helm-operator-84b98dccb8-vqnqd	1/1	Running	0	47m
default	flux-memcached-5c5f957f5f-4wmhk	1/1	Running	0	47m
default	vault-6dfb6f859c-jxnkc	1/1	Running	0	69d
kube-system	aws-node-9dsph	1/1	Running	0	69d
kube-system	aws-node-d77v8	1/1	Running	0	69d
kube-system	aws-node-nklpp	1/1	Running	0	69d
kube-system	coredns-5f6dcd954-grlml	1/1	Running	0	69d
kube-system	coredns-5f6dcd954-w4fv4	1/1	Running	0	69d
kube-system	kube-proxy-42tvf	1/1	Running	0	69d
kube-system	kube-proxy-76v6t	1/1	Running	0	69d
kube-system	kube-proxy-pqcjq	1/1	Running	0	69d
kube-system	tiller-deploy-7b659b7fbd-db554	1/1	Running	0	69d
supplychain-ns	doorman-5b54fcbdf-sv9cr	1/1	Running	0	41m
supplychain-ns	mongodb-doorman-756c5d6898-xt494	1/1	Running	0	41m
supplychain-ns	mongodb-networkmap-64c8f4cdcd-52zqj	1/1	Running	0	41m
supplychain-ns	networkmap-866b9b4c9b-hbdms	1/1	Running	0	41m
supplychain-ns	notary-6b677879cc-4n798	2/2	Running	0	29m
supplychain-ns	notary-register-44nw6	0/2	Completed	0	26m
supplychain-ns	notary-registration-6p26n	0/2	Completed	0	35m
supplychain-ns	notarydb-6678649488-2qhd9	1/1	Running	0	35m

- `Kubectl get pods -n xxxxx` : To check status of pods of a single namespace mentioned in place of xxxxx. Example

```
kubectl get po -n manufacturer-ns
```

NAME	READY	STATUS	RESTARTS	AGE
manufacturer-76fd7db7c5-gsjj7	2/2	Running	0	6d6h

- `Kubectl logs -f <PODNAME> -n <NAMESPACE>` : To check logs of a pod by giving required pod name and namespace in the command. Example-

```
$ kubectl logs -f doorman-5b54fcbdf-4nqjt -n supplychain-ns
newdbnm
starting doorman with the following options
cache-timeout      - 2S
db                  - /opt/doorman/db
doorman             - true
hostname            - 0.0.0.0
mongo-connection-string - mongodb://doorman:newdbnm@mongodb-doorman:27017/admin
mongod-database     - nms
mongod-location     -
network-map-delay   - 1S
param-update-delay  - 10S
port                - 8080
```

For a successful setup of DLT Network all the pods should be in running state.

## 5.8.6 Deleting an existing DLT/Blockchain network

The above mentioned playbook [site.yaml](#) (ReadMe) can be run to reset the network using the network configuration file having the specifications which was used to setup the network using the following command:

```
ansible-playbook platforms/shared/configuration/site.yaml -e "@./build/network.yaml" -
-e "reset=true"
```

## 5.9 Adding a new organization in R3 Corda

- *Prerequisites*

- *Create configuration file*
- *Running playbook to deploy R3 Corda network*

### 5.9.1 Prerequisites

To add a new organization Corda Doorman and Networkmap services should already be running. The public certificates from Doorman and Networkmap should be available and specified in the configuration file.

**NOTE:** Addition of a new organization has been tested on an existing network which is created by BAF. Networks created using other methods may be suitable but this has not been tested by BAF team.

### 5.9.2 Create Configuration File

Refer [this guide](#) for details on editing the configuration file.

The `network.yaml` file should contain the specific `network.organization` patch along with the orderer information about the networkmap and doorman service.

**NOTE:** Make sure the doorman and networkmap service certificates are in plain text and not encoded in base64 or any other encoding scheme, along with correct paths to them mentioned in `network.yaml`.

For reference, sample `network.yaml` file looks like below (but always check the latest at `platforms/r3-corda/configuration/samples`):

```
network:
  # Network level configuration specifies the attributes required for each
  ↪organization
  # to join an existing network.
  type: corda
  version: 4.0
  #enabled flag is frontend is enabled for nodes
  frontend: enabled

  #Environment section for Kubernetes setup
  env:
    type: "env_type"          # tag for the environment. Important to run
    ↪multiple flux on single cluster
    proxy: ambassador         # value has to be 'ambassador' as 'haproxy' has
    ↪not been implemented for Corda
    ambassadorPorts: 15010,15020 # Any additional Ambassador ports can be given
    ↪here, must be comma-separated without spaces, this is valid only if proxy=
    ↪'ambassador'
    retry_count: 20           # Retry count for the checks
    external_dns: enabled     # Should be enabled if using external-dns for
    ↪automatic route configuration

    # Docker registry details where images are stored. This will be used to create k8s
    ↪secrets
```

(continues on next page)

(continued from previous page)

```

# Please ensure all required images are built and stored in this registry.
# Do not check-in docker_password.
docker:
  url: "docker_url"
  username: "docker_username"
  password: "docker_password"

# Remote connection information for doorman and networkmap (will be blank or
↳removed for hosting organization)
orderers:
  - orderer:
    type: doorman
    uri: https://doorman.test.corda.blockchaincloudpoc.com:8443
    certificate: home_dir/platforms/r3-corda/configuration/build/corda/doorman/tls/
↳ambassador.crt
  - orderer:
    type: networkmap
    uri: https://networkmap.test.corda.blockchaincloudpoc.com:8443
    certificate: home_dir/platforms/r3-corda/configuration/build/corda/networkmap/
↳tls/ambassador.crt

# Allows specification of one or many organizations that will be connecting to a
↳network.
# If an organization is also hosting the root of the network (e.g. doorman,
↳membership service, etc),
# then these services should be listed in this section as well.
organizations:
  # Specification for the new organization. Each organization maps to a VPC and a
↳separate k8s cluster
  - organization:
    name: neworg
    country: US
    state: New York
    location: New York
    subject: "O=Neworg,OU=Neworg,L=New York,C=US"
    type: node
    external_url_suffix: test.corda.blockchaincloudpoc.com

    cloud_provider: aws # Options: aws, azure, gcp
    aws:
      access_key: "aws_access_key" # AWS Access key, only used when cloud_
↳provider=aws
      secret_key: "aws_secret_key" # AWS Secret key, only used when cloud_
↳provider=aws

# Kubernetes cluster deployment variables. The config file path and name has to
↳be provided in case
# the cluster has already been created.
k8s:
  region: "cluster_region"
  context: "cluster_context"
  config_file: "cluster_config"

# Hashicorp Vault server address and root-token. Vault should be unsealed.
# Do not check-in root_token
vault:
  url: "vault_addr"

```

(continues on next page)

(continued from previous page)

```

    root_token: "vault_root_token"

    # Git Repo details which will be used by GitOps/Flux.
    # Do not check-in git_password
    gitops:
        git_ssh: "gitops_ssh_url"          # Gitops ssh url for flux value files like
        ↪ "ssh://git@github.com/hyperledger-labs/blockchain-automation-framework.git"
        branch: "gitops_branch"           # Git branch where release is being made
        release_dir: "gitops_release_dir" # Relative Path in the Git repo for flux
        ↪ sync per environment.
        chart_source: "gitops_charts"      # Relative Path where the Helm charts are
        ↪ stored in Git repo
        git_push_url: "gitops_push_url"    # Gitops https URL for git push like
        ↪ "github.com/hyperledger-labs/blockchain-automation-framework.git"
        username: "git_username"          # Git Service user who has rights to check-
        ↪ in in all branches
        password: "git_password"          # Git Server user password
        email: "git_email"                # Email to use in git config
        private_key: "path_to_private_key" # Path to private key file which
        ↪ has write-access to the git repo

    services:
        peers:
        - peer:
            name: neworg
            subject: "O=Neworg,OU=Neworg,L=New York,C=US"
            type: node
            p2p:
                port: 10002
                targetPort: 10002
                ambassador: 10070          #Port for ambassador service (use one port per
        ↪ org if using single cluster)
            rpc:
                port: 10003
                targetPort: 10003
            rpcadmin:
                port: 10005
                targetPort: 10005
            dbtcp:
                port: 9101
                targetPort: 1521
            dbweb:
                port: 8080
                targetPort: 81
            springboot:
                targetPort: 20001
                port: 20001
            expressapi:
                targetPort: 3000
                port: 3000

```

### 5.9.3 Run playbook

The `add-new-organization.yaml` playbook is used to add a new organization to the existing network. This can be done using the following command

```
ansible-playbook platforms/shared/configuration/add-new-organization.yaml --extra-
↳vars "@path-to-network.yaml"
```

**NOTE:** If you have CorDapps and applications, please deploy them as well.

## 5.10 Adding cordapps to R3 Corda network

### 5.10.1 1. Adding directly from build directory

#### Pre-requisites:

R3 Corda network deployed and network.yaml configuration file already set.

#### Build CorDapp jars

Build the CorDapp jars. If you have multiple jars, place them in a single location e.g. at path/to/cordapp-jars.

#### Run playbook

The playbook `deploy-cordapps.yaml` is used to deploy cordapps over the existing R3 Corda network. This can be done manually using the following command

```
ansible-playbook platforms/r3-corda/configuration/deploy-cordapps.yaml -e "@path-to-
↳network.yaml" -e "source_dir='path/to/cordapp-jars'"
```

### 5.10.2 2. Adding from a nexus repository

#### Pre-requisites:

Build the CorDapp jars. If you have multiple jars, place them in a single location e.g. at path/to/cordapp-jars. Publishing the CorDapp jars to the nexus repository.

Inorder to publish the jars add the following information in `example\supplychain-app\corda\gradle.properties` file

```
# Repository URL e.g : https://alm.accenture.com/nexus/repository/
↳AccentureBlockchainFulcrum_Release/
repoURI=nexus_repository_url
# Repository credentials
repoUser=repository_user_name
repoPassword=repository_user_password
```

Add the appropriate jar information as artifacts in `example\supplychain-app\corda\build.gradle` file, change this file only if you need to add or remove jars other than the ones mentioned below

```
publishing{
    publications {
        maven1(MavenPublication) {
            artifactId = 'cordapp-supply-chain'
            artifact('build/cordapp-supply-chain-0.1.jar')
        }
        maven2(MavenPublication) {
            artifactId = 'cordapp-contracts-states'
            artifact('build/cordapp-contracts-states-0.1.jar')
        }
    }
    repositories {
        maven {
            url project.repoURI
            credentials {
                username project.repoUser
                password project.repoPassword
            }
        }
    }
}
```

Publishing the artifacts/jars, use the below command to publish the jars into the nexus repository

```
gradle publish
```

Change the corda configuration file to add jar information under the cordapps field of required organisation.

Example given in the sample configuration file `platforms/r3-corda/configuration/samples/network-cordav2.yaml`

The snapshot from the sample configuration file with the example values is below

```
# Cordapps Repository details (optional use if cordapps jar are store in a_
↪ repository)
cordapps:
  jars:
    - jar:
        # e.g https://alm.accenture.com/nexus/repository/
        ↪ AccentureBlockchainFulcrum_Release/com/supplychain/bcc/cordapp-supply-chain/0.1/
        ↪ cordapp-supply-chain-0.1.jar
        url:
    - jar:
        # e.g https://alm.accenture.com/nexus/repository/
        ↪ AccentureBlockchainFulcrum_Release/com/supplychain/bcc/cordapp-contracts-states/0.1/
        ↪ cordapp-contracts-states-0.1.jar
        url:
        username: "cordapps_repository_username"
        password: "cordapps_repository_password"
```

## Adding the jars by deploying the network

After the configuration file is updated and saved, now run the following command from the **blockchain-automation-framework** folder to deploy your network.

```
ansible-playbook platforms/shared/configuration/site.yaml --extra-vars "@path-to-
↪ network.yaml"
```

This would deploy the network and add the cordapps.

## 5.11 Upgrading chaincode in Hyperledger Fabric

- *Pre-requisites*
- *Modifying configuration file*
- *Running playbook to upgrade chaincode to new version in Hyperledger Fabric network*

### 5.11.1 Pre-requisites

Hyperledger Fabric network deployed, network.yaml configuration file already set and chaincode is installed and instantiated.

### 5.11.2 Modifying configuration file

Refer [this guide](#) for details on editing the configuration file.

The network.yaml file should contain the specific network.organizations.services.peers.chaincode.arguments, network.organizations.services.peers.chaincode.version and network.organizations.services.peers.chaincode.name variables which are used as arguments while upgrading the chaincode.

For reference, following snippet shows that section of network.yaml

```
---
network:
  ..
  ..
  organizations:
    - organization:
        name: manufacturer
        ..
        ..
        services:
          peers:
            - peer:
                name: peer0
                ..
                chaincode:
                  name: "chaincode_name" #This has to be replaced with the name of the_
↪chaincode
                  version: "chaincode_version" # This has to be different than the current_
↪version
                  maindirectory: "chaincode_main" #The main directory where chaincode is_
↪needed to be placed
                  repository:
                    username: "git_username" # Git Service user who has rights to_
↪check-in in all branches
                    password: "git_password"
                    url: "github.com/hyperledger-labs/blockchain-automation-framework.git"
                    branch: develop
                    path: "chaincode_src" #The path to the chaincode
```

(continues on next page)

(continued from previous page)

```
arguments: 'chaincode_args' #Arguments to be passed along with the ↵  
↵chaincode parameters  
endorsements: "" #Endorsements (if any) provided along with the chaincode
```

### 5.11.3 Run playbook

The playbook `chaincode-upgrade.yaml` is used to upgrade chaincode to a new version in the existing fabric network. This can be done by using the following command

```
ansible-playbook platforms/hyperledger-fabric/configuration/chaincode-upgrade.  
↵yaml --extra-vars "@path-to-network.yaml"
```

---

**NOTE:** The Chaincode should be upgraded for all participants of the channel.

## 5.12 Upgrading Hyperledger Fabric version

- *Pre-requisites*
- *Modifying image versions*

### 5.12.1 Pre-requisites

Hyperledger Fabric image versions, which are compatible with the target fabric version need to be known.

For example, for Fabric v1.4.4, these are the image tags of the supporting docker images

And, for Fabric v2.0.0, these are the image tags of the supporting docker images

---

**NOTE:** This change only upgrades the docker images, any other configuration changes is not covered by this guide. Please refer to Fabric documentation for any specific configuration changes.

---

### 5.12.2 Modifying image versions

The `network.yaml` [here](#) should be updated with the required version tag under `network.version` for upgrading the base images of CA, orderer and peer. For example:

```
network:  
  version: 1.4.4
```

2 files need to be edited in order to support version change for kafka, zookeeper and couchDB

### 5.12.3 Executing Ansible playbook

The playbook `site.yaml` ([ReadMe](#)) can be run after the configuration file (for example: `network.yaml` for Fabric) has been updated.

```
ansible-playbook platforms/shared/configuration/site.yaml --extra-vars "@path-to-
↪network.yaml"
\`
```

## 5.13 Adding a new organization in Hyperledger Fabric

- *Prerequisites*
- *Modifying configuration file*
- *Running playbook to deploy Hyperledger Fabric network*

### 5.13.1 Prerequisites

To add a new organization a fully configured Fabric network must be present already, i.e. a Fabric network which has Orderers, Peers, Channels (with all Peers already in the channels). The corresponding crypto materials should also be present in their respective Hashicorp Vault.

---

**NOTE:** Addition of a new organization has been tested on an existing network which is created by BAF. Networks created using other methods may be suitable but this has not been tested by BAF team.

---

### 5.13.2 Modifying Configuration File

Refer [this guide](#) for details on editing the configuration file.

While modifying the configuration file(`network.yaml`) for adding new organization, all the existing organizations should have `org_status` tag as `existing` and the new organization should have `org_status` tag as `new` under `network.channels` eg.

```
network:
  channels:
    - channel:
        ..
        ..
    participants:
      - organization:
          ..
          ..
          org_status: new # new for new organization(s)
      - organization:
          ..
          ..
          org_status: existing # existing for old organization(s)
```

and under `network.organizations` as

```
network:
  organizations:
    - organization:
        ..
        ..
        org_status: new # new for new organization(s)
    - organization:
        ..
        ..
        org_status: existing # existing for old organization(s)
```

The `network.yaml` file should contain the specific `network.organization` patch along with the orderer information.

For reference, see `network-fabric-add-organization.yaml` file [here](#).

### 5.13.3 Run playbook

The `add-new-organization.yaml` playbook is used to add a new organization to the existing network. This can be done using the following command

```
ansible-playbook platforms/shared/configuration/add-new-organization.yaml --extra-
vars "@path-to-network.yaml"
```

---

**NOTE:** Make sure that the `org_status` label was set as `new` when the network is deployed for the first time. If you have additional applications, please deploy them as well.

## 5.14 Verify that Indy DLT/Blockchain network is working

1. You need to verify that network is successfully configured as mentioned in [setting\\_dlt.md](#).
2. Verify that there are no errors in the logs of stewards.
3. Connect to cluster using generated pool-genesis. Readme for pool genesis is [here](#).
4. Execute multiple commands using CLI for indy-SDK. Readme how to install and use indy CLI is [here](#).

## 5.15 Adding a new node in Quorum

- *Prerequisites*
- *Create Configuration File*
- *Run playbook*

### 5.15.1 Prerequisites

To add a new organization in Quorum, an existing quorum network should be running, enode information of all existing nodes present in the network should be available, genesis block should be available in base64 encoding and the geth information of a node should be available and that node account should be unlocked prior adding the new node to the existing quorum network.

**NOTE:** Addition of a new organization has been tested on an existing network which is created by BAF. Networks created using other methods may be suitable but this has not been tested by BAF team.

---

## 5.15.2 Create Configuration File

Refer [this guide](#) for details on editing the configuration file.

The `network.yaml` file should contain the specific `network.organization` patch along with the enode information, genesis block in base64 encoding and geth account details

---

**NOTE:** Make sure that the genesis block information is given in base64 encoding. Also, if you are adding node to the same cluster as of another node, make sure that you add the ambassador ports of the existing node present in the cluster to the `network.yaml`

---

For reference, sample `network.yaml` file looks like below for RAFT consensus (but always check the latest `network-quorum-newnode.yaml` at `platforms/quorum/configuration/samples`):

```
---
# This is a sample configuration file for Quorum network which has 4 nodes.
# All text values are case-sensitive
network:
  # Network level configuration specifies the attributes required for each
  ↪organization
  # to join an existing network.
  type: quorum
  version: 2.5.0 #this is the version of Quorum docker image that will be deployed.
  ↪older version 2.1.1 is not compatible with supplychain contracts

  #Environment section for Kubernetes setup
  env:
    type: "dev" # tag for the environment. Important to run multiple
  ↪flux on single cluster
    proxy: ambassador # value has to be 'ambassador' as 'haproxy' has
  ↪not been implemented for Quorum
    ## Any additional Ambassador ports can be given below, must be comma-separated
  ↪without spaces, this is valid only if proxy='ambassador'
    # These ports are enabled per cluster, so if you have multiple clusters you do
  ↪not need so many ports
    # This sample uses a single cluster, so we have to open 4 ports for each Node.
  ↪These ports are again specified for each organization below
    ambassadorPorts: 15010,15011,15012,15013,15020,15021,15022,15023,15030,15031,
  ↪15032,15033,15040,15041,15042,15043
    retry_count: 20 # Retry count for the checks on Kubernetes cluster
    external_dns: enabled # Should be enabled if using external-dns for
  ↪automatic route configuration

  # Docker registry details where images are stored. This will be used to create k8s
  ↪secrets
  # Please ensure all required images are built and stored in this registry.
```

(continues on next page)

(continues on next page)

(continued from previous page)

```

# MDAwMDAwMDAwIgogICAgICAgIH0sCiAgICAgICAgImY2MjkyNTQ1YWVjNTkyMDU4MzQ
# geth account details
# make sure that the account is unlocked prior to adding a new node
bootnode:
  #name of the node
  name: carrier
  #ambassador url of the node
  url: carrier.test.quorum.blockchaincloudpoc.com
  #rpc port of the node
  rpcport: 15011
  #id of the node.
  nodeid: 1

# Allows specification of one or many organizations that will be connecting to a
↪network.
organizations:
  # Specification for the 1st organization. Each organization should map to a VPC
↪and a separate k8s cluster for production deployments
  - organization:
    name: neworg
    external_url_suffix: test.quorum.blockchaincloudpoc.com # This is the url
↪suffix that will be added in DNS recordset. Must be different for different clusters
    cloud_provider: aws # Options: aws, azure, gcp
    aws:
      access_key: "aws_access_key" # AWS Access key, only used when cloud
↪provider=aws
      secret_key: "aws_secret_key" # AWS Secret key, only used when cloud
↪provider=aws
    # Kubernetes cluster deployment variables. The config file path and name has to
↪be provided in case
    # the cluster has already been created.
    k8s:
      context: "cluster_context"
      config_file: "cluster_config"
    # Hashicorp Vault server address and root-token. Vault should be unsealed.
    # Do not check-in root_token
    vault:
      url: "vault_addr"
      root_token: "vault_root_token"
    # Git Repo details which will be used by GitOps/Flux.
    # Do not check-in git_access_token
    gitops:
      git_ssh: "ssh://git@github.com/<username>/blockchain-automation-framework.git
↪" # Gitops ssh url for flux value files
      branch: "develop" # Git branch where release is being made
      release_dir: "platforms/quorum/releases/dev" # Relative Path in the Git repo
↪for flux sync per environment.
      chart_source: "platforms/quorum/charts" # Relative Path where the Helm
↪charts are stored in Git repo
      git_push_url: "github.com/<username>/blockchain-automation-framework.git" #
↪Gitops https URL for git push
      username: "git_username" # Git Service user who has rights to check-
↪in in all branches
      password: "git_access_token" # Git Server user password
      email: "git_email" # Email to use in git config
      private_key: "path_to_private_key" # Path to private key file which
↪has write-access to the git repo

```

(continues on next page)

(continued from previous page)

```
# The participating nodes are named as peers
services:
  peers:
    - peer:
        name: neworg
        subject: "O=Neworg,OU=Neworg,L=51.50/-0.13/London,C=GB" # This is the node_
        ↪subject. L=lat/long is mandatory for supplychain sample app
        type: validator # value can be validator or non-validator, only_
        ↪applicable if consensus = 'ibft'
        geth_passphrase: 12345 # Passphrase to be used to generate geth account
        p2p:
            port: 21000
            ambassador: 15010 #Port exposed on ambassador service (use one port_
            ↪per org if using single cluster)
        rpc:
            port: 8546
            ambassador: 15011 #Port exposed on ambassador service (use one port_
            ↪per org if using single cluster)
        transaction_manager:
            port: 8443 # use port: 9001 when transaction_manager =
            ↪"constellation"
            ambassador: 8443 # use ambassador: 15012 when transaction_manager =
            ↪"constellation"
        raft: # Only used if consensus = 'raft'
            port: 50401
            ambassador: 15013
        db: # Only used if transaction_manager = "tessera"
            port: 3306
```

Three new sections are added to the network.yaml | Field | Description | |-----|
|-----| | staticnodes | Existing network's static nodes need to be provided
as an array. | | genesis | Existing network's genesis.json needs to be provided in base64. | | bootnode | Bootnode account
details. |

The network.config.bootnode field contains:

### 5.15.3 Run playbook

The site.yaml playbook is used to add a new organization to the existing network. This can be done using the following command

```
ansible-playbook platforms/shared/configuration/site.yaml --extra-vars "@path-to-
↪network.yaml"
```

## 5.16 Adding a new storageclass

As storageclass templates vary as per requirements and cloud provider specifications, this guide will help in using a new storageclass which is not supported by Blockchain Automation Framework (BAF)

- Adding a new storage class for Hyperledger Fabric
- Adding a new storage class for R3-Corda

- *Adding a new storage class for Hyperledger Indy*
- *Adding a new storage class for Quorum*

### 5.16.1 Adding a new storage class for Hyperledger Fabric

To add a new storageclass for Hyperledger Fabric, perform the following steps:

1. Add the new storageclass template `sample_sc.tpl`, under `platforms/hyperledger-fabric/configuration/roles/create/storageclass/templates` with `metadata.name` (storageclass name) as the variable `sc_name`. For example,

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: {{ sc_name }}
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  encrypted: "true"
```

1. Mention the template file, which you created above, under `platforms/hyperledger-fabric/configuration/roles/create/storageclass/vars/main.yaml` with a variable reference. For example,

```
sc_templates:
  sample-sc: sample_sc.tpl
```

1. Set the type variable to `sample-sc` (variable created in step2) in the task `Create Storage Class` value file for `orderers` and `Create Storage Class` value file for `Organizations`, located in `platforms/hyperledger-fabric/configuration/roles/create/storageclass/tasks/main.yaml`

### 5.16.2 Adding a new storage class for R3-Corda

To add a new storageclass for R3-Corda, perform the following steps:

1. Add the new storageclass template `sample_sc.tpl`, under `platforms/r3-corda/configuration/roles/create/k8_component/templates` with `metadata.name` (storageclass name) as the variable `component_name`. For example,

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: {{ component_name }}
provisioner: kubernetes.io/aws-ebs
reclaimPolicy: Delete
volumeBindingMode: Immediate
parameters:
  encrypted: "true"
```

1. Mention the template file, which you created above, under `platforms/r3-corda/configuration/roles/create/k8_component/vars/main.yaml` with a variable reference. For example,

```
dlt_templates:
  sample-sc: sample_sc.tpl
```

1. Set the `component_type` and `component_name` variable to `sample-sc` (variable created in step2) in the task `Create storageclass`, located in `platforms/r3-corda/configuration/roles/create/storageclass/tasks/main.yaml`

### 5.16.3 Adding a new storage class for Hyperledger Indy

To add a new storageclass for Hyperledger Indy, perform the following steps:

1. Add the new storageclass template `sample_sc.tpl`, under `platforms/hyperledger-indy/configuration/roles/create/k8_component/templates` with `metadata.name` (storageclass name) as the variable `component_name`. For example,

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: {{ component_name }}
provisioner: kubernetes.io/aws-ebs
reclaimPolicy: Delete
volumeBindingMode: Immediate
parameters:
  encrypted: "true"
```

1. Mention the template file, which you created above, under `platforms/hyperledger-indy/configuration/roles/create/k8_component/vars/main.yaml` with a variable reference. For example,

```
k8_templates:
  sample-sc: sample_sc.tpl
```

1. Set the `component_name` variable to `sample-sc` (variable created in step2) in the task `Create Storage Class`, located in `platforms/hyperledger-indy/configuration/deploy-network.yaml`

### 5.16.4 Adding a new storage class for Quorum

To add a new storageclass for Quorum, perform the following steps:

1. Add the new storageclass template `sample_sc.tpl`, under `platforms/quorum/configuration/roles/create/k8_component/templates` with `metadata.name` (storageclass name) as the variable `component_name`. For example,

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: {{ component_name }}
provisioner: kubernetes.io/aws-ebs
reclaimPolicy: Delete
volumeBindingMode: Immediate
parameters:
  encrypted: "true"
```

1. Mention the template file, which you created above, under `platforms/quorum/configuration/roles/create/k8_component/vars/main.yaml` with a variable reference. For example,

```
dlt_templates:
  sample-sc: sample_sc.tpl
```

1. Set the `component_type` and `component_name` variable to `sample-sc` (variable created in step2) in the task `Create storageclass`, located in `platforms/quorum/configuration/roles/create/storageclass/tasks/main.yaml`

## 5.17 Adding a new node in besu

- *Prerequisites*
- *Create Configuration File*
- *Run playbook*

### 5.17.1 Prerequisites

To add a new organization in Besu, an existing besu network should be running, enode information of all existing nodes present in the network should be available, genesis file should be available in base64 encoding and the information of orion nodes should be available and new node account should be unlocked prior adding the new node to the existing besu network.

---

**NOTE:** Addition of a new organization has been tested on an existing network which is created by BAF. Networks created using other methods may be suitable but this has not been tested by BAF team.

---

**NOTE:** The guide is only for the addition of non-validator organization in existing besu network.

---

### 5.17.2 Create Configuration File

Refer [this guide](#) for details on editing the configuration file.

The `network.yaml` file should contain the specific `network.organization` patch along with the enode information, genesis file in base64 encoding and orion transaction manager details

---

**NOTE:** Make sure that the genesis file is provided in base64 encoding. Also, if you are adding node to the same cluster as of another node, make sure that you add the ambassador ports of the existing node present in the cluster to the `network.yaml`

---

For reference, sample `network.yaml` file looks like below for IBFT consensus (but always check the latest `network-besu-newnode.yaml` at `platforms/hyperledger-besu/configuration/samples`):

```
---
# This is a sample configuration file for Hyperledger Besu network which has 4 nodes.
# All text values are case-sensitive
network:
  # Network level configuration specifies the attributes required for each
  ↪organization
  # to join an existing network.
  type: besu
```

(continues on next page)

(continued from previous page)

```

version: 1.4.4 #this is the version of Besu docker image that will be deployed.

#Environment section for Kubernetes setup
env:
  type: "dev" # tag for the environment. Important to run multiple_
↪flux on single cluster
  proxy: ambassador # value has to be 'ambassador' as 'haproxy' has_
↪not been implemented for besu
  ## Any additional Ambassador ports can be given below, must be comma-separated_
↪without spaces, this is valid only if proxy='ambassador'
  # These ports are enabled per cluster, so if you have multiple clusters you do_
↪not need so many ports
  # This sample uses a single cluster, so we have to open 4 ports for each Node._
↪These ports are again specified for each organization below
  ambassadorPorts: 15010,15011,15012,15013,15020,15021,15022,15023,15030,15031,
↪15032,15033,15040,15041,15042,15043
  retry_count: 20 # Retry count for the checks on Kubernetes cluster
  external_dns: enabled # Should be enabled if using external-dns for_
↪automatic route configuration

# Docker registry details where images are stored. This will be used to create k8s_
↪secrets
# Please ensure all required images are built and stored in this registry.
# Do not check-in docker_password.
docker:
  url: "index.docker.io/hyperledgerlabs"
  username: "docker_username"
  password: "docker_password"

# Following are the configurations for the common Besu network
config:
  consensus: "ibft" # Options are "ibft". "ethash" and "clique"_
↪will be implemented in future release
  ## Certificate subject for the root CA of the network.
  # This is for development usage only where we create self-signed certificates_
↪and the truststores are generated automatically.
  # Production systems should generate proper certificates and configure_
↪truststores accordingly.
  subject: "CN=DLT Root CA,OU=DLT,O=DLT,L=London,C=GB"
  transaction_manager: "orion" # Transaction manager is "orion"
  # This is the version of "orion" docker image that will be deployed
  tm_version: "1.5.2"
  # TLS can be True or False for the orion tm
  tm_tls: True
  # Tls trust value
  tm_trust: "ca-or-tofu" # Options are: "whitelist", "ca-or-tofu",
↪"ca", "tofu"
  ## File location for saving the genesis file should be provided.
  genesis: "/home/user/blockchain-automation-framework/build/besu_genesis" #_
↪Location where genesis file will be fetched
  ## Transaction Manager nodes public addresses should be provided.
  # - "https://node.test.besu.blockchain-develop.com:15013"
  # The above domain name is formed by the (http or https):(peer.name).(org.
↪external_url_suffix):(ambassador orion node port)
  tm_nodes:
    - "https://carrier.test.besu.blockchaincloudpoc-develop.com:15013"
    - "https://manufacturer.test.besu.blockchaincloudpoc-develop.com:15023"

```

(continues on next page)

(continued from previous page)

```

- "https://store.test.besu.blockchaincloudpoc-develop.com:15033"
- "https://warehouse.test.besu.blockchaincloudpoc-develop.com:15043"
# Besu rpc public address list for existing validator and member nodes
# - "http://node.test.besu.blockchaincloudpoc-develop.com:15011"
# The above domain name is formed by the (http):(peer.name).(org.external_url_
↪suffix):(ambassador node rpc port)
besu_nodes:
- "http://validator.test.besu.blockchaincloudpoc-develop.com:15051"
- "http://carrier.test.besu.blockchaincloudpoc-develop.com:15011"
- "http://manufacturer.test.besu.blockchaincloudpoc-develop.com:15021"
- "http://store.test.besu.blockchaincloudpoc-develop.com:15031"

# Allows specification of one or many organizations that will be connecting to a
↪network.
organizations:
# Specification for the 1st organization. Each organization should map to a VPC
↪and a separate k8s cluster for production deployments
- organization:
name: neworg
type: member
# Provide the url suffix that will be added in DNS recordset. Must be different
↪for different clusters
# This is not used for Besu as Besu does not support DNS hostnames currently.
↪Here for future use
external_url_suffix: test.besu.blockchaincloudpoc.com
# List of all public IP addresses of each availability zone from all
↪organizations in the same k8s cluster
# The Ambassador will be set up using these static IPs. The child services will
↪be assigned the first IP in this list.
public_ips: ["3.221.78.194","21.23.74.154"] # List of all public IP addresses
↪of each availability zone from all organizations in the same k8s cluster
cloud_provider: aws # Options: aws, azure, gcp
aws:
access_key: "aws_access_key" # AWS Access key, only used when cloud_
↪provider=aws
secret_key: "aws_secret_key" # AWS Secret key, only used when cloud_
↪provider=aws
region: "aws_region" # AWS Region where cluster and EIPs are
↪created
# Kubernetes cluster deployment variables. The config file path and name has to
↪be provided in case
# the cluster has already been created.
k8s:
context: "cluster_context"
config_file: "cluster_config"
# Hashicorp Vault server address and root-token. Vault should be unsealed.
# Do not check-in root_token
vault:
url: "vault_addr"
root_token: "vault_root_token"
# Git Repo details which will be used by GitOps/Flux.
# Do not check-in git_access_token
gitops:
git_ssh: "ssh://git@github.com:<username>/blockchain-automation-framework.git
↪"
# Gitops ssh url for flux value files
branch: "develop" # Git branch where release is being made
release_dir: "platforms/hyperledger-besu/releases/dev" # Relative Path in the
↪Git repo for flux sync per environment.

```

(continues on next page)

(continued from previous page)

```

    chart_source: "platforms/hyperledger-besu/charts"      # Relative Path where
↳the Helm charts are stored in Git repo
    git_push_url: "github.com/<username>/blockchain-automation-framework.git" #
↳Gitops https URL for git push
    username: "git_username"          # Git Service user who has rights to check-
↳in in all branches
    password: "git_access_token"      # Git Server user password
    email: "git_email"               # Email to use in git config
    private_key: "path_to_private_key" # Path to private key file which
↳has write-access to the git repo
    # The participating nodes are named as peers
    services:
      peers:
        - peer:
            name: newOrg
            subject: "O=Neworg,OU=Neworg,L=51.50/-0.13/London,C=GB" # This is the node
↳subject. L=lat/long is mandatory for supplychain sample app
            geth_passphrase: 12345 # Passphrase to be used to generate geth account
            lock: false           # Sets Besu node to lock or unlock mode. Can be true or
↳false
          p2p:
            port: 30303
            ambassador: 15020      #Port exposed on ambassador service (use one port
↳per org if using single cluster)
          rpc:
            port: 8545
            ambassador: 15021      #Port exposed on ambassador service (use one port
↳per org if using single cluster)
          ws:
            port: 8546
            tm_nodeport:
              port: 15022          # Port exposed on ambassador service must be same
              ambassador: 15022
            tm_clientport:
              port: 8888

```

Three new sections are added to the network.yaml | Field | Description | |-----|

Field	Description
tm_nodes	Existing network's transaction manager nodes public addresses with nodeport provided as an array.
besu_nodes	Existing network's besu nodes public addresses with rpc port provided as an array.
genesis	Path to existing network's genesis.json in base64.

### 5.17.3 Run playbook

The `site.yaml` playbook is used to add a new organization to the existing network. This can be done using the following command

```

ansible-playbook platforms/shared/configuration/site.yaml --extra-vars "@path-to-
↳network.yaml" --extra-vars "add_new_org=True"

```



### 6.1 Quickstart Guides

#### 6.1.1 Developer Prerequisites

The following mandatory pre-requisites must be completed to set up a development environment for BAF. The process of setting up developer pre-requisites can be done manually or via an automation script (currently script is for windows OS only)

##### Script Based Setup

You can use the scripts [here](#) to setup developer prerequisites for Windows or Mac systems.

---

**NOTE:** You need to run the script with admin rights. This can be done by right clicking the script and selecting 'Run as administrator'.

---

##### Manual Setup

**The estimated total effort is 55 mins.**

---

**NOTE:** You will need at least 8GB RAM to run BAF on local machine.

---

## Setting up Git on your machine

*Estimated Time: 10 minutes*

To use Git, you need to install the software on your local machine.

1. Download and install **git bash** from <http://git-scm.com/downloads>.
2. Open 'git bash' (For Windows, Start > Run, C:\Program Files (x86)\Git\bin\sh.exe --login -i )
3. After the install has completed you can test whether Git has installed correctly by running the command `git --version`
4. If this works successfully you will need to configure your Git instance by specifying your username and email address. This is done with the following two commands (Use your GitHub username and email address, if you already have a Github Account):

```
git config --global user.name "<username>"
git config --global user.email "<useremail>"
```

5. To verify that the username and password was entered correctly, check by running

```
git config user.name
git config user.email
```

6. Windows users should additionally execute the following so that the EOLs are not updated to Windows CRLF.

```
git config --global core.autocrlf false
```

## Setting up Github

*Estimated Time: 5 minutes*

**GitHub** is a web-based Git repository hosting service. It offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. You can create projects and repositories for you and your teams' need.

Complete the following steps to download and configure BAF repository on your local machine.

1. If you already have an account from previously, you can use the same account. If you don't have an account, create one.
2. Go to [blockchain-automation-framework](#) on GitHub and click **Fork** button on top right. This will create a copy of the repo to your own GitHub account.
3. In git bash, write and execute the command:

```
ssh-keygen -q -N "" -f ~/.ssh/gitops
```

This generates an SSH key-pair in your user/.ssh directory: gitops (private key) and gitops.pub (public key).

4. Add the public key contents from gitops.pub (starts with ssh-rsa) as an Access Key (with read-write permissions) in your Github repository by following [this guide](#).
5. Execute the following command to add the key to your ssh-agent

```
eval "$(ssh-agent)"
ssh-add ~/.ssh/gitops
```

6. Create a `project` directory in your home directory and clone the forked repository to your local machine.

```
mkdir ~/project
cd ~/project
git clone git@github.com:<githubuser>/blockchain-automation-framework.git
```

7. Checkout the `develop` branch.

```
cd blockchain-automation-framework
git checkout develop
```

---

**NOTE:** If you have 2-Factor Authentication enabled on your GitHub account, you have to use GitHub token. Otherwise, password is fine.

1. On GitHub page, click your profile icon and then click **Settings**.
2. On the sidebar, click **Developer settings**.
3. On the sidebar, click **Personal access tokens**.
4. Click **Generate new token**.
5. Add a token description, enable suitable access and click **Generate token**.
6. Copy the token to a secure location or password management app.

For security reasons, after you leave the page, you can no longer see the token again.

---

## Setting up Docker

*Estimated Time: 10 minutes*

Install [Docker Toolbox](#) to make sure your local environment has the capability to execute `docker` commands. You can check the version of Docker you have installed with the following command from a terminal prompt:

```
docker --version
```

---

**NOTE:** For Windows, you **MUST** use Docker Toolbox with VirtualBox. Do not use Docker Desktop for Windows. Also HyperV should be **DISABLED** for Mac and Windows.

---

## Setting up HashiCorp Vault

*Estimated Time: 15 minutes*

We need [Hashicorp Vault](#) for the certificate and key storage.

1. To install the precompiled binary, [download](#) the appropriate package for your system.
2. Once the zip is downloaded, unzip it into any directory. The `vault` binary inside is all that is necessary to run Vault (or `vault.exe` for Windows). Any additional files, if any, aren't required to run Vault.
3. Create a directory `project/bin` and copy the binary there. Add `project/bin` directory to your `PATH`. Run following from git bash.

```
mkdir ~/project/bin
mv vault.exe ~/project/bin
export PATH=~/project/bin:$PATH
```

4. Create a `config.hcl` file in the `project` directory with the following contents (use a file path in the `path` attribute which exists on your local machine)

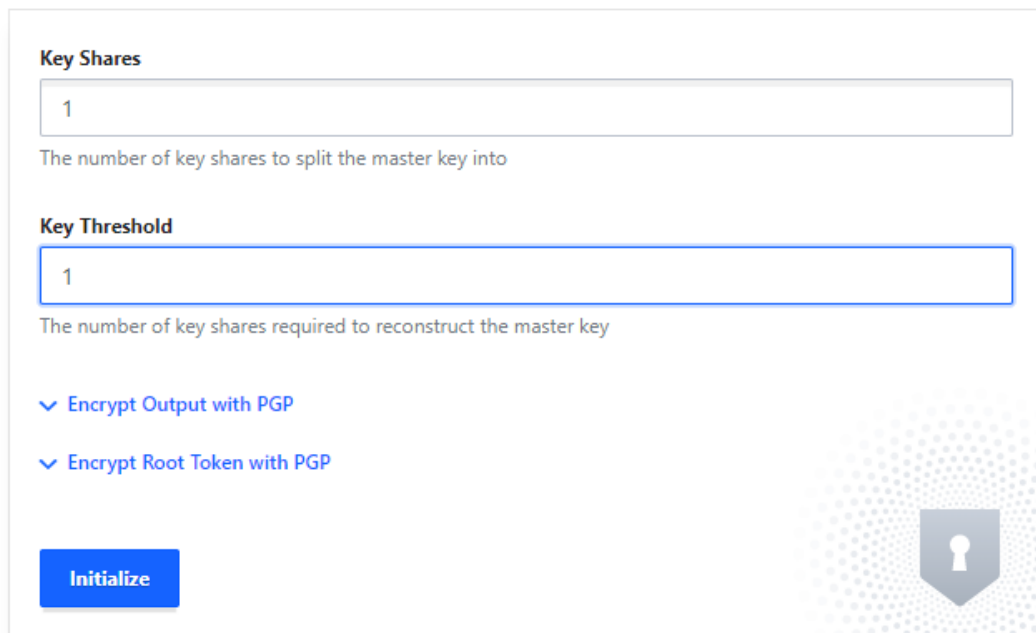
```
ui = true
storage "file" {
  path = "~/project/data"
}
listener "tcp" {
  address = "0.0.0.0:8200"
  tls_disable = 1
}
```

5. Start the Vault server by executing (this will occupy one terminal). Do not close this terminal.

```
vault server -config=config.hcl
```

6. Open browser at <http://localhost:8200/>. And initialize the Vault by providing your choice of key shares and threshold. (below example uses 1)

## Let's set up the initial set of master keys that you'll need in case of an emergency



**Key Shares**

The number of key shares to split the master key into

**Key Threshold**

The number of key shares required to reconstruct the master key

✓ Encrypt Output with PGP

✓ Encrypt Root Token with PGP

**Initialize**

7. Click **Download Keys** or copy the keys, you will need them. Then click **Continue to Unseal**. Provide the unseal key first and then the root token to login.
8. In a new terminal, execute the following (assuming `vault` is in your `PATH`):

```
export VAULT_ADDR='http://<Your Vault local IP address>:8200' #e.g. http://192.
↪168.0.1:8200
export VAULT_TOKEN="<Your Vault root token>"
vault secrets enable -version=1 -path=secret kv
```

## Setting up Minikube

*Estimated Time: 15 minutes*

For development environment, minikube can be used as the Kubernetes cluster on which the DLT network will be deployed.

1. Follow platform specific [instructions](#) to install minikube on your local machine. Also install [Virtualbox](#) as the Hypervisor. (If you already have **HyperV** it should be removed or disabled.)
2. minikube is also a binary, so move it into your ~/project/bin directory as it is already added to PATH.
3. Configure minikube to use 4GB memory and default kubernetes version

```
minikube config set memory 4096
minikube config set kubernetes-version v1.16.13
```

4. Then start minikube. This will take longer the first time.

```
minikube start --vm-driver=virtualbox
```

5. Check status of minikube by running

```
minikube status
```

The Kubernetes config file is generated at ~/.kube/config

6. To stop (do not delete) minikube execute the following

```
minikube stop
```

Now your development environment is ready!

**NOTE:** Minikube uses port in range 30000-32767. If you would like to change it, use the following command:

```
minikube start --vm-driver=virtualbox --extra-config=apiserver.service-node-port-
↪range=15000-20000
```

## Troubleshooting

At Step 5, if you get the following error:

```
2020-03-10T17:00:21.664Z [ERROR] core: failed to initialize barrier: error="failed to
↪persist keyring: mkdir /project: permission denied"
```

Update the path in Vault config.hcl to absolute path:

```
storage "file" {
  path = "/full/path/to/project/vault"
}
```

For example, /home/users/Desktop/project/vault.

## 6.1.2 Running BAF DLT network on Minikube

### Pre-requisites

Before proceeding, first make sure that you've completed [Developer Pre-requisites](#).

### Clone forked repo

1. If you have not already done so, fork [blockchain-automation-framework](#) and clone the forked repo to your machine.

```
cd ~/project
git clone git@github.com:<githubuser>/blockchain-automation-framework.git
```

2. Add a “local” branch to your machine

```
cd ~/project/blockchain-automation-framework
git checkout -b local
git push --set-upstream origin local
```

### Update kubeconfig file

1. Create a build folder inside your BAF repository:

```
cd ~/project/blockchain-automation-framework
mkdir build
```

2. Copy `ca.crt`, `client.key`, `client.crt` from `~/minikube` to build:

```
cp ~/.minikube/ca.crt build/
cp ~/.minikube/client.key build/
cp ~/.minikube/client.crt build/
```

3. Copy `~/kube/config` file to build:

```
cp ~/.kube/config build/
```

4. Open the above config file and remove the paths for certificate-authority, client-certificate and client-key as in the figure below.

```
apiVersion: v1
clusters:
- cluster:
  certificate-authority: ca.crt
  server: https://192.168.99.101:8443
  name: minikube
contexts:
- context:
  cluster: minikube
  user: minikube
  name: minikube
current-context: minikube
kind: Config
preferences: {}
users:
- name: minikube
  user:
    client-certificate: client.crt
    client-key: client.key
```

---

**NOTE:** If you ever delete and recreate minikube, the above steps have to be repeated.

---

5. Copy gitops file from ~/.ssh to build. (This is the private key file which you used to authenticate to your GitHub in pre-requisites)

```
cp ~/.ssh/gitops build/
```

### Additional Windows configurations

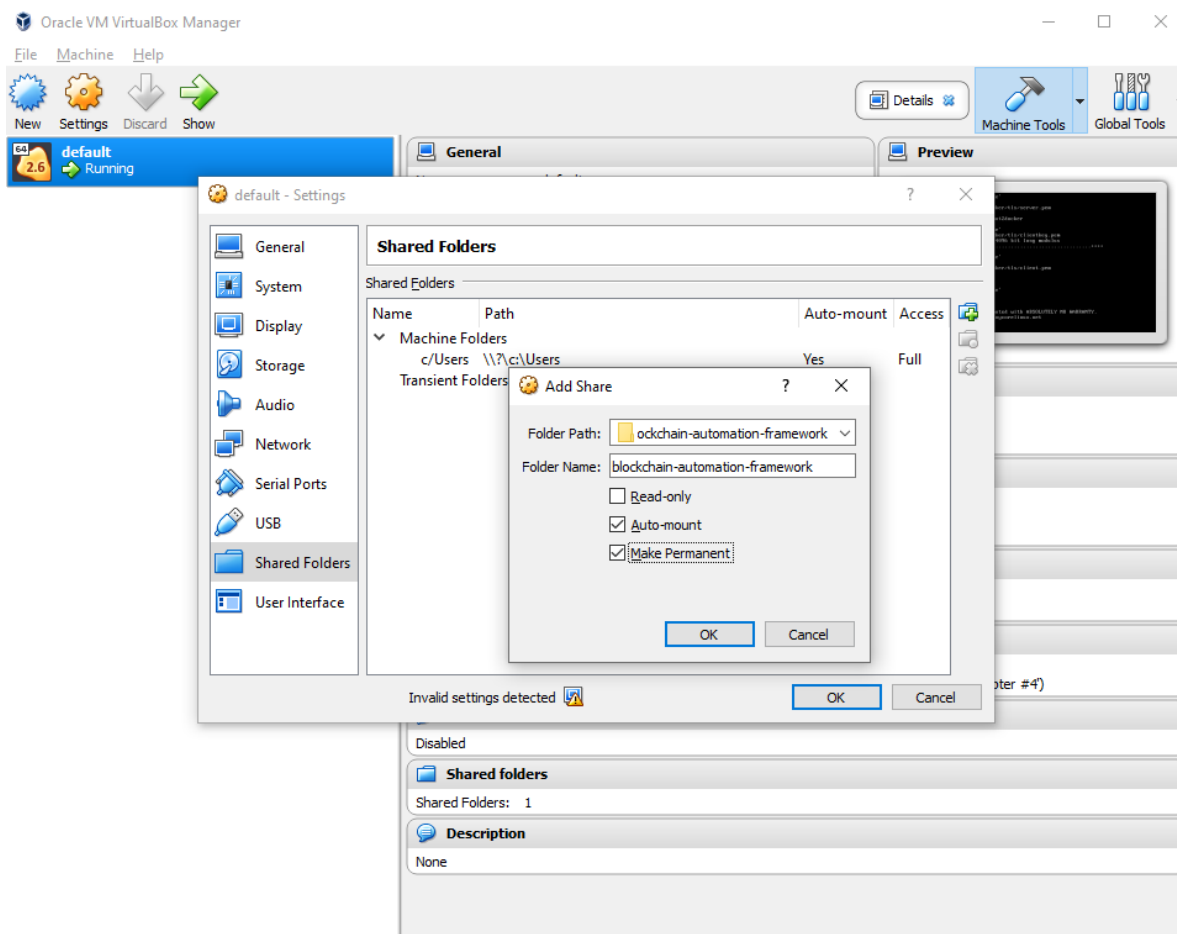
1. Ensure that you have set the following git config before cloning the repo.

```
git config --global core.autocrlf false
```

2. If not, update the EOL to LF for platforms/hyperledger-fabric/scripts/\*.sh files.
3. Execute following to correctly set docker environment.

```
eval $('docker-machine.exe' env)
```

4. Mount windows local folder (blockchain-automation-framework folder) to VirtualBox docker VM ( the machine named “default” by default) from right-click menu, Settings -> Shared Folders. All paths in network.yaml should be the mounted path. Shut down and restart the “default” machine after this.



## Edit the configuration file

1. Choose the DLT/Blockchain platform you want to run and copy the relevant sample network.yaml to build folder; rename it to network.yaml.

```
cd ~/project/blockchain-automation-framework
cp platforms/hyperledger-fabric/configuration/samples/network-minikube.yaml build/
↪ network.yaml
```

2. Update Docker configurations:

```
docker:
  url: "index.docker.io/hyperledgerlabs"
  username: "<your docker username>"
  password: "<your docker password/token>"
```

3. For each organization, update ONLY the following and leave everything else as-is:

```
vault:
  url: "http://<Your Vault local IP address>:8200" # Use the local IP address,
↪ rather than localhost e.g. http://192.168.0.1:8200
  root_token: "<your vault_root_token>"
gitops:
  git_ssh: "<ssh url of your forked repo>" #e.g. "ssh://git@github.com/
↪ hyperledger-labs/blockchain-automation-framework.git"
```

(continues on next page)

(continued from previous page)

```
git_push_url: "<https url of your forked repo without the https://>" #e.g.
→ "github.com/hyperledger-labs/blockchain-automation-framework.git"
username: "<github_username>"
password: "<github token/password>"
email: "<github_email>"
```

If you need help, you can use each platform's sample network-minikube.yaml:

- For Fabric, use `platforms/hyperledger-fabric/configuration/samples/network-minikube.yaml`
- For Quorum, use `platforms/quorum/configuration/samples/network-minikube.yaml`
- For Corda, use `platforms/r3-corda/configuration/samples/network-minikube.yaml`

And simply replace the placeholder values.

**NOTE:** If you have 2-Factor Authentication enabled on your GitHub account, you have to use GitHub token. Otherwise, password is fine.

1. On GitHub page, click your profile icon and then click **Settings**.
2. On the sidebar, click **Developer settings**.
3. On the sidebar, click **Personal access tokens**.
4. Click **Generate new token**.
5. Add a token description, enable suitable access and click **Generate token**.
6. Copy the token to a secure location or password management app.

For security reasons, after you leave the page, you can no longer see the token again.

1. Deploying the sample “supplychain” chaincode is optional, so you can delete the “chaincode” section. If deploying chaincode, update the following for the peers.

```
chaincode:
  repository:
    username: "<github_username>"
    password: "<github_token>"
```

## Execute

Make sure that Minikube and Vault server are running. Double-check by running:

```
minikube status
vault status
```

Now run the following to deploy BAF Fabric on minikube:

```
docker run -it -v $(pwd):/home/blockchain-automation-framework/ hyperledgerlabs/baf-
→build
```

Windows users should use following (make sure that the local volume was mounted as per *this step*):

```
docker run -it -v /blockchain-automation-framework:/home/blockchain-automation-  
↪framework/ hyperledgerlabs/baf-build
```

Meanwhile you can also check if pods are being deployed:

```
kubect1 get pods --all-namespaces -w
```

**NOTE:** If you need public address for nodes in your `network.yaml` file, you can use the output of `minikube ip`.

## Troubleshooting

### Failed to establish a new connection: [Errno 111] Connection refused

This is because you have re-created minikube but have not updated K8s `config` file. Repeat “*Update kubeconfig file*” steps 3 - 4 and try again.

**kubernetes.config.config\_exception.ConfigException: File does not exists:**  
**/Users/.minikube/ca.crt**

This is because you have not removed the absolute paths to the certificates in `config` file. Repeat “*Update kubeconfig file*” step 4 and try again.

**error during connect: Get http://%2F%2F.%2Fpipe%2Fdocker\_engine/v1.40/**  
**version: open //./pipe/docker\_engine: The system cannot find the file**  
**specified. In the default daemon configuration on Windows, the docker client**  
**must be run elevated to connect. This error may also indicate that the docker**  
**daemon is not running**

This is because docker isn’t running. To start it, just close all the instances of Docker Quickstart Terminal and open again.

**ERROR! the playbook: /home/blockchain-automation-framework/platforms/shared/**  
**configuration/site.yaml could not be found**

This is because the blockchain-automation-framework repository isn’t mounted to the default VM. Check *this step*).

## 6.1.3 DLT Blockchain Network deployment using docker build

### Building the docker image

Build the docker image using the docker file `Dockerfile` provided in the blockchain-automation-framework repository.

The docker image should be built from the root directory of the repository, the image builds the required base environment for BAF deployment and copies the a provisional script `run.sh` in the `WORKDIR`, it also sets a path for mounting volume

Following is a code snippet of the docker image showing `WORKDIR` and `VOLUME`

```
WORKDIR /home/
```

The `VOLUME` instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host

```
VOLUME /home/blockchain-automation-framework/
```

Use the below command to build the image

```
docker build . -t hyperledgerlabs/baf-build
```

This would create an image named hyperledgerlabs/baf-build

## Running the docker and provisioning script

A shell script `run.sh` is provided in repository to set the environment variables and run the network deployment playbook.

The BAF `Dockerfile` provides defaults for the executing container using the `CMD` variable

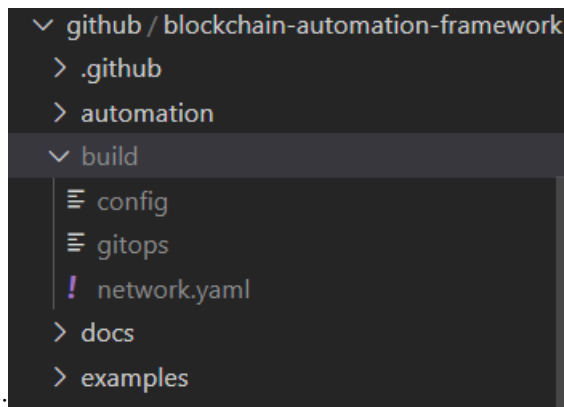
```
CMD ["/home/run.sh"]
```

Use the below command to run the container and the provisioning scripts, the command needs to be run from the root directory of the repository. The command also binds and mounts a volume, in this case it binds the repository

```
docker run -it -v $(pwd):/home/blockchain-automation-framework/ hyperledgerlabs/baf-  
↪build
```

Before running the above command add a build folder in the root directory of the repository, this build folder should have the following files:

1. K8s config file as `config`
2. Network specific configuration file as `network.yaml`
3. Private key file which has write-access to the git repo



Screen shot of the folder structure is below:

The paths in network configuration file should be changed accordingly.

## 6.2 Additional Developer prerequisites

- *Sphinx tool*
- *Molecule*

### 6.2.1 Sphinx tool

Sphinx is a tool that makes it easy to create intelligent and beautiful documentation. This tool is needed to build the Blockchain Automation Framework documentation from `docs` folder.

- Sphinx version used 2.1.1

**Sphinx installation:** Follow the [link](#) to install sphinx documentation tool.

All the Blockchain Automation Framework documentation and Sphinx Configuration files (`conf.py`) are located in `docs/source` folder. To build the documentation, execute the following command from `docs` directory:

```
make html
# or for Windows
.\Make.bat html
```

### 6.2.2 Molecule

[Molecule](#) is designed to aid in the development and testing of [Ansible](#) roles. In BAF, Molecule is used to check for common coding standards, yaml errors and unit testing Ansible code/roles.

- Molecule version used 2.22

#### Requirements

- Docker Engine
- Python3 (and pip configured with python3)

**Molecule installation** Please refer to the [Virtual environment](#) documentation for installation best practices. If not using a virtual environment, please consider passing the widely recommended `-user` flag when invoking `pip`.

```
$ pip install --user 'molecule[docker]'
```

The existing test scenarios are found in the `molecule` folder under configuration of each platform e.g. [platforms/shared/configuration/molecule](#) folder.

## 6.3 Ansible Roles and Playbooks

### 6.3.1 Common Configurations

The Blockchain Automation Framework installs the common pre-requisites when the `site.yaml` playbook is run. To read more about setting up DLT/Blockchain networks, refer [Setting up a Blockchain/DLT network](#).

Following playbooks can be executed independently to setup the environment and can be found [here](#)

1. **enviornment-setup.yaml** Playbook enviornment-setup.yaml executes the roles which has tasks to install the binaries for:
  - kubectl
  - helm
  - vault client
  - aws-authenticator
  - tiller

2. **kubernetes-env-setup.yaml** Playbook kubernetes-env-setup.yaml executes the roles which has tasks to configure the following on each Kubernetes cluster:

- flux
- ambassador (if chosen)
- haproxy-ingress (if chosen)

All the common Ansible roles can be found at [platforms/shared/configuration/roles](#)

- setup/ambassador
- setup/aws-auth
- setup/aws-cli
- setup/flux
- setup/haproxy-ingress
- setup/helm
- setup/kubectrl
- setup/tiller
- setup/vault

Follow [Readme](#) for detailed information on each of these roles.

### 6.3.2 Corda Configurations

In the Blockchain Automation Framework project, ansible is used to automate the certificate generation, putting them in vault and generate value files, which are then pushed to the repository for deployment, using GitOps. This is achieved using Ansible playbooks. Ansible playbooks contains a series of roles and tasks which run in sequential order to achieve the automation.

```
/r3-corda
|-- charts
|   |-- doorman
|   |-- doorman-tls
|   |-- h2
|   |-- h2-addUser
|   |-- h2-password-change
|   |-- mongodb
|   |-- mongodb-tls
|   |-- nms
|   |-- nms-tls
|   |-- node
|   |-- node-initial-registration
|   |-- notary
|   |-- notary-initial-registration
|   |-- storage
|-- images
|-- configuration
|   |-- roles/
|   |-- samples/
|   |-- playbook(s)
|   |-- openssl.conf
|-- releases
```

(continues on next page)

(continued from previous page)

```
| |-- dev/  
|-- scripts
```

For R3-Corda, the ansible roles and playbooks are located at `/platforms/r3-corda/configuration/`. Some of the common roles and playbooks between Hyperledger-Fabric, Hyperledger-Indy, Hyperledger-Besu, R3 Corda and Quorum are located at `/platforms/shared/configurations/`.

---

### Roles for setting up Corda Network

Roles in ansible are a combination of logically inter-related tasks.

Below is the single playbook that you need to execute to setup complete corda network.

#### deploy\_network

This is the main ansible playbook which call all the roles in below sequence to setup corda network.

- Create Storage Class
- Create namespace and vault auth
- Deploy Doorman service node
- Deploy Networkmap service node
- Check that orderer uri are reachable
- Deploy notary
- Deploy nodes
- Remove build directory

Follow [Readme](#) for detailed information.

Below are the roles that `deploy_network` playbook calls to complete the setup process.

#### setup/nms

- Perform all the prerequisites (namespace, Vault auth, rbac, imagepullsecret)
- Create nms helm value files
- Check-in to git repo

Follow [Readme](#) for detailed information.

#### setup/doorman

- Perform all the prerequisites (namespace, Vault auth, rbac, imagepullsecret)
- Create doorman and mongodb helm value files
- Check-in to git repo

Follow [Readme](#) for detailed information.

### **create/certificates**

- Generate root certificates for doorman and nms

Follow [Readme](#) for detailed information.

### **setup/notary**

- Perform all the prerequisites (namespace, Vault auth, rbac, imagepullsecret)
- Get crypto from doorman/nms, store in Vault
- Create notary db helm value files
- Create notary initial registration helm value files
- Create notary value files
- Check-in to git repo

Follow [Readme](#) for detailed information.

### **setup/node**

- Perform all the prerequisites (namespace, Vault auth, rbac, imagepullsecret)
- Get crypto from doorman/nms, store in Vault
- Create node db helm value files
- Create node initial registration helm value files
- Create node value files
- Check-in to git repo

Follow [Readme](#) for detailed information.

### **deploy/cordapps**

- Deploy cordapps into each node and notary

Follow [Readme](#) for detailed information.

### **setup/springboot\_services**

- Create springboot webserver helm value files for each node
- Check-in to git repo

Follow [Readme](#) for detailed information.

### **setup/get\_crypto**

- Ensure admincerts directory exists
- Save the cert file
- Save the key file

- Save the root keychain
- Save root cert
- Save root key

Follow [Readme](#) for detailed information.

### 6.3.3 Corda Enterprise Configurations

In the Blockchain Automation Framework project, Ansible is used to automate the certificate generation, putting them in vault and generate value files, which are then pushed to the git repository for deployment, using GitOps. This is achieved using Ansible playbooks. Ansible playbooks contains a series of roles and tasks which run in sequential order to achieve the automation. For R3-Corda Enterprise, the ansible roles and playbooks are located at `platforms/r3-corda-ent/configuration/` Some of the common roles and playbooks between Hyperledger-Fabric, Hyperledger-Indy, Hyperledger-Besu, R3 Corda and Quorum are located at `platforms/shared/configurations/`

```
platforms/r3-corda-ent/configuration
├── deploy-network.yaml
├── deploy-nodes.yaml
├── openssl.conf
├── README.md
├── reset-network.yaml
├── roles
│   ├── create
│   │   ├── certificates
│   │   ├── k8_component
│   │   ├── namespace_serviceaccount
│   │   └── storageclass
│   ├── delete
│   │   ├── flux_releases
│   │   ├── gitops_files
│   │   └── vault_secrets
│   ├── helm_component
│   │   ├── Readme.md
│   │   ├── tasks
│   │   ├── templates
│   │   └── vars
│   └── setup
│       ├── bridge
│       ├── cenm
│       ├── credentials
│       ├── float
│       ├── get_crypto
│       ├── idman
│       ├── nmap
│       ├── node
│       ├── node_registration
│       ├── notary
│       ├── notary-initial-registration
│       ├── pki-generator
│       ├── signer
│       ├── tlscerts
│       └── vault_kubernetes
└── samples
    ├── network-cordaent.yaml
    └── README.md
```

## Playbooks for setting up Corda Enterprise Network

Below are the playbooks available for the network operations.

### **deploy\_network.yaml**

This is the main ansible playbook which calls all the roles in the below sequence to setup Corda Enterprise network.

- Remove build directory
- Create Storage Class
- Create namespace and vault auth
- Deploy CENM services
- Check that orderer uri are reachable
- Deploy nodes

### **deploy\_nodes.yaml**

This ansible playbook should be used when deploying only the nodes. This can be used when the CENM Services are already up and managed by a different network.yaml. This calls the below supporting roles in sequence.

- Remove build directory
- Create Storage Class
- Create namespace and vault auth
- Check that orderer uri are reachable
- Deploy nodes

### **reset\_network.yaml**

This ansible playbook is used when deleting the network. This calls the below supporting roles in sequence.

- Deletes the Gitops release files
- Deletes the Vault secrets and authpaths
- Uninstalls Flux
- Deletes the helm releases from Kubernetes
- Remove build directory

Follow [Readme](#) for detailed information.

## Roles defined for Corda Enterprise

Roles in ansible are a combination of logically inter-related tasks. Below are the roles that are defined for Corda Enterprise.

### **create/certificates/cenm**

- Creates the Ambassador Proxy TLS Certificates for CENM components
- Saves them to Vault
- Creates Kubernetes secrets to be used by Ambassador pods

Follow [Readme](#) for detailed information.

### **create/certificates/node**

- Creates the Ambassador Proxy TLS Certificates for Corda Nodes
- Saves them to Vault
- Creates Kubernetes secrets to be used by Ambassador pods

Follow [Readme](#) for detailed information.

### **create/k8\_component**

- Creates various Kubernetes components based on the `templates`
- Checks-in to git repo

Add new `tpl` files in `templates` folder when defining new storageclass.

Follow [Readme](#) for detailed information.

### **create/namespace\_serviceaccount**

- Creates the namespace, serviceaccounts and clusterrolebinding
- Checks-in to git repo

### **create/storageclass**

- Creates the storageclass template with name “cordaentsc”
- Checks-in to git repo

Follow [Readme](#) for detailed information.

### **delete/flux\_releases**

- Deletes all helmreleases in the namespace
- Deletes the namespace

Follow [Readme](#) for detailed information.

### **delete/gitops\_files**

- Deletes all gitops files from release folder
- Checks-in to git repo

Follow [Readme](#) for detailed information.

### **delete/vault\_secrets**

- Deletes all contents of Vault
- Deletes the related Kubernetes secrets
- Deletes Vault access policies

Follow [Readme](#) for detailed information.

### **helm\_component**

- Creates various Helmrelease components based on the `templates`
- Performs helm lint (when true)

Most default values are in the tpl files in `templates` folder. If any need to be changed, these tpl files need to be edited.

Follow [Readme](#) for detailed information.

### **setup/bridge**

- Create helmrelease files for Bridge component
- Check-in to git repo

Follow [Readme](#) for detailed information.

### **setup/cenm**

- Checks all the prerequisite namespaces and serviceaccounts are created
- Creates vault access for cenm organization
- Calls `setup/pki-generator` role to generate network crypto.
- Calls `setup/signer` role to deploy signer service.
- Calls `setup/idman` role to deploy idman service.
- Calls `setup/nmap` role to deploy nmap service.
- Calls `setup/notary` role to deploy notary service.

Follow [Readme](#) for detailed information.

### setup/credentials

- Writes keystore,truststore,ssl passwords for CENM services
- Writes node keystore, node truststore, network root-truststore passwords for CENM services

Follow [Readme](#) for detailed information.

### setup/float

- Create helmrelease files for Float component
- Check-in to git repo

Follow [Readme](#) for detailed information.

### setup/get\_crypto

- Saves the Ambassador cert and key file to local file from Vault when playbook is re-run.

Follow [Readme](#) for detailed information.

### setup/idman

- Wait for Signer pod to be “Running”
- Creates Ambassador certs by calling create/certificates/cenm role
- Create idman value files
- Check-in to git repo

### setup/nmap

- Wait for PKI Job to “Complete” if certificates are not on Vault
- Creates Ambassador certs by calling create/certificates/cenm role
- Gets network-root-truststore.jks from Vault to save to local
- Create Notary-registration Job if not done already
- Wait forNotary-registration Job to “Complete” if not done already
- Create nmap value files
- Check-in to git repo

Follow [Readme](#) for detailed information.

### setup/node

- Wait for all the prerequisites (namespace, Vault auth, rbac, imagepullsecret)
- Create Vault access using setup/vault\_kubernetes role
- Create ambassador certificates by calling create/certificates/node
- Save idman/networkmap tls certs to Vault for this org

- Create node initial registration by calling `setup/node_registration` role
- Create node value files
- Create bridge, if enabled, by calling `setup/bridge`
- Create float, if enabled, by calling `setup/float`
- Check-in to git repo

Follow [Readme](#) for detailed information.

### **setup/node\_registration**

- Create node db helm value files
- Create node initial registration helm value files, if not registered already
- Check-in to git repo

Follow [Readme](#) for detailed information.

### **setup/notary**

- Wait for networkmap pod to be “Running”
- Create ambassador certificates by calling `create/certificates/cenm`
- Create notary value files
- Check-in to git repo

Follow [Readme](#) for detailed information.

### **setup/notary-initial-registration**

- Wait for idman pod to be “Running”
- Create notary db helm value files
- Create notary initial registration helm value files, if not registered already
- Check-in to git repo

Follow [Readme](#) for detailed information.

### **setup/pki-generator**

- Create pki-generator value files, if values are not in Vault
- Check-in to git repo

### **setup/signer**

- Wait for pki-generator Job to be “Completed”
- Create signer value files
- Check-in to git repo

Follow [Readme](#) for detailed information.

### setup/tlscerts

- Copies the idman/nmap certificates and truststore to each node's Vault

Follow [Readme](#) for detailed information.

### setup/vault\_kubernetes

- Creates vault auth path if it does not exist
- Gets Kubernetes ca certs
- Enables Kubernetes and Vault authentication
- Creates Vault policies if they do not exist
- Creates docker credentials if they do not exist

If the Vault policies need to be changed, then this role will need to be edited.

Follow [Readme](#) for detailed information.

## 6.3.4 Fabric Configurations

In the Blockchain Automation Framework project, ansible is used to automate the certificate generation, putting them in vault and generate value files, which are then pushed to the repository for deployment, using GitOps. This is achieved using Ansible playbooks. Ansible playbooks contains a series of roles and tasks which run in sequential order to achieve the automation.

```
/hyperledger-fabric
|-- charts
|   |-- ca
|   |-- catools
|   |-- zkkafka
|   |-- orderernode
|   |-- peernode
|   |-- create_channel
|   |-- join_channel
|   |-- install_chaincode
|   |-- instantiate_chaincode
|   |-- upgrade_chaincode
|-- images
|-- configuration
|   |-- roles/
|   |-- samples/
|   |-- playbook(s)
|   |-- openssl.conf
|-- releases
|   |-- dev/
|-- scripts
```

For Hyperledger-Fabric, the ansible roles and playbooks are located at `/platforms/hyperledger-fabric/configuration/`. Some of the common roles and playbooks between Hyperledger-Fabric, Hyperledger-Indy, Hyperledger-Besu, R3 Corda and Quorum are located at `/platforms/shared/configurations/`

## **Roles for setting up Fabric Network**

Roles in ansible are a combination of logically inter-related tasks.

Below is the single playbook that you need to execute to setup complete fabric network.

### **create/anchorpeer**

- Call nested\_anchorpeer for each organization
- Check join channel job is done
- Creating value file of anchor peer for {{ channel\_name }}
- Git Push

Follow [Readme](#) for detailed information.

### **create/ca\_server**

- Check if ca certs already created
- Ensures crypto dir exists
- Get ca certs and key
- Generate the CA certificate
- Copy the crypto material to Vault
- Check if ca admin credentials are already created
- Write the ca server admin credentials to Vault
- Check Ambassador cred exists
- Create the Ambassador credentials
- Create CA server values for Orderer
- Create CA server values for Organisations
- Git Push

Follow [Readme](#) for detailed information.

### **create/ca\_tools**

- Check CA-server is available
- Create CA-tools Values file
- Git Push

Follow [Readme](#) for detailed information.

### **create/chaincode/install**

- Create value file for chaincode installation
- Check/Wait for anchorpeer update job
- Check for install-chaincode job
- Write the git credentials to Vault
- Create value file for chaincode installation ( nested )
- Git Push

Follow [Readme](#) for detailed information.

### **create/chaincode/instantiate**

- Create value file for chaincode instantiation
- Check/Wait for install-chaincode job
- Check for instantiate-chaincode job
- Create value file for chaincode instantiation (nested)
- Git Push

Follow [Readme](#) for detailed information.

### **create/chaincode/invoke**

- Create value file for chaincode invocation
- Check/Wait for install-chaincode job
- Create value file for chaincode invocation (nested)
- Git Push

Follow [Readme](#) for detailed information.

### **create/chaincode/upgrade**

- Check/Wait for install-chaincode job
- Create value file for chaincode upgrade
- Git Push

Follow [Readme](#) for detailed information.

### **create/channel\_artifacts**

- Check configtxgen
- Getting the configtxgen binary tar
- Unzipping the downloaded file
- Moving the configtxgen from the extracted folder and place in its path

- Creating channel-artifacts folder
- Write BASE64 encoded genesis block to Vault
- Remove old channel block
- Creating channels
- Creating Anchor artifacts
- Creating JSON configuration for new organization

Follow [Readme](#) for detailed information.

### **create/genesis**

- Remove old genesis block
- Creating genesis block
- Write genesis block to Vault

Follow [README](#) for more information.

### **create/channels**

- Call valuefile when participant is creator
- Check orderer pod is up
- Check peer pod is up
- Create Create\_Channel value file
- Git Push

Follow [Readme](#) for detailed information.

### **create/channels\_join**

- Call nested\_channel\_join for each peer
- Check create channel job is done
- “join channel {{ channel\_name }}”
- Git Push
- Call check for each peer
- Check join channel job is done

Follow [Readme](#) for detailed information.

### **create/configtx**

- Remove old configtx file
- Create configtx file
- Adding init patch to configtx.yaml
- Adding organization patch to configtx.yaml

- Adding orderer patch to configtx.yaml
- Adding profile patch to configtx.yaml

Follow [Readme](#) for detailed information.

### **create/crypto/orderer**

- Call orderercheck.yaml for orderer
- Check if CA-tools is running
- Ensure ca directory exists
- Check if ca certs already created
- Check if ca key already created
- Call orderer.yaml for each orderer
- Check if orderer msp already created
- Get MSP info
- Check if orderer tls already created
- Ensure tls directory exists
- Get Orderer tls crt
- Create directory path on CA Tools
- Copy generate-usercrypto.sh to destination directory
- Changing the permission of msp files
- Copy the generate\_crypto.sh file into the CA Tools
- Generate crypto material for organization orderers
- Copy the crypto config folder from the ca tools
- Copy the crypto material for orderer
- Check Ambassador cred exists
- Check if orderer ambassador secrets already created
- Get Orderer ambassador info
- Generate the orderer certificate
- Create the Ambassador credentials
- Copy the crypto material to Vault

Follow [Readme](#) for detailed information.

### **create/crypto/peer**

- Check if CA-tools is running
- Ensure ca directory exists
- Check if ca certs already created
- Check if ca key already created

- Call peercheck.yaml for each peer
- Check if peer msp already created
- Get MSP info
- Call common.yaml for each peer
- Create directory path on CA Tools
- Copy generate-usercrypto.sh to destination directory
- Changing the permission of msp files
- Copy the generate\_crypto.sh file into the CA Tools
- Generate crypto material for organization peers
- Copy the crypto config folder from the ca tools
- Check that orderer-certificate file exists
- Ensure orderer tls cert directory exists
- Copy tls ca.crt from auto-generated path to given path
- Check if Orderer certs exist in Vault
- Save Orderer certs if not in Vault
- Copy organization level certificates for orderers
- Check if admin msp already created
- Copy organization level certificates for orgs
- Check if user msp already created
- Copy user certificates for orgs

Follow [Readme](#) for detailed information.

### **create/crypto\_script**

- Create generate\_crypto script file for orderers
- Create generate\_crypto script file for organizations

Follow [Readme](#) for detailed information.

### **create/namespace\_vaultauth**

- Check namespace is created
- Create namespaces
- Create vault reviewer service account for Organizations
- Create vault auth service account for Organizations
- Create clusterrolebinding for Orderers
- Git Push

Follow [Readme](#) for detailed information.

### **create/new\_organisation/create\_block**

- Call nested\_create\_json for each peer
- Ensure channel-artifacts dir exists
- Remove old anchor file
- Creating new anchor file
- adding new org peers anchor peer information
- Create create-block-{{ channel\_name }}.sh script file for new organisations

Follow [Readme](#) for detailed information.

### **create/orderers**

- create kafka clusters
- create orderers
- Git push

Follow [Readme](#) for detailed information.

### **create/peers**

- Write the couchdb credentials to Vault
- Create Value files for Organization Peers
- Git Push

Follow [Readme](#) for detailed information.

### **create/storageclass**

- Check if storage class created
- Ensures “*component\_type*” dir exists
- Create Storage class for Orderer
- Create Storage class for Organizations
- Git push

Follow [Readme](#) for detailed information.

### **delete/flux\_releases**

- Deletes all the helmreleases CRD
- Remove all Helm releases
- Deletes namespaces

Follow [Readme](#) for detailed information.

### **delete/gitops\_files**

- Delete release files
- Git push

Follow [Readme](#) for detailed information.

### **delete/vault\_secrets**

- Delete docker creds
- Delete Ambassador creds
- Delete vault-auth path
- Delete Crypto for orderers
- Delete Crypto for peers
- Delete policy

Follow [Readme](#) for detailed information.

### **helm\_component**

- Ensures value directory exist
- Create value file
- Helm lint

Follow [Readme](#) for detailed information.

### **k8\_component**

- Ensures value directory exist
- Create value file

Follow [Readme](#) for detailed information.

### **setup/config\_block/fetch**

- Call nested\_create\_cli for the peer
- create valuefile for cli {{ peer.name }}-{{ participant.name }}-{{ channel\_name }}
- Call nested\_fetch\_role for the peer
- start cli
- fetch and copy the configuration block from the blockchain
- delete cli

Follow [Readme](#) for detailed information.

### setup/config\_block/sign\_and\_update

- Call valuefile when participant is new
- Check peer pod is up
- Call nested\_sign\_and\_update for each peer
- create cli value files for {{peer.name}}-{{org.name}} for signing the modified configuration block
- start cli {{peer.name}}-{{org.name}}
- Check if fabric cli is present
- signing from the admin of {{org.name}}
- delete cli {{peer.name}}-{{participant.name}}-cli
- Call nested\_update\_channel for the peer
- start cli for {{peer.name}}-{{org.name}} for updating the channel
- Check if fabric cli is present
- updating the channel with the new configuration block
- delete cli {{peer.name}}-{{participant.name}}-cli

Follow [Readme](#) for detailed information.

### setup/get\_ambassador\_crypto

- Ensure ambassador secrets directory exists
- Save keys
- Save certs
- Ensure ambassador secrets directory exists
- Save keys
- Save certs
- signing from the admin of {{org.name}}
- delete cli {{peer.name}}-{{participant.name}}-cli
- Call nested\_update\_channel for the peer
- start cli for {{peer.name}}-{{org.name}} for updating the channel
- Check if fabric cli is present
- updating the channel with the new configuration block
- delete cli {{peer.name}}-{{participant.name}}-cli

### setup/get\_crypto

- Ensure admincerts directory exists
- Save admincerts
- Ensure cacerts directory exists

- Save cacerts
- Ensure tlscacerts directory exists
- Save tlscacerts

Follow [Readme](#) for detailed information.

### setup/vault\_kubernetes

- Check if namespace is created
- Ensures build dir exists
- Check if Kubernetes-auth already created for Organization
- Enable and configure Kubernetes-auth for Organization
- Get Kubernetes cert files for organizations
- Write reviewer token for Organisations
- Check if policy exists
- Create policy for Orderer Access Control
- Create policy for Organisations Access Control
- Write policy for vault
- Create Vault auth role
- Check docker cred exists
- Create the docker pull credentials

Follow [Readme](#) for detailed information.

## 6.3.5 Indy Configurations

In the Blockchain Automation Framework project, ansible is used to automate the certificate generation, putting them in vault and generate value files, which are then pushed to the repository for deployment, using GitOps. This is achieved using Ansible playbooks. Ansible playbooks contains a series of roles and tasks which run in sequential order to achieve the automation.

```
/hyperledger-indy
|-- charts
|   |-- indy-auth-job
|   |-- indy-cli
|   |-- indy-domain-genesis
|   |-- indy-domain-genesis
|   |-- indy-key-mgmt
|   |-- indy-ledger-txn
|   |-- indy-node
|   |-- indy-pool-genesis
|-- images
|-- configuration
|   |-- roles/
|   |-- samples/
|   |-- playbook(s)
|   |-- cleanup.yaml
```

(continues on next page)

(continued from previous page)

```
|-- releases
|   |-- dev/
|-- scripts
|   |-- indy_nym_txn
|   |-- setup indy cluster
```

For Hyperledger-Indy, the ansible roles and playbooks are located at `/platforms/hyperledger-indy/configuration/`. Some of the common roles and playbooks between Hyperledger-Fabric, Hyperledger-Indy, Hyperledger-Besu, R3 Corda and Quorum are located at `/platforms/shared/configurations/`.

---

## Roles for setting up Indy Network

Roles in ansible are a combination of logically inter-related tasks.

To deploy the indy network, run the `deploy-network.yaml` in `blockchain-automation-framework\platforms\hyperledger`. The roles included in the file are as follows.

### check/auth\_job

This role checks all auth jobs completing.

- Check if Indy auth job pod for trustee is completed
- Check if Indy auth job pod for stewards is completed
- Check if Indy auth job pod for endorser is completed
- Check if Indy auth job pod for baf-ac is completed

Follow [Readme](#) for detailed information.

### check/crypto

This role is checking if all crypto jobs are completed and all crypto data are in Vault.

- Check if Indy Key management pod for trustee is completed
- Check if Indy Key management pod for stewards is completed
- Check if Indy Key management pod for endorser is completed
- Check trustee in vault
- Check stewards in vault
- Check endorser in vault

Follow [Readme](#) for detailed information.

### check/endorser

This role check transaction job is done

- Check if endorser job pod is completed for organization `{{ component_name }}`

Follow [Readme](#) for detailed information.

### check/k8\_component

This role is used for waiting to kubernetes component.

- Wait for {{ component\_type }} {{ component\_name }}
- Wait for {{ component\_type }} {{ component\_name }}
- Wait for {{ component\_type }} {{ component\_name }}
- Get a ServiceAccount token for {{ component\_name }}
- Store token

Follow [Readme](#) for detailed information.

### check/validation

This role checks for validation of network.yaml

- Check Validation
  - Counting Genesis Steward
  - Set trustee count to zero
  - Counting trustees per Org
  - Print error and end playbook if trustee count limit fails
  - Counting Endorsers
  - Print error abd end playbook if endorser count limit fails
  - Reset Endorser count
- Print error and end playbook if genesis steward count limit fails
- Print error and end playbook if total trustee count limit fails

Follow [Readme](#) for detailed information.

### clean/flux

The role deletes the Helm release of Flux and git authentication secret from Kubernetes.

- Delete Helm release
- Wait for deleting of Helm release flux-{{ network.env.type }}

Follow [Readme](#) for detailed information.

### clean/gitops

This role deletes all the gitops release files

- Delete release files
- Git push

Follow [Readme](#) for detailed information.

### **clean/k8s\_resources**

The role deletes all running Kubernetes components and Helm releases of all organizations.

- Remove all Helm releases of organization {{ organization }}
- Get all existing Cluster Role Bindings of organization {{ organization }}
- Remove an existing Cluster Role Binding of {{ organization }}
- Remove an existing Namespace {{ organization\_ns }}
- Remove an existing Storage Class of {{ organization }}

Follow [Readme](#) for detailed information.

### **clean/vault**

This role get vault root token for organization and remove Indy crypto from vault

- Remove Indy Crypto of {{ organization }}
- Remove Policies of trustees
- Remove Policies of stewards
- Remove Policies of endorsers
- Remove Policies of {{ organization }}
- Remove Kubernetes Authentication Methods of {{ organization }}
- Remove Kubernetes Authentication Methods of {{ organization }} of trustees
- Remove Kubernetes Authentication Methods of {{ organization }} of stewards
- Remove Kubernetes Authentication Methods of {{ organization }} of endorsers

Follow [Readme](#) for detailed information.

### **create/helm\_component/auth\_job**

**This role create the job value file for creating Vault auth methods**

This role creates the job value file for stewards

- Ensures {{ release\_dir }}/{{ component\_type }}/{{ component\_name }} dir exists
- Get the kubernetes server url
- Trustee vault policy and role generating
- Stewards vault policy and role generating
- Endorser vault policy and role generating
- baf-ac vault policy and role generating

Follow [Readme](#) for detailed information.

### **create/helm\_component/crypto**

#### **This role create the job value file for creating Hyperledger Indy Crypto**

This role creates the job value file for stewards

- Ensures {{ release\_dir }}/{{ component\_type }}/{{ component\_name }} dir exists
- Trustee crypto generating
- Stewards crypto generating
- Endorser crypto generating

Follow [Readme](#) for detailed information.

### **create/helm\_component/domain\_genesis**

#### **This role create the config map value file for storing domain genesis for Indy cluster.**

This role creates the domain genesis file for organization

- Ensures {{ release\_dir }}/{{ component\_type }}/{{ component\_name }} dir exists
- Generate domain genesis for organization
- create value file for {{ component\_name }} {{ component\_type }}

Follow [Readme](#) for detailed information.

### **create/helm\_component/ledger\_txn**

#### **This role create the job value file for Indy NYM ledger transactions**

This role create the job value file for Indy NYM ledger transactions

- Ensures {{ release\_dir }}/{{ component\_type }}/{{ component\_name }} dir exists
- Create HelmRelease file
  - Ensures {{ release\_dir }}/{{ component\_type }}/{{ component\_name }} dir exists
  - Get identity data from vault
  - Inserting file into Variable
  - create value file for {{ new\_component\_name }} {{ component\_type }}
  - Delete file
  - Helm lint

Follow [Readme](#) for detailed information.

### **create/helm\_component/node**

#### **This role creates value file for Helm Release of stewards.**

This role creates the job value file for stewards

- Ensures {{ release\_dir }}/{{ component\_name }} dir exists
- create value file for {{ component\_name }} {{ component\_type }}

Follow [Readme](#) for detailed information.

### **create/helm\_component/pool\_genesis**

This role creates the pool genesis file for organization

- Ensures {{ release\_dir }}/{{ component\_type }}/{{ component\_name }} dir exists
- Generate pool genesis for organization
- create value file for {{ component\_name }} {{ component\_type }}

Follow [Readme](#) for detailed information.

### **create/imagepullsecret**

**This role creates secret in Kubernetes for pull docker images from repository.**

This role creates the docker pull registry secret within each namespace

- Check for ImagePullSecret for {{ organization }}
- Create the docker pull registry secret for {{ component\_ns }}

Follow [Readme](#) for detailed information.

### **create/k8\_component**

**This role create value file for kubernetes component by inserted type.**

This role generates value files for various k8 components

- Ensures {{ component\_type\_name }} dir exists
- create {{ component\_type }} file for {{ component\_type\_name }}

Follow [Readme](#) for detailed information.

### **create/namespace**

This role creates value files for namespaces of organizations

- Check namespace is created
- Create namespaces
- Git Push

Follow [Readme](#) for detailed information.

### **create/serviceaccount/by\_identities**

This role creates value files for service account

- Check if service account for {{ component\_name }} exists
- Create service account for {{ component\_name }}
- Check cluster role binding for {{ component\_name }}
- Get component\_name to var
- Get organization and admin string to var
- Create cluster role binding for {{ component\_name }}
- Create admin cluster role binding for {{ component\_name }}

Follow [Readme](#) for detailed information.

### **create/serviceaccount/main**

This role creates value files for service account for vault

- Create service account for trustees [{{ organization }}]
- Create service account for stewards [{{ organization }}]
- Create service account for endorsers [{{ organization }}]
- Create service account for organization [{{ organization }}]
- Create service account for read only public crypto [{{ organization }}]
- Push the created deployment files to repository
- Waiting for trustees accounts and cluster binding roles
- Waiting for stewards accounts and cluster binding roles
- Waiting for endorsers accounts and cluster binding roles
- Waiting for organization accounts and cluster binding roles
- Waiting for organization read only account and cluster binding role

Follow [Readme](#) for detailed information.

### **create/serviceaccount/waiting**

This role is waiting for create inserted ServiceAccounts or ClusterRoleBinding.

- Wait for creation for service account
- Wait for creation for cluster role binding

Follow [Readme](#) for detailed information.

### create/storageclass

This role creates value files for storage class

- Check if storageclass exists
- Create storageclass
- Push the created deployment files to repository
- Wait for Storageclass creation for {{ component\_name }}

Follow [Readme](#) for detailed information.

### setup/auth\_job

This role generates Helm releases of kubernetes jobs, which create Auth Methods into HashiCorp Vault for getting Vault token by Kubernetes Service Accounts

- Wait for namespace creation for stewards
- Create auth\_job of stewards, trustee and endorser
- Push the created deployment files to repository
- Check if auth job finished correctly

Follow [Readme](#) for detailed information.

### setup/crypto

This role creates the deployment files for indy crypto generate job and pushes them to repository

- Wait for namespace creation for stewards
- Create image pull secret for stewards
- Create crypto of stewards, trustee and endorser
- Push the created deployment files to repository
- Check Vault for Indy crypto

Follow [Readme](#) for detailed information.

### setup/domain\_genesis

This role creates the values files for organizations domain genesis and pushes them to repository

- Create domain genesis
- Push the created deployment files to repository
- Wait until domain genesis configmap are created

Follow [Readme](#) for detailed information.

### **setup/endorsers**

This role creates the deployment files for endorsers and pushes them to repository

- Wait for namespace creation
- Create image pull secret for identities
- Create Deployment files for Identities
  - Select Admin Identity for Organisation {{ component\_name }}
  - Inserting file into Variable
  - Calling Helm Release Development Role...
  - Delete file
  - Push the created deployment files to repository
- Wait until identities are creating

Follow [Readme](#) for detailed information.

### **setup/node**

This role creates the deployment files for stewards and pushes them to repository

- Wait for namespace creation for stewards
- Create image pull secret for stewards
- Create steward deployment file
- Push the created deployment files to repository
- Wait until steward pods are running

Follow [Readme](#) for detailed information.

### **setup/pool\_genesis**

This role creates the values files for organizations domain genesis and pushes them to repository

- Create pool genesis
- Push the created deployment files to repository
- Wait until pool genesis configmap are created

Follow [Readme](#) for detailed information.

### **setup/vault\_kubernetes**

This role setups communication between the vault and kubernetes cluster and install necessary configurations.

- Check namespace is created
- Ensures build dir exists
- Check if Kubernetes-auth already created for Organization
- Enable and configure Kubernetes-auth for Organization

- Get Kubernetes cert files for organizations
- Write reviewer token for Organizations
- Check if policy exists
- Create policy for Access Control
- Write Policy to Vault
- Create Vault auth role

Follow [Readme](#) for detailed information.

### 6.3.6 Quorum Configurations

In the Blockchain Automation Framework project, ansible is used to automate the certificate generation, putting them in vault and generate value files, which are then pushed to the repository for deployment, using GitOps. This is achieved using Ansible playbooks. Ansible playbooks contains a series of roles and tasks which run in sequential order to achieve the automation.

```
/quorum
|-- charts
|   |-- node_constellation
|   |-- node_tessera
|-- images
|-- configuration
|   |-- roles/
|   |-- samples/
|   |-- deploy-network.yaml
|-- releases
|   |-- dev/
|-- scripts
```

For Quorum, the ansible roles and playbooks are located at `/platforms/quorum/configuration/`. Some of the common roles and playbooks between Hyperledger-Fabric, Hyperledger-Indy, Hyperledger-Besu, R3 Corda and Quorum are located at `/platforms/shared/configurations/`

---

### Roles for setting up a Quorum Network

Roles in ansible are a combination of logically inter-related tasks.

To deploy the quorum network, run the `deploy-network.yaml` in `blockchain-automation-framework\platforms\quorum\`. The roles included in the file are as follows:

#### **\*\*check/k8\_component**

This role checks for the k8s resources in the cluster

- Wait for `{{ component_type }}` `{{ component_name }}`
- Wait for `{{ component_type }}` `{{ component_name }}` Follow [Readme](#) for detailed information.

**\*\*check/node\_component**

This role checks for the k8s resources in the cluster

- Wait for {{ component\_type }} {{ component\_name }}
- Wait for {{ component\_type }} {{ component\_name }} Follow [Readme](#) for detailed information.

**create/certificates/ambassador**

This role calls for ambassador certificate creation for each node.

- Create Ambassador certificates
- Ensure rootca dir exists
- Ensure ambassador tls dir exists
- Check if certs already created
- Get root certs
- check root certs
- Generate CAroot certificate
- Check if ambassador tls already created
- Get ambassador tls certs
- Generate openssl conf file
- Generate ambassador tls certs
- Putting certs to vault
- Check Ambassador cred exists
- Create the Ambassador credentials Follow [Readme](#) for detailed information.

**create/crypto/constellation**

This role creates crypto for constellation.

- Create Crypto material for each node for constellation
- Check tm key is present the vault
- Create build directory
- Generate Crypto for constellation
- Copy the crypto into vault

Follow [Readme](#) for detailed information.

**create/crypto/ibft**

This role creates crypto for ibft.

- Create crypto material for each peer with IBFT consensus
- Check if nodekey already present in the vault

- Create build directory if it does not exist
- Generate enode url for each node and create a geth account and keystore
- Copy the crypto material to Vault

Follow [Readme](#) for detailed information.

### **create/crypto/raft**

This role creates crypto for raft.

- Create crypto material for each peer with RAFT consensus
- Check if nodekey already present in the vault
- Create build directory if it does not exist
- Generate crypto for raft consensus
- Copy the crypto material to Vault

Follow [Readme](#) for detailed information.

### **create/crypto/tessera**

This role creates crypto for tessera.

- Create tessera tm crypto material for each peer
- Check if tm key is already present in the vault
- Create build directory if it does not exist
- Check if tessera jar file exists
- Download tessera jar
- Generate node tm keys
- Copy the crypto material to Vault

Follow [Readme](#) for detailed information.

### **create/genesis\_nodekey**

This role creates genesis nodekey.

- Check if nodekey is present in vault
- Call nested check for each node
- Check if nodekey already present in the vault
- vault\_check variable
- Fetching data of validator nodes in the network from network.yaml
- Get validator node data
- Create build directory if it does not exist
- Generate istanbul files
- Rename the directories created above with the elements of validator\_node\_list

- Delete the numbered directories

Follow [Readme](#) for detailed information.

### **create/k8\_component**

This role creates deployment files for nodes, namespace storageclass, service accounts and clusterrolebinding. Deployment file for a node is created in a directory with name=nodeName, nodeName is stored in component\_name

- “Ensures {{ release\_dir }}/{{ component\_name }} dir exists”
- create {{ component\_type }} file for {{ component\_name }}
- Helm lint

Follow [Readme](#) for detailed information.

### **create/namespace\_serviceaccount**

This role creates the deployment files for namespaces, vault-auth, vault-reviewer and clusterrolebinding for each node

- Check if namespace exists
- Create namespace for {{ organisation }}
- Create vault auth service account for {{ organisation }}
- Create vault reviewer for {{ organisation }}
- Create clusterrolebinding for {{ organisation }}
- Push the created deployment files to repository

Follow [Readme](#) for detailed information.

### **create/storageclass**

This role creates value files for storage class

- Check if storageclass exists
- Create storageclass
- Push the created deployment files to repository
- Wait for Storageclass creation for {{ component\_name }}

Follow [Readme](#) for detailed information.

### **create/tessera**

- Set enode\_data\_list to []
- Get enode data for each node of all organization
- Get enode data
- Check if enode is present in the build directory or not
- Create build directory if it does not exist
- Get the nodekey from vault and generate the enode

- Get enode\_data
- Get validator node data
- Git Push

Follow [Readme](#) for detailed information.

### helm\_component

This role generates the value file for the helm releases.

- Ensures {{ values\_dir }}/{{ name }} dir exists
- create value file for {{ component\_name }}
- Helm lint

Follow [Readme](#) for detailed information.

### setup/bootnode

This role is used to setup bootnode.

- Check bootnode
- Check quorum repo dir exists
- Clone the git repo
- Make bootnode
- Create bin directory
- Copy bootnode binary to destination directory

Follow [Readme](#) for detailed information.

### setup/constellation-node

This role is used to setup constellation-node.

- Register temporary directory
- check constellation
- Finding the release for os
- Release version
- Download the constellation-node binary
- Unarchive the file.
- Create the bin directory
- This task puts the constellation-node binary into the bin directory

Follow [Readme](#) for detailed information.

### **setup/get\_crypto**

This role saves the crypto from Vault into ansible\_provisioner.

- Ensure directory exists
- Save cert
- Save key
- Save root keychain
- Extracting root certificate from .jks

Follow [Readme](#) for detailed information.

### **setup/geth**

This role setups geth.

- Check geth
- Check quorum repo dir exists
- Clone the git repo
- Make geth
- Create bin directory
- Copy geth binary to destination directory

Follow [Readme](#) for detailed information.

### **setup/golang**

This role setups geth.

- Register temporary directory
- Check go
- Download golang tar
- Extract the Go tarball
- Create bin directory
- Copy go binary to destination directory
- Test go installation

Follow [Readme](#) for detailed information.

### **setup/istanbul**

This role setups istanbul.

- Register temporary directory
- Check istanbul
- Clone the istanbul-tools git repo

- Make istanbul
- Create bin directory
- Copy istanbul binary to destination directory

Follow [Readme](#) for detailed information.

### **setup/vault\_kubernetes**

This role setups communication between the vault and kubernetes cluster and install necessary configurations.

- Check namespace is created
- Ensures build dir exists
- Check if Kubernetes-auth already created for Organization
- Vault Auth enable for organisation
- Get Kubernetes cert files for organizations
- Write reviewer token
- Check if secret-path already created for Organization
- Create Vault secrets path
- Check if policy exists
- Create policy for Access Control
- Create Vault auth role
- Create the docker pull credentials

Follow [Readme](#) for detailed information.

### **delete/flux\_releases**

This role deletes the helm releases and uninstalls Flux

- Uninstall flux
- Delete the helmrelease for each peer
- Remove node helm releases
- Deletes namespaces

Follow [Readme](#) for detailed information.

### **delete/gitops\_files**

This role deletes all the gitops release files

- Delete release files
- Delete release files (namespaces)
- Git Push

Follow [Readme](#) for detailed information.

### delete/vault\_secrets

This role deletes the Vault configurations

- Delete docker creds
- Delete Ambassador creds
- Delete vault-auth path
- Delete Crypto material
- Delete Access policies

Follow [Readme](#) for detailed information.

### deploy-network.yaml

This playbook deploys a DLT/Blockchain network on existing Kubernetes clusters. The Kubernetes clusters should already be created and the information to connect to the clusters be updated in the network.yaml file that is used as an input to this playbook. It calls the following roles.

- create/namespace\_serviceaccount
- create/storageclass
- setup/vault\_kubernetes
- create/certificates/ambassador
- create/crypto/raft
- create/genesis\_raft
- setup/istanbul
- create/genesis\_nodekey
- create/crypto/ibft
- create/crypto/tessera
- create/crypto/constellation
- create/tessera
- create/constellation

### reset-network.yaml

This playbook deletes the DLT/Blockchain network on existing Kubernetes clusters which has been created using the Blockchain Automation Framework. It calls the following roles. **THIS PLAYBOOK DELETES EVERYTHING, EVEN NAMESPACES and FLUX.**

- delete/vault\_secrets
- delete/flux\_releases
- delete/gitops\_files
- Remove build directory

### 6.3.7 Hyperledger Besu Configurations

In the Blockchain Automation Framework project, ansible is used to automate the certificate generation, put them in the vault and generate value files, which are then pushed to the repository for deployment, using GitOps. This is achieved using Ansible playbooks. Ansible playbooks contains a series of roles and tasks which run in sequential order to achieve the automation.

```
/hyperledger-besu
|-- charts
|   |-- node_orion
|-- images
|-- configurations
|   |-- roles/
|   |-- samples/
|   |-- deploy-network.yaml
|-- releases
|   |-- dev/
|-- scripts
```

For Hyperledger Besu, the ansible roles and playbooks are located at `/platforms/hyperledger-besu/configuration/`. Some of the common roles and playbooks between Hyperledger-Fabric, Hyperledger-Indy, Hyperledger-Besu, R3 Corda and Quorum are located at `/platforms/shared/configurations/`

---

#### Roles for setting up a Hyperledger Besu Network

Roles in ansible are a combination of logically inter-related tasks.

To deploy the Hyperledger-Besu network, run the `deploy-network.yaml` in `blockchain-automation-framework\platforms\hyperledger-besu\configuration\`. The roles included in the files are as follows:

##### **create/certificates/ambassador**

This role calls for ambassador certificate creation for each node.

- Create Ambassador certificates
- Ensure rootca dir exists
- Ensure ambassador tls dir exists
- Check if certs already created
- Get root certs
- check root certs
- Generate CAroot certificate
- Check if ambassador tls already created
- Get ambassador tls certs
- Generate openssl conf file
- Generate ambassador tls certs
- Putting certs to vault

- Check Ambassador cred exists
- Create the Ambassador credentials Follow [Readme](#) for detailed information.

### **create/crypto/ibft**

This role creates crypto for ibft.

- Create crypto material for each peer with IBFT consensus
- Check if nodekey already present in the vault
- Create build directory if it does not exist
- Generate enode url for each node and create a geth account and keystore
- Copy the crypto material to Vault

Follow [Readme](#) for detailed information.

### **create/k8\_component**

This role creates deployment files for nodes, namespace storageclass, service accounts and clusterrolebinding. Deployment file for a node is created in a directory with name=nodeName, nodeName is stored in component\_name

- “Ensures {{ release\_dir }}/{{ component\_name }} dir exists”
- create {{ component\_type }} file for {{ component\_name }}
- Helm lint

Follow [Readme](#) for detailed information.

### **create/namespace\_serviceaccount**

This role creates the deployment files for namespaces, vault-auth, vault-reviewer and clusterrolebinding for each node

- Check if namespace exists
- Create namespace for {{ organisation }}
- Create vault auth service account for {{ organisation }}
- Create vault reviewer for {{ organisation }}
- Create clusterrolebinding for {{ organisation }}
- Push the created deployment files to repository

Follow [Readme](#) for detailed information.

### **create/storageclass**

This role creates value files for storage class

- Check if storageclass exists
- Create storageclass
- Push the created deployment files to repository
- Wait for Storageclass creation for {{ component\_name }}

Follow [Readme](#) for detailed information.

### **setup/get\_crypto**

This role saves the crypto from Vault into ansible\_provisioner.

- Ensure directory exists
- Save cert
- Save key
- Save root keychain
- Extracting root certificate from .jks

Follow [Readme](#) for detailed information.

### **setup/vault\_kubernetes**

This role setups communication between the vault and kubernetes cluster and install necessary configurations.

- Check namespace is created
- Ensures build dir exists
- Check if Kubernetes-auth already created for Organization
- Vault Auth enable for organisation
- Get Kubernetes cert files for organizations
- Write reviewer token
- Check if secret-path already created for Organization
- Create Vault secrets path
- Check if policy exists
- Create policy for Access Control
- Create Vault auth role
- Create the docker pull credentials

Follow [Readme](#) for detailed information.

## **6.4 Helm Charts**

### **6.4.1 Common Charts**

### **6.4.2 Corda Charts**

The structure below represents the Chart structure for R3 Corda components in the Blockchain Automation Framework implementation.

```
/r3-corda
|-- charts
|   |-- doorman
|   |-- doorman-tls
|   |-- h2
|   |-- h2-addUser
|   |-- h2-password-change
|   |-- mongodb
|   |-- mongodb-tls
|   |-- nms
|   |-- nms-tls
|   |-- node
|   |-- node-initial-registration
|   |-- notary
|   |-- notary-initial-registration
|   |-- storage
```

---

## Pre-requisites

helm to be installed and configured on the cluster.

## doorman

### About

This folder consists Doorman helm charts which are used by the ansible playbooks for the deployment of Doorman component. The folder contains a templates folder,a chart file and a value file.

### Folder Structure

```
/doorman
|-- templates
|   |-- pvc.yaml
|   |-- deployment.yaml
|   |-- service.tpl
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for Doorman implementation.
- This folder contains following template files for Doorman implementation

- deployment.yaml: This file is used as a basic manifest for creating a Kubernetes deployment for Doorman . The file basically describes the container and volume specifications of the Doorman. The file defines container where doorman container is defined with corda image and corda jar details. The init container init-creds creates doorman db root password and user credentials at mount path , init-certificates init container basically configures doorman keys.jks by fetching certsecretprefix from the vault, change permissions init-containers provides permissions to base directory and db-healthcheck init-container checks for db is up or not.
- pvc.yaml: This yaml is used to create persistent volumes claim for the Doorman deployment.A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates persistentVolumeClaim for Doorman pvc.
- service.yaml: This template is used as a basic manifest for creating a service endpoint for our deployment. This service.yaml creates ca service endpoint.The file basically specifies service type and kind of service ports for doorman server.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## doorman-tls

### About

This folder consists Doorman helm charts which are used by the ansible playbooks for the deployment of Doorman component when TLS is on for the doorman. The folder contains a templates folder, a chart file and a value file.

### Folder Structure

```
/doorman-tls
|-- templates
|   |-- pvc.yaml
|   |-- deployment.yaml
|   |-- service.tpl
|-- Chart.yaml
|-- values.yaml
```

### Charts description

#### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for Doorman implementation.

- This folder contains following template files for Doorman implementation
  - deployment.yaml: This file is used as a basic manifest for creating a Kubernetes deployment for Doorman . The file basically describes the container and volume specifications of the Doorman. The file defines container where doorman container is defined with corda image and corda jar details. The init container init-creds creates doorman db root password and user credentials at mount path , init-certificates init container basically configures doorman keys.jks by fetching certsecretprefix from the vault, change permissions init-containers provides permissions to base directory and db-healthcheck init-container checks for db is up or not.
  - pvc.yaml: This yaml is used to create persistent volumes claim for the Doorman deployment.A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates persistentVolumeClaim for Doorman pvc.
  - service.yaml: This template is used as a basic manifest for creating a service endpoint for our deployment. This service.yaml creates ca service endpoint.The file basically specifies service type and kind of service ports for doorman server.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion, name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## nms

### About

This folder consists networkmapservice helm charts which are used by the ansible playbooks for the deployment of networkmapservice component. The folder contains a templates folder,a chart file and a value file.

### Folder Structure

```
/nms
|-- templates
|   |-- volume.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

### Charts description

## templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for nms implementation.
- This folder contains following template files for nms implementation
  - deployment.yaml: This file is used as a basic manifest for creating a Kubernetes deployment for NMS . The file basically describes the container and volume specifications of the NMS. The file defines containers where NMS container is defined with corda image and corda jar details. The init container init-certificates-creds creates NMS db root password and user credentials at mount path, init-certificates init container basically configures NMS keys.jks by fetching certsecretprefix from the vault, changepermissions init-containers provides permissions to base directory and db-healthcheck init-container checks for db is up or not.
  - service.yaml: This template is used as a basic manifest for creating a service endpoint for our deployment. This service.yaml creates nms service endpoint.The file basically specifies service type and kind of service ports for the nms server.
  - volume.yaml: This yaml is used to create persistent volumes claim for the nms deployment.A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates nms pvc for, the volume claim for nms.

## Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

## values.yaml

- This file contains the default configuration values for the chart.
- 

## nms-tls

### About

This folder consists networkmapservice helm charts that establishes a TLS connection with mongodb, which are used by the ansible playbooks for the deployment of networkmapservice component. This chart is deployed when TLS is on for nms. The folder contains a templates folder, a chart file and a value file.

### Folder Structure

```
/nms-tls
|-- templates
|   |-- volume.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for nms implementation.
- This folder contains following template files for nms implementation
  - deployment.yaml: This file is used as a basic manifest for creating a Kubernetes deployment for NMS . The file basically describes the container and volume specifications of the NMS. The file defines containers where NMS container is defined with corda image and corda jar details. The init container init-certificates-creds creates NMS db root password and user credentials at mount path, init-certificates init container basically configures NMS keys.jks by fetching certsecretprefix from the vault, changepermissions init-containers provides permissions to base directory and db-healthcheck init-container checks for db is up or not.
  - service.yaml: This template is used as a basic manifest for creating a service endpoint for our deployment. This service.yaml creates nms service endpoint. The file basically specifies service type and kind of service ports for the nms server.
  - volume.yaml: This yaml is used to create persistent volumes claim for the nms deployment. A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates nms pvc for, the volume claim for nms.

### Chart.yaml

- This file contains the information about the chart such as appversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## h2 (database)

### About

This folder consists H2 helm charts which are used by the ansible playbooks for the deployment of the H2 database. The folder contains a templates folder, a chart file and a value file.

### Folder Structure

```
/h2
|-- templates
|   |-- pvc.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

### Charts description

#### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for H2 implementation.
- This folder contains following template files for H2 implementation
  - deployment.yaml: This file is used as a basic manifest for creating a Kubernetes deployment.For the H2 node, this file creates H2 deployment.
  - service.yaml: This template is used as a basic manifest for creating a service endpoint for our deployment.This service.yaml creates H2 service endpoint
  - pvc.yaml: This yaml is used to create persistent volumes claim for the H2 deployment.A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates h2-pvc for , the volume claim for H2.

#### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion , name etc.

#### values.yaml

- This file contains the default configuration values for the chart.
- 

### h2-addUser

#### About

This folder consists H2-adduser helm charts which are used by the ansible playbooks for the deployment of the Peer component. The folder contains a templates folder,a chart file and a value file.

#### Folder Structure

```
/h2-addUser
|-- templates
|   |-- job.yaml
|-- Chart.yaml
|-- values.yaml
```

#### Pre-requisites

helm to be installed and configured

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for h2 add user implementation.
- This folder contains following template file for adding users to h2 implementation
  - job.yaml: The job.yaml file through template engine runs create h2-add-user container and thus runs newuser.sql to create users and create passwords for new users.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## h2-password-change

### About

This folder consists H2-password-change helm charts which are used by the ansible playbooks for the deployment of the Peer component. The folder contains a templates folder,a chart file and a value file.

### Folder Structure

```
/h2-password-change
|-- templates
|   |-- job.yaml
|-- Chart.yaml
|-- values.yaml
```

### Pre-requisites

helm to be installed and configured

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for h2 password change implementation.

- This folder contains following template file for changing h2 password implementation
  - job.yaml: The job.yaml file through template engine runs create h2-add-user container and thus runs newuser.sql to create users and create passwords for new users.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## mongodb

### About

This folder consists MongoDB helm charts which are used by the ansible playbooks for the deployment of the MongoDB component. The folder contains a templates folder, a chart file and a value file.

### Folder Structure

```
/mongodb
|-- templates
|   |-- pvc.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

### Charts description

#### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for MongoDB implementation.
- This folder contains following template files for MongoDB implementation
  - deployment.yaml: This file is used as a basic manifest for creating a Kubernetes deployment.For the MongoDB node, this file creates MongoDB deployment.
  - service.yaml: This template is used as a basic manifest for creating a service endpoint for our deployment.This service.yaml creates MongoDB service endpoint
  - pvc.yaml: This yaml is used to create persistent volumes claim for the MongoDB deployment.A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates mongodb-pvc for , the volume claim for mongodb.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## mongodb-tls

### About

This folder consists MongoDB helm charts which are used by the ansible playbooks for the deployment of the MongoDB component. It allows for TLS connection. When TLS is on for nms or doorman, this chart is deployed for them else mongodb chart is deployed. The folder contains a templates folder, a chart file and a value file.

### Folder Structure

```
/mongodb-tls
|-- templates
|   |-- pvc.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

### Charts description

#### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for MongoDB implementation.
- This folder contains following template files for MongoDB implementation
  - deployment.yaml: This file is used as a basic manifest for creating a Kubernetes deployment.For the MongoDB node, this file creates MongoDB deployment.
  - service.yaml: This template is used as a basic manifest for creating a service endpoint for our deployment.This service.yaml creates MongoDB service endpoint
  - pvc.yaml: This yaml is used to create persistent volumes claim for the MongoDB deployment.A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates mongodb-pvc for , the volume claim for mongodb.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

### node

#### About

This folder consists node helm charts which are used by the ansible playbooks for the deployment of the corda node component. The folder contains a templates folder, a chart file and a value file.

#### Folder Structure

```
/node
|-- templates
|   |-- deployment.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

#### Charts description

##### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for R3 Corda node implementation.
- This folder contains following template files for node implementation
  - deployment.yaml: This file is used as a basic manifest for creating a Kubernetes deployment. For the corda node, this file creates a node deployment. The file defines containers where node container is defined with corda image and corda jar details and corda-logs container is used for logging purpose. The init container init-nodeconf defines node.conf file for node, init-certificates init container basically configures networkmap.crt, doorman.crt , SSLKEYSTORE and TRUSTSTORE at mount path for node by fetching certsecretprefix from the vault and init-healthcheck init-container checks for h2 database. Certificates and notary server database are defined on the volume mount paths.
  - service.yaml: This template is used as a basic manifest for creating a service endpoint for our deployment. This service.yaml creates node service endpoint. The file basically specifies service type and kind of service ports for the corda nodes.

##### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

## values.yaml

- This file contains the default configuration values for the chart.
- 

## node-initial-registration

### About

This folder contains node-initial-registration helm charts which are used by the ansible playbooks for the deployment of the install\_chaincode component. The folder contains a templates folder,a chart file and the corresponding value file.

### Folder Structure

```
/node-initial-registration
|-- templates
|   |-- job.yaml
|   |-- _helpers.tpl
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for node-initial-registration implementation.
- This folder contains following template files for node-initial-registration implementation
  - job.yaml: It is used as a basic manifest for creating a Kubernetes deployment for initial node registration. The file basically describes the container and volume specifications of the node. corda-node container is used for running corda jar.store-certs-in-vault container is used for putting certificate into the vault. init container is used for creating node.conf which is used by corda node, download corda jar, download certificate from vault,getting passwords of keystore from vault and also performs health checks
  - \_helpers.tpl: A place to put template helpers that you can re-use throughout the chart.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

## values.yaml

- This file contains the default configuration values for the chart.
-

## notary

### About

This folder consists Notary helm charts, which are used by the ansible playbooks for the deployment of the Notary component. The folder contains a templates folder,a chart file and a value file.

### Folder Structure

```
/notary
|-- templates
|   |-- deployment.tpl
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

### Charts description

#### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for Notary implementation.
- This folder contains following template files for Notary implementation
  - deployment.yaml: This file is used as a basic manifest for creating a Kubernetes deployment.For the corda notary, this file creates a notary deployment.The file defines conatainers where notary container is defined with corda image and corda jar details also registers the notary with nms and corda-logs container is used for logging purpose. The init container init-nodeconf defines node.conf file for notary, init-certificates init container basically configures networkmap.crt, doorman.crt , SSLKEYSTORE and TRUSTSTORE at mount path by fetching certsecretprefix from vault and db-healthcheck init-container checks for h2 database.Certificates and notary server database are defined on the volume mount paths.
  - service.yaml This template is used as a basic manifest for creating a service endpoint for our deployment. This service.yaml creates Notary service endpoint.The file basically specifies service type and kind of service ports for Notary.

#### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

#### values.yaml

- This file contains the default configuration values for the chart.

## notary-initial-registration

### About

This folder consists notary-initial-registration helm charts, which are used by the ansible playbooks for the deployment of the initial notary components. The folder contains a templates folder,a chart file and a corresponding value file.

### Folder Structure

```
/notary-initial-registration
|-- templates
|   |--job.yaml
|   |--_helpers.tpl
|-- Chart.yaml
|-- values.yaml
```

### Charts description

#### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for registering notary components.
- This folder contains following template files for initializing notary implementation.
  - job.yaml: It is used as a basic manifest for creating a Kubernetes deployment for initial notary registration. The file basically describes the container and volume specifications of the notary. corda-node container is used for running corda jar.store-certs-in-vault container is used for putting certificate into the vault. init container is used for creating node.conf which is used by corda node, download corda jar, download certificate from vault,getting passwords of keystore from vault and also performs health checks.
  - \_helpers.tpl: A place to put template helpers that you can re-use throughout the chart.

#### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

#### values.yaml

- This file contains the default configuration values for the chart.

## springbootwebserver

### About

This folder consists springbootwebserver helm charts which are used by the ansible playbooks for the deployment of the springbootwebserver component. The folder contains a templates folder,a chart file and a value file.

## Folder Structure

```
/springbootwebserver
|-- templates
|   |-- deployment.yaml
|   |-- pvc.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for springbootwebserver implementation.
- This folder contains following template files for springbootwebserver implementation
  - deployment.yaml: This file is used as a basic manifest for creating a Kubernetes deployment.For the corda springbootwebserver, this file creates a springbootwebserver deployment.The file defines containers where springbootwebserver container is defined with corda image and app jar details and the init container basically creates app.properties file, configures the vault with various vault parameters.Certificates and springbootwebserver database are defined on the volume mount paths.
  - pvc.yaml: This yaml is used to create persistent volumes claim for the springbootwebserver deployment.A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates springbootwebserver-pvc for , the volume claim for springbootwebserver.
  - service.yaml: This template is used as a basic manifest for creating a service endpoint for our deployment. This service.yaml creates springbootwebserver service endpoint.The file basically specifies service type and kind of service ports for the corda springbootwebserver.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## storage

### About

This folder consists storage helm charts, which are used by the ansible playbooks for the deployment of the storage component. The folder contains a templates folder,a chart file and a value file.

## Folder Structure

```
/storage
|-- templates
|   |-- storageclass.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for storageclass implementation.
- This folder contains following template files for storageclass implementation
  - storageclass.yaml: This yaml file basically creates storageclass.We define provisioner,storagename and namespace from value file.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## webserver Chart

### About

This folder consists webserver helm charts which are used by the ansible playbooks for the deployment of the web-server component. The folder contains a templates folder,a chart file and a value file.

## Folder Structure

```
/webserver
|-- templates
|   |-- pvc.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for webserver implementation.
- This folder contains following template files for webserver implementation
  - deployment.yaml: This file is used as a basic manifest for creating a Kubernetes deployment.For the webserver node, this file creates webserver deployment.
  - service.yaml: This template is used as a basic manifest for creating a service endpoint for our deployment.This service.yaml creates webserver service endpoint
  - pvc.yaml: This yaml is used to create persistent volumes claim for the webserver deployment.A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates webserver-pvc for , the volume claim for webserver.
  - volume.yaml: This yaml is used to create persistent volumes claim for the webserver deployment.A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates webserver pvc for, the volume claim for webserver.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.

## 6.4.3 Corda Enterprise Helm Charts

Following are the helm charts used for R3 Corda Enterprise in Blockchain Automation Framework.

```
platforms/r3-corda-ent/charts
├── bridge
├── float
├── generate-pki
├── h2
├── idman
├── nmap
├── node
├── node-initial-registration
├── notary
├── notary-initial-registration
└── signer
```

## Pre-requisites

helm version 2.x.x to be installed and configured on the cluster.

## Bridge

### About

This chart deploys the Bridge component of Corda Enterprise firewall. The folder contents are below:

### Folder Structure



## Charts description

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion, name etc.

### files

- This folder contains the configuration files needed for bridge.
  - firewall.conf: The main configuration file for firewall.

### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for Corda Firewall implementation. This folder contains following template files:
  - configmap.yaml: This creates a configmap of all the files from the `files` folder above.
  - deployment.yaml: This creates the main Kubernetes deployment. It contains one `init-container init-certificates` to download the keys/certs from Vault, and one `main containers` which executes the firewall service.
  - \_helpers.tpl: This is a helper file to add any custom labels.
  - pvc.yaml: This creates the PVC used by firwall
  - service.yaml: This creates the firewall service endpoint.

## values.yaml

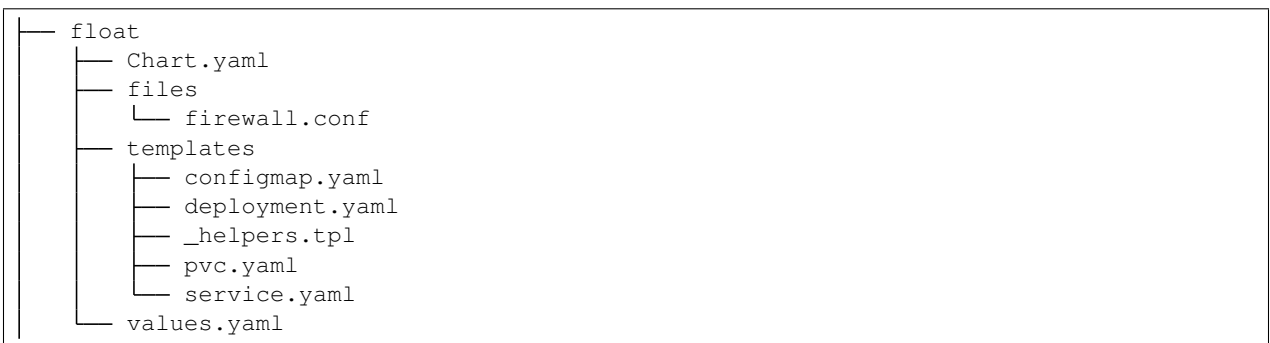
- This file contains the default values for the chart.
- 

## Float

### About

This chart deploys the Float component of Corda Enterprise firewall. The folder contents are below:

### Folder Structure



### Charts description

#### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion, name etc.

#### files

- This folder contains the configuration files needed for float.
  - firewall.conf: The main configuration file for firewall.

#### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for Corda Firewall implementation. This folder contains following template files:
  - configmap.yaml: This creates a configmap of all the files from the `files` folder above.
  - deployment.yaml: This creates the main Kubernetes deployment. It contains one `init-container init-certificates` to download the keys/certs from Vault, and one `main containers` which executes the firewall service.
  - \_helpers.tpl: This is a helper file to add any custom labels.

- pvc.yaml: This creates the PVC used by firewall
- service.yaml: This creates the firewall service endpoint.

### values.yaml

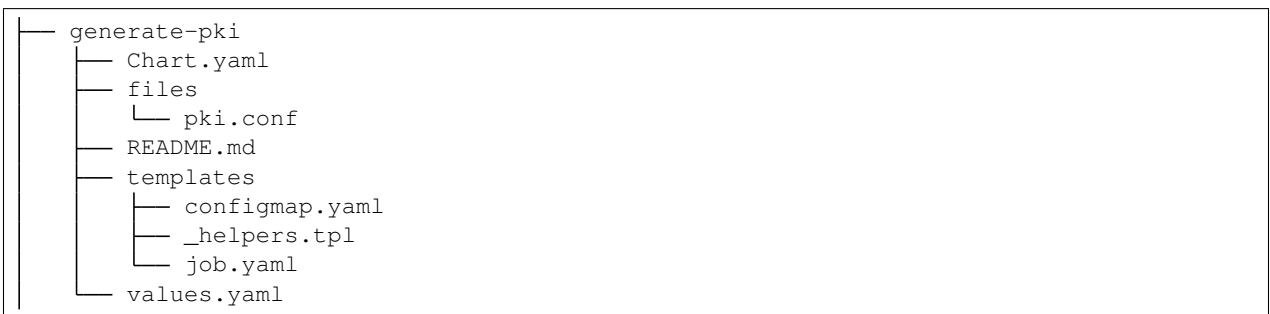
- This file contains the default values for the chart.
- 

## Generate-pki

### About

This chart deploys the Generate-PKI job on Kubernetes. The folder contents are below:

### Folder Structure



## Charts description

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion, name etc.

### files

- This folder contains the configuration files needed for PKI.
  - pki.conf: The main configuration file for generate-pki.

### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for PKI job. This folder contains following template files:
  - configmap.yaml: This creates a configmap of all the files from the `files` folder above.
  - \_helpers.tpl: This is a helper file to add any custom labels.

- `job.yaml`: This creates the main Kubernetes job. It contains a `main` container which runs the `pkitool` to generate the certificates and keystores, and a `store-certs` container to upload the certificates/keystores to Vault.

### values.yaml

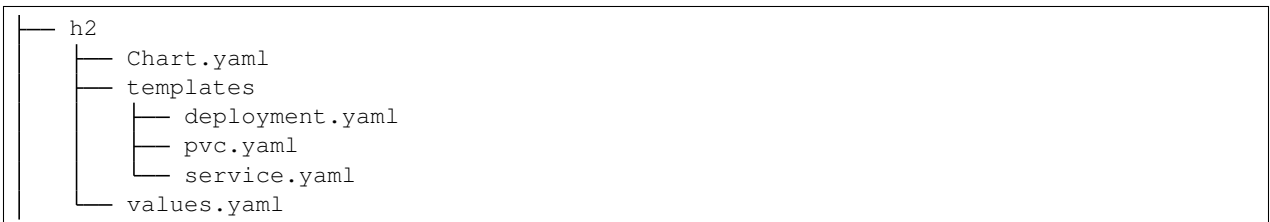
- This file contains the default values for the chart.
- 

## h2 (database)

### About

This chart deploys the H2 database pod on Kubernetes. The folder contents are below:

### Folder Structure



### Charts description

#### Chart.yaml

- This file contains the information about the chart such as `apiversion`, `appversion`, `name` etc

#### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for H2 implementation. This folder contains following template files:
  - `deployment.yaml`: This file is used as a basic manifest for creating a Kubernetes deployment. For the H2 node, this file creates H2 pod.
  - `pvc.yaml`: This yaml is used to create persistent volumes claim for the H2 deployment. This file creates `h2-pvc` for, the volume claim for H2.
  - `service.yaml`: This template is used as a basic manifest for creating a service endpoint for our deployment. This `service.yaml` creates H2 service endpoint.

### values.yaml

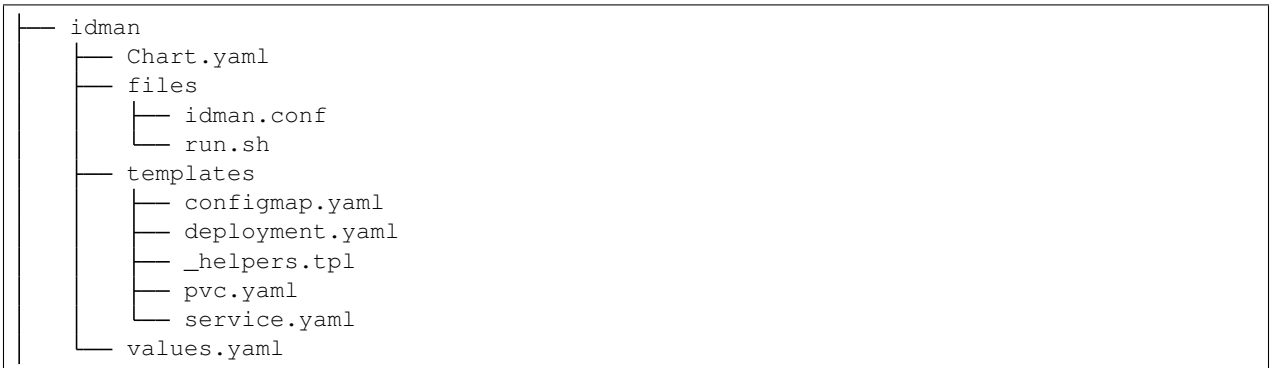
- This file contains the default configuration values for the chart.
-

## idman

### About

This chart deploys the Idman component of Corda CENM. The folder contents are below:

### Folder Structure



### Charts description

#### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion, name etc.

#### files

- This folder contains the configuration files needed for idman.
  - idman.conf: The main configuration file for idman.
  - run.sh: The executable file to run the idman service in the kubernetes pod.

#### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for Idman implementation. This folder contains following template files:
  - configmap.yaml: This creates a configmap of all the files from the `files` folder above.
  - deployment.yaml: This creates the main Kubernetes deployment. It contains one init-container `init-certificates` to download the keys/certs from Vault, and two main containers: `idman` and `logs`.
  - \_helpers.tpl: This is a helper file to add any custom labels.
  - pvc.yaml: This creates the PVCs used by idman: one for logs and one for the file H2 database.
  - service.yaml: This creates the idman service endpoint with Ambassador proxy configurations.

## values.yaml

- This file contains the default values for the chart.
- 

## nmap

### About

This chart deploys the NetworkMap component of Corda CENM. The folder contents are below:

### Folder Structure



### Charts description

#### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion, name etc.

#### files

- This folder contains the configuration files needed for nmap.
  - nmap.conf: The main configuration file for nmap.
  - run.sh: The executable file to run the nmap service in the kubernetes pod.
  - set-network-parameters.sh: This executable file which creates the initial network-parameters.

#### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for NetworkMap implementation. This folder contains following template files:
  - configmap.yaml: This creates a configmap of all the files from the `files` folder above.

- `deployment.yaml`: This creates the main Kubernetes deployment. It contains a `init-container init-certificates` to download the keys/certs from Vault, a `setnparam` container to set the `network-parameters`, and two main containers: `main` and `logs`.
- `_helpers.tpl`: This is a helper file to add any custom labels.
- `pvc.yaml`: This creates the PVCs used by `nmap`: one for logs and one for the file H2 database.
- `service.yaml`: This creates the `nmap` service endpoint with Ambassador proxy configurations.

### values.yaml

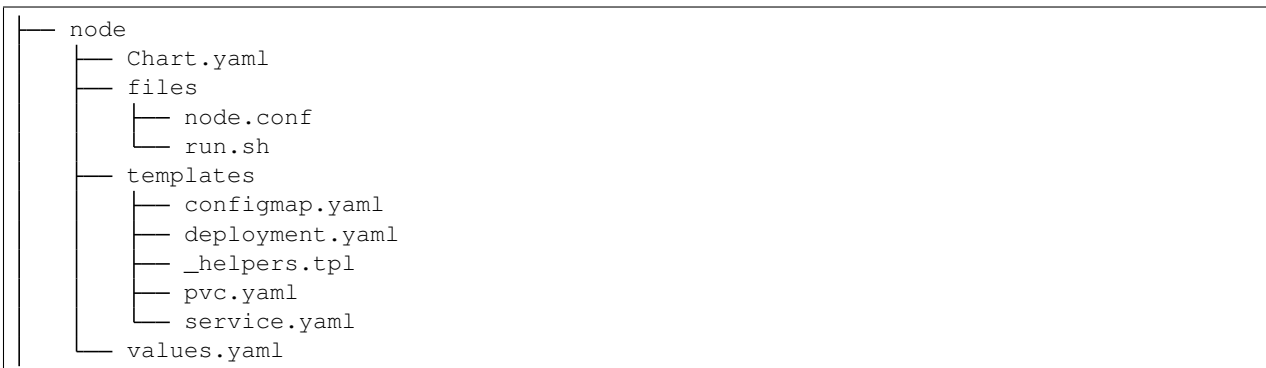
- This file contains the default values for the chart.
- 

## node

### About

This chart deploys the Node component of Corda Enterprise. The folder contents are below:

### Folder Structure



### Charts description

#### Chart.yaml

- This file contains the information about the chart such as `apiversion`, `appversion`, `name` etc.

#### files

- This folder contains the configuration files needed for Corda node.
  - `node.conf`: The main configuration file for node.
  - `run.sh`: The executable file to run the node service in the kubernetes pod.

## templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for Corda Node implementation. This folder contains following template files:
  - `configmap.yaml`: This creates a configmap of all the files from the `files` folder above.
  - `deployment.yaml`: This creates the main Kubernetes deployment. It contains three init-containers: `init-check-registration` to check if node-initial-registration was completed, `init-certificates` to download the keys/certs from Vault, and a `db-healthcheck` container to check if the database service is reachable, and two main containers: `node` and `logs`.
  - `_helpers.tpl`: This is a helper file to add any custom labels.
  - `pvc.yaml`: This creates the PVC used by the node.
  - `service.yaml`: This creates the node service endpoint with Ambassador proxy configurations.

## values.yaml

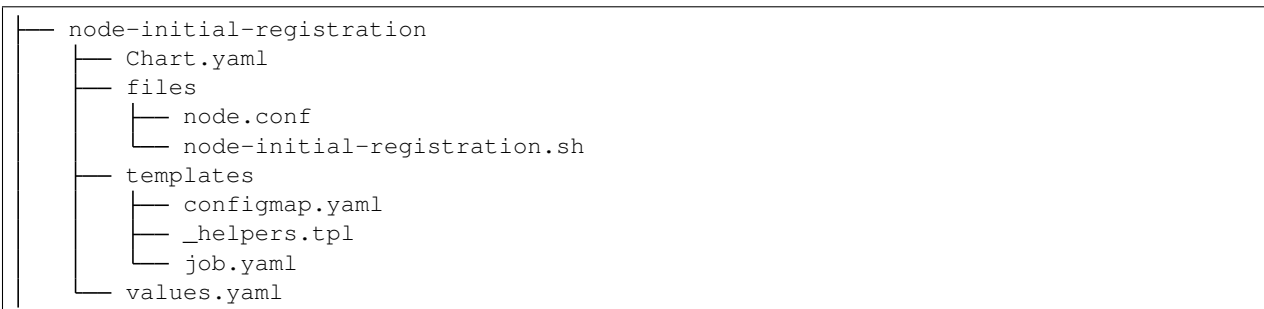
- This file contains the default values for the chart.
- 

## node-initial-registration

### About

This chart deploys the Node-Registration job for Corda Enterprise. The folder contents are below:

### Folder Structure



## Charts description

### Chart.yaml

- This file contains the information about the chart such as `apiversion`, `appversion`, `name` etc.

## files

- This folder contains the configuration files needed for Corda node.
  - node.conf: The main configuration file for node.
  - node-initial-registration.sh: The executable file to run the node initial-registration.

## templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for registration job. This folder contains following template files:
  - configmap.yaml: This creates a configmap of all the files from the `files` folder above.
  - \_helpers.tpl: This is a helper file to add any custom labels.
  - job.yaml: This creates the main Kubernetes job. It contains two init-containers: `init-certificates` to download the keys/certs from Vault, and a `db-healthcheck` container to check if the database service is reachable, and two main containers: `registration` for the actual registration and `store-certs` to upload the certificates to Vault.

## values.yaml

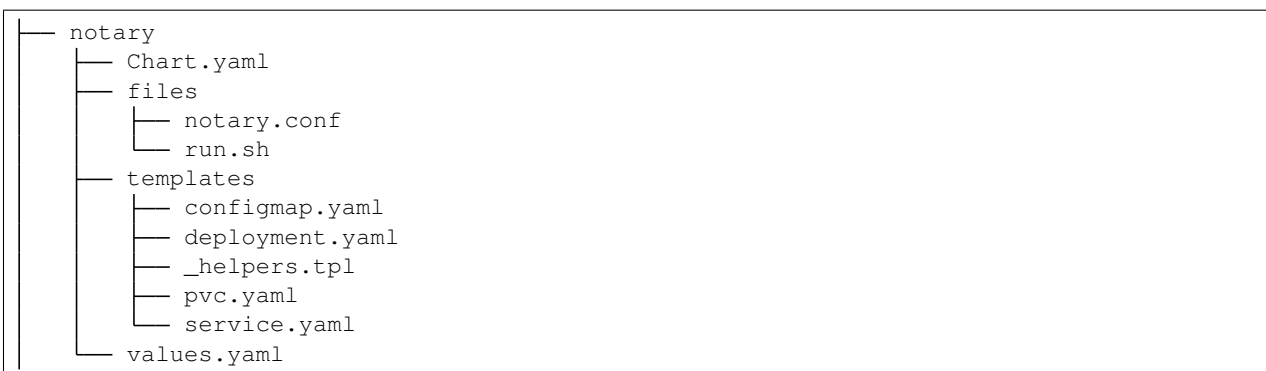
- This file contains the default values for the chart.
- 

## notary

### About

This chart deploys the Notary component of Corda Enterprise. The folder contents are below:

### Folder Structure



## Charts description

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion, name etc.

### files

- This folder contains the configuration files needed for Corda Notary.
  - notary.conf: The main configuration file for notary.
  - run.sh: The executable file to run the notary service in the kubernetes pod.

### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for Corda Notary implementation. This folder contains following template files:
  - configmap.yaml: This creates a configmap of all the files from the `files` folder above.
  - deployment.yaml: This creates the main Kubernetes deployment. It contains three init-containers: `init-check-registration` to check if notary-initial-registration was completed, `init-certificates` to download the keys/certs from Vault, and a `db-healthcheck` container to check if the database service is reachable, and two main containers: `notary` and `logs`.
  - \_helpers.tpl: This is a helper file to add any custom labels.
  - pvc.yaml: This creates the PVC used by the notary.
  - service.yaml: This creates the notary service endpoint with Ambassador proxy configurations.

### values.yaml

- This file contains the default values for the chart.

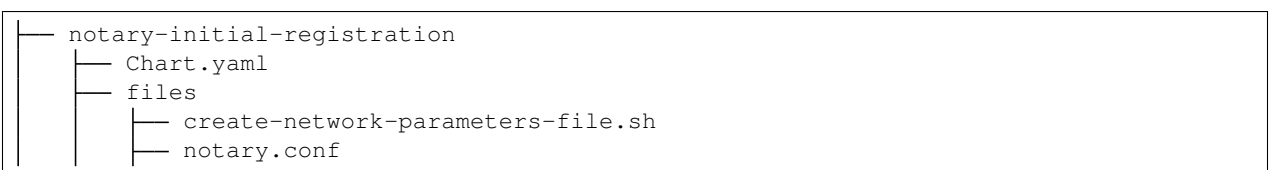
---

## notary-initial-registration

### About

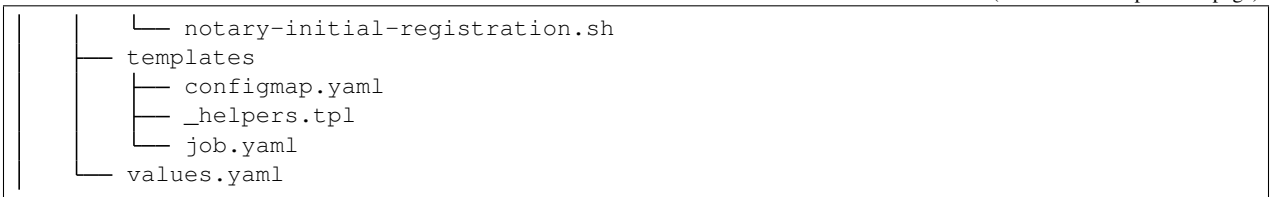
This chart deploys the Notary-Registration job for Corda Enterprise. The folder contents are below:

### Folder Structure



(continues on next page)

(continued from previous page)



## Charts description

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion, name etc.

### files

- This folder contains the configuration files needed for Corda Notary.
  - create-network-parameters-file.sh: Creates the network parameters file.
  - notary.conf: The main configuration file for notary.
  - notary-initial-registration.sh: The executable file to run the notary initial-registration.

### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for Notary registration job. This folder contains following template files:
  - configmap.yaml: This creates a configmap of all the files from the `files` folder above.
  - \_helpers.tpl: This is a helper file to add any custom labels.
  - job.yaml: This creates the main Kubernetes job. It contains two init-containers: `init-certificates` to download the keys/certs from Vault, and a `db-healthcheck` container to check if the database service is reachable, and two main containers: `registration` for the actual registration and `store-certs` to upload the certificates to Vault.

### values.yaml

- This file contains the default values for the chart.

## signer

### About

This chart deploys the Signer component of Corda CENM. The folder contents are below:

## Folder Structure



## Charts description

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion, name etc.

### files

- This folder contains the configuration files needed for signer.
  - signer.conf: The main configuration file for signer.

### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for Signer implementation. This folder contains following template files:
  - configmap.yaml: This creates a configmap of all the files from the `files` folder above.
  - deployment.yaml: This creates the main Kubernetes deployment. It contains two init-containers: `init-check-certificates` to check if the signer certificates are saved on Vault and `init-certificates` to download the keys/certs from Vault, and two main containers: `signer` and `logs`.
  - \_helpers.tpl: This is a helper file to add any custom labels.
  - service.yaml: This creates the signer service endpoint.

### values.yaml

- This file contains the default values for the chart.

## 6.4.4 Hyperledger Fabric Charts

The structure below represents the Chart structure for Hyperledger fabric components in the Blockchain Automation Framework implementation.

```
/hyperledger-fabric
|-- charts
|   |-- ca
|   |-- catools
|   |-- create_channel
|   |-- fabric_cli
|   |-- install_chaincode
|   |-- instantiate_chaincode
|   |-- join_channel
|   |-- orderernode
|   |-- peernode
|   |-- upgrade_chaincode
|   |-- verify_chaincode
|   |-- zkkafka
```

---

## Pre-requisites

helm to be installed and configured on the cluster.

## CA (certification authority)

### About

This folder consists CA helm charts which are used by the ansible playbooks for the deployment of the CA component. The folder contains a templates folder, a chart file and a value file.

### Folder Structure

```
/ca
|-- templates
|   |-- _helpers.tpl
|   |-- volumes.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for CA implementation.
- This folder contains following template files for CA implementation
  - *helpers.tpl* This file doesnt output a Kubernetes manifest file as it begins with underscore (). And its a place to put template helpers that we can re-use throughout the chart. That file is the default location for template partials ,as we have defined a template to encapsulate a Kubernetes block of labels for CA.

- deployment.yaml This file is used as a basic manifest for creating a Kubernetes deployment. For the ca node, this file creates a ca deployment. The file defines where ca container is defined with fabric image and CA client and CA server onfiguration details and the init container basically configures the vault with various vault parameters. Certificates and CA server database are defined on the volume mount paths.
- service.yaml This template is used as a basic manifest for creating a service endpoint for our deployment. This service.yaml creates ca service endpoint. The file basically specifies service type and kind of service ports for the CA client and CA server.
- volume.yaml This yaml is used to create persistent volumes claim for the CA deployment. A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates ca pvc for, the volume claim for CA.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## CA tools

### About

This folder consists CA tools helm charts which are used by the ansible playbooks for the deployment of the CA tools component. The folder contains a templates folder, a chart file and a value file.

### Folder Structure

```
/catools
|-- templates
|   |-- volumes.yaml
|   |-- deployment.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for CA tools implementation.
- This folder contains following template files for CA tools implementation
  - deployment.yaml This file is used as a basic manifest for creating a Kubernetes deployment for CA tools . The file basiclly describes the container and volume specifications of the ca tools

- `volume.yaml` This yaml is used to create persistent volumes claim for the Orderer deployment. A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates two persistentVolumeClaims , one for CA tools pvc and the other to store crypto config in the ca-tools-crypto-pvc persistent volume.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## Create channel

### About

This folder consists create\_channel helm charts which are used by the ansible playbooks for the deployment of the create\_channel component. The folder contains a templates folder, a chart file and a value file.

### Folder Structure

```
/create_channel
|-- templates
|   |--_helpers.tpl
|   |-- create_channel.yaml
|   |-- configmap.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for Peer implementation.
- This folder contains following template files for peer implementation
  - `helpers.tpl` This file doesn't output a Kubernetes manifest file as it begins with underscore ( ) .And it's a place to put template helpers that we can re-use throughout the chart. That file is the default location for template partials, as we have defined a template to encapsulate a Kubernetes block of labels for channels.
  - `configmap.yaml` The configmap.yaml file through template engine generate configmaps. In Kubernetes, a ConfigMap is a container for storing configuration data. Things like pods, can access the data in a ConfigMap. The configmap.yaml file creates two configmaps namely genesis-block-peer and peer-config .

When Tiller reads this configmap.yaml template, it sends it to Kubernetes as-is. For Create\_channel component, it creates two configmaps, one for the channel creation having various data fields such as channel, peer and orderer details and other for the generation of channel artifacts containing the channel transaction (channeltx) block and other labels.

- create\_channel.yaml This file creates channel creation job where in the createchannel container the create channel peer commands are fired based on checking the results obtained from fetching channeltx block to see if channel has already been created or not. Additionally, the commands are fired based on the status whether it is enabled or not. The init container is used to setup vault configurations. And certificates are obtained from the volume mount paths.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion, name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## Install Chaincode

### About

This folder consists install\_chaincode helm charts which are used by the ansible playbooks for the deployment of the install\_chaincode component. The folder contains a templates folder, a chart file and a value file.

### Folder Structure

```
/install_chaincode
|-- templates
|   |-- helpers.tpl
|   |-- install_chaincode.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for install\_chaincode implementation.
- This folder contains following template files for install\_chaincode implementation
  - helpers.tpl This file doesn't output a Kubernetes manifest file as it begins with underscore (). And it's a place to put template helpers that we can re-use throughout the chart. This file is the default location for template partials, as we have defined a template to encapsulate a Kubernetes block of labels for install\_chaincodes.

- `install_chaincode.yaml` This yaml file basically creates a job for the installation of chaincode. We define containers where fabric tools image is pulled and chaincode install peer commands are fired. Moreover, the chart provides the environment requirements such as docker endpoint, peer and orderer related information, volume mounts etc for the chaincode to be installed. The init container basically configures the vault with various vault parameters.

## Chart.yaml

- This file contains the information about the chart such as appversion, appversion, name etc.

## values.yaml

- This file contains the default configuration values for the chart.
- 

## Instantiate Chaincode

### About

This folder consists of `instantiate_chaincode` helm charts, which are used by the ansible playbooks for the deployment of the `instantiate_chaincode` component. The folder contains a `templates` folder, a chart file and a value file.

### Folder Structure

```
/instantiate_chaincode
|-- templates
|   |-- _helpers.tpl
|   |-- instantiate_chaincode.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for `instantiate_chaincode` implementation.
- This folder contains following template files for `instantiate_chaincode` implementation
  - `helpers.tpl` This file doesn't output a Kubernetes manifest file as it begins with underscore (`_`). And it's a place to put template helpers that we can re-use throughout the chart. This file is the default location for template partials, as we have defined a template to encapsulate a Kubernetes block of labels for `instantiate_chaincode`s.
  - `instantiate_chaincode.yaml` This yaml file basically creates a job for the instantiation of chaincode. We define containers where fabric tools image is pulled and based on the endorsement policies set, chaincode `instantiate` peer commands are fired. Moreover, the chart provides the environment requirements such as docker endpoint, peer and orderer related information, volume mounts etc for the chaincode to be instantiated. The init container basically configures the vault with various vault parameters.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

### Join channel

#### About

This folder consists join\_channel helm charts which are used by the ansible playbooks for the deployment of the join\_channel component. The foldere contains a templates folder,a chart file and a value file.

#### Folder Structure

```
/join_channel
|-- templates
|   |--_helpers.tpl
|   |-- join_channel.yaml
|   |-- configmap.yaml
|-- Chart.yaml
|-- values.yaml
```

### Charts description

#### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for Peer implementation.
- This folder contains following template files for peer implementation
  - *helpers.tpl* This fie doesnt output a Kubernetes manifest file as it begins with underscore ( ) .And its a place to put template helpers that we can re-use throughout the chart. That file is the default location for template partials ,as we have defined a template to encapsulate a Kubernetes block of labels for peers.
  - *configmap.yaml* The configmap.yaml file through template engine generate configmaps.In Kubernetes, a ConfigMap is a container for storing configuration data.Things like pods, can access the data in a ConfigMap. The configmap.yaml file creates two configmaps namely genesis-block-peer and peer-config . When Tiller reads this configmap.yaml template, it sends it to Kubernetes as-is. For join\_channel component , it creates two configmaps, one for the channel creation having various data fields such as channel , peer and orderer details and other for the generation of channel artifacts containing the channel transaction (channeltx) block and other labels.
  - *join\_channel.yaml* This file creates channel join job where in the joinchannel container the commands are fired based on the tls status whether it is enabled or not wherein first the channel config is fetched and then the peers join the created channel. The init container is used to setup vault configurations.And certificates are obatined from the volume mount paths.

## Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

## values.yaml

- This file contains the default configuration values for the chart.
- 

## Orderer Chart

### About

This folder consists Orderer helm charts which are used by the ansible playbooks for the deployment of the Orderer component. The folder contains a templates folder,a chart file and a value file.

### Folder Structure

```
/Orderernode
|-- templates
|   |--_helpers.tpl
|   |-- volumes.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|   |-- configmap.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for Orderer implementation.
- This folder contains following template files for Orderer implementation
  - *helpers.tpl* This file doesnt output a Kubernetes manifest file as it begins with underscore ( ) .And its a place to put template helpers that we can re-use throughout the chart. That file is the default location for template partials ,as we have defined a template to encapsulate a Kubernetes block of labels for Orderers.
  - *configmap.yaml* The configmap.yaml file through template engine generate configmaps.In Kubernetes, a ConfigMap is a container for storing configuration data.Things like pods, can access the data in a ConfigMap. The configmap.yaml file creates two configmaps namely genesis-block-orderer and orderer-config . When Tiller reads this configmap.yaml template, it sends it to Kubernetes as-is.
  - *deployment.yaml* This file is used as a basic manifest for creating a Kubernetes deployment.For the Orderer node, this file creates orderer deployment.
  - *service.yaml* This template is used as a basic manifest for creating a service endpoint for our deployment.This service.yaml creates orderer service endpoint

- `volume.yaml` This yaml is used to create persistent volumes claim for the Orderer deployment. A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates orderer-pvc for , the volume claim for Orderer.

### Chart.yaml

- This file contains the information about the chart such as appversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## Peer Chart

### About

This folder consists Peer helm charts which are used by the ansible playbooks for the deployment of the Peer component. The folder contains a templates folder, a chart file and a value file.

### Folder Structure

```
/peernode
|-- templates
|   |-- _helpers.tpl
|   |-- volumes.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|   |-- configmap.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for Peer implementation.
- This folder contains following template files for peer implementation
  - `helpers.tpl` This file does not output a Kubernetes manifest file as it begins with underscore ( `_` ). And it's a place to put template helpers that we can re-use throughout the chart. That file is the default location for template partials, as we have defined a template to encapsulate a Kubernetes block of labels for peers.
  - `configmap.yaml` The configmap.yaml file through template engine generate configmaps. In Kubernetes, a ConfigMap is a container for storing configuration data. Things like pods, can access the data in a ConfigMap. The configmap.yaml file creates two configmaps namely `genesis-block-peer` and `peer-config` . When Tiller reads this configmap.yaml template, it sends it to Kubernetes as-is.

- `service.yaml` This template is used as a basic manifest for creating a service endpoint for our deployment. This `service.yaml` creates peer service endpoint.
- `volume.yaml` This `yaml` is used to create persistent volumes claim for the peer deployment. A `persistentVolumeClaim` volume is used to mount a `PersistentVolume` into a `Pod`. `PersistentVolumes` provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates `peer-pvc` for , the volume claim for peer.
- `deployment.yaml` This file is used as a basic manifest for creating a Kubernetes deployment. For the peer node, this file creates three deployments namely `ca`, `ca-tools` and `peer`

### Chart.yaml

- This file contains the information about the chart such as `apiVersion`, `appVersion` ,`name` etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## Upgrade Chaincode

### About

This folder consists `upgrade_chaincode` helm charts, which are used by the ansible playbooks for the deployment of the `upgrade_chaincode` component. The folder contains a `templates` folder, a chart file and a value file.

### Folder Structure

```
/upgrade_chaincode
|-- templates
|   |--_helpers.tpl
|   |-- upgrade_chaincode.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for `upgrade_chaincode` implementation.
- This folder contains following template files for `upgrade_chaincode` implementation
  - `helpers.tpl` This file doesn't output a Kubernetes manifest file as it begins with underscore (`_`) .And its a place to put template helpers that we can re-use throughout the chart. This file is the default location for template partials ,as we have defined a template to encapsulate a Kubernetes block of labels for `upgrade_chaincodes`.

- `upgrade_chaincode.yaml` This yaml file basically creates a job for the upgradation of chaincode. We define containers where fabric tools image is pulled and based on the endorsement policies set, chaincode upgrade peer commands are fired. Moreover, the chart provides the environment requirements such as docker endpoint, peer and orderer related information, volume mounts, channel information etc for the chaincode to be upgraded. The init container basically configures the vault with various vault parameters.

### Chart.yaml

- This file contains the information about the chart such as app version, app version, name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## zkkafka

### About

This folder consists of zkkafka helm charts which are used by the ansible playbooks for the deployment of the zkkafka component. The folder contains a templates folder, a chart file and a value file.

### Folder Structure

```
/zkkafka
|-- templates
|   |-- _helpers.tpl
|   |-- volumes.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

### Charts description

#### templates

- This folder contains template structures which when combined with values, will generate valid Kubernetes manifest files for zkkafka implementation.
- This folder contains following template files for zkkafka implementation
  - `helpers.tpl` This file doesn't output a Kubernetes manifest file as it begins with underscore (`_`). And it's a place to put template helpers that we can re-use throughout the chart. That file is the default location for template partials, as we have defined a template to encapsulate a Kubernetes block of labels for zkkafka.
  - `deployment.yaml` This file is used as a basic manifest for creating a Kubernetes deployment. For the zkkafka node, this file creates zkkafka deployment.

- service.yaml This template is used as a basic manifest for creating a service endpoint for our deployment. This service.yaml creates zkkafka service endpoint
- volume.yaml This yaml is used to create persistent volumes claim for the zkkafka deployment. A persistentVolumeClaim volume is used to mount a PersistentVolume into a Pod. PersistentVolumes provide a way for users to ‘claim’ durable storage without having the information details of the particular cloud environment. This file creates zkkafka pvc for , the volume claim for zkkafka.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.

## 6.4.5 Indy Charts

The structure below represents the Chart structure for Hyperledger Indy components in the Blockchain Automation Framework implementation.

```
/hyperledger-indy
|-- charts
|   |-- indy-auth-job
|   |-- indy-cli
|   |-- indy-domain-genesis
|   |-- indy-key-mgmt
|   |-- indy-ledger-txn
|   |-- indy-node
|   |-- indy-pool-genesis
```

---

### Pre-requisites

helm to be installed and configured on the cluster.

### Indy-Auth-Job

#### About

This chart is using admin auth to generate auth. The folder contains a templates folder, a chart file and a value file.

#### Folder Structure

```
/indy-auth-job
|-- templates
|   |-- job.yaml
|-- Chart.yaml
|-- values.yaml
```

### Charts description

#### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for auth job implementation.
- This folder contains following template files for auth job implementation
  - Job.yaml This job uses admin auth to generate auth read only methods, policies and roles for stewards, so they have the right they need to work.

#### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

#### values.yaml

- This file contains the default configuration values for the chart.
- 

## Indy-Domain-Genesis

### About

This folder consists of domain genesis helm chart which is used to generate the domain genesis for indy network.

### Folder Structure

```
/indy-domain-genesis
|-- templates
|   |-- configmap.yaml
|-- Chart.yaml
|-- values.yaml
```

### Charts description

#### templates

- This chart is used to generate the domain genesis.
  - configmap.yaml The ConfigMap API resource provides mechanisms to inject containers with configuration data while keeping containers agnostic of Kubernetes.Here it is used to store Domain Genesis Data.

#### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

## **values.yaml**

- This file contains the default configuration values for the chart.
- 

## **Indy Key Management**

### **About**

This folder consists indy-key-management helm charts which are used by the ansible playbooks for the generation of indy crypto material. The folder contains a templates folder,a chart file and a value file.

### **Folder Structure**

```
/indy-key-management
|-- templates
|   |-- job.yaml
|-- Chart.yaml
|-- values.yaml
```

## **Charts description**

### **templates**

- This folder contains template structures which when combined with values ,will generate crypto material for Indy.
- This folder contains following template files for peer implementation
  - job.yaml This job is used to generate crypto and save into vault.

### **Chart.yaml**

- This file contains the information about the chart such as apiversion, appversion ,name etc.

## **values.yaml**

- This file contains the default configuration values for the chart.
- 

## **Indy Ledger Txn**

### **About**

This folder contains helm chart which is used to run Indy Ledger Transaction Script.

### Folder Structure

```
/indy-ledger-txn
|-- templates
|   |-- job.yaml
|-- Chart.yaml
|-- values.yaml
```

### Charts description

#### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for ledger NYM transaction implementation.
- This folder contains following template files for indy-ledger NYM Transaction implementation
  - job.yaml This Job is used to generate a NYM transaction between an admin identity and an endorser identity.

#### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

#### values.yaml

- This file contains the default configuration values for the chart.
- 

### Indy Node

#### About

This folder consists indy-node helm charts, which are used by the ansible playbooks for the deployment of the indy nodes. The folder contains a templates folder,a chart file and a value file.

### Folder Structure

```
/instantiate_chaincode
|-- templates
|   |-- configmap.yaml
|   |-- service.yaml
|   |-- statesfulset.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate Indy nodes.
- This folder contains following template files for instantiate\_chaincode implementation
  - configmap.yaml The configmap.yaml file through template engine generate configmaps.In Kubernetes, a ConfigMap is a container for storing configuration data. Things like pods, can access the data in a ConfigMap.This file is used to inject Kubernetes container with indy config data.
  - service.yaml This creates a service for indy node and indy node client.A service in Kubernetes is a grouping of pods that are running on the cluster
  - statefulset.yaml Statefulsets is used for Stateful applications, each replica of the pod will have its own state, and will be using its own Volume. This statefulset is used to create indy nodes.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## Indy Pool Genesis

### About

This folder consists of pool genesis helm chart which is used to generate the pool genesis for indy network.

### Folder Structure

```
/indy-pool-genesis
|-- templates
|   |-- configmap.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This chart is used to generate the initial pool genesis which is used to connect to indy network.
  - configmap.yaml The ConfigMap API resource provides mechanisms to inject containers with configuration data while keeping containers agnostic of Kubernetes.Here it is used to store Pool Genesis Data.

## Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

## values.yaml

- This file contains the default configuration values for the chart.

## 6.4.6 Quorum Charts

The structure below represents the Chart structure for Quorum components in the Blockchain Automation Framework implementation.

```
/quorum
|-- charts
|   |-- node_constellation
|   |-- node_tessera
```

---

## Pre-requisites

helm to be installed and configured on the cluster.

## node\_constellation

### About

This chart is used to deploy Quorum nodes with constellation transaction manager.

## Folder Structure

```
/node_constellation
|-- templates
|   |-- _helpers.tpl
|   |-- configmap.yaml
|   |-- ingress.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for auth job implementation.

- This folder contains following template files for node\_constellation implementation
  - *helpers.tpl* This file doesnt output a Kubernetes manifest file as it begins with underscore ( ) .And its a place to put template helpers that we can re-use throughout the chart. That file is the default location for template partials ,as we have defined a template to encapsulate a Kubernetes block of labels for node\_constellation.
  - *deployment.yaml* This file is used as a basic manifest for creating a Kubernetes deployment.For the node\_constellation, this file creates a constellation node deployment deployment.The file defines 3 container, init container which gets all the secrets from the vault, constellation node container and a quorum container.
  - *service.yaml* This template is used as a basic manifest for creating a service endpoint for our deployment.The file basically specifies service type and kind of service ports for the constellation node.
  - *configmap.yaml* The ConfigMap API resource provides mechanisms to inject containers with configuration data while keeping containers agnostic of Kubernetes.Here it is used to store Genesis Data.
  - *ingress.yaml* Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource. This file contains those resources.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.
- 

## node\_tessera

### About

This chart is used to deploy Quorum nodes with tessera transaction manager.

### Folder Structure

```
/node_constellation
|-- templates
|   |-- _helpers.tpl
|   |-- configmap.yaml
|   |-- ingress.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

## Charts description

### templates

- This folder contains template structures which when combined with values ,will generate valid Kubernetes manifest files for tessera implementation.
- This folder contains following template files for node\_constellation implementation
  - *helpers.tpl* This file doesnt output a Kubernetes manifest file as it begins with underscore ( ) .And its a place to put template helpers that we can re-use throughout the chart. That file is the default location for template partials ,as we have defined a template to encapsulate a Kubernetes block of labels for node\_constellation.
  - *deployment.yaml* This file is used as a basic manifest for creating a Kubernetes deployment.For the node\_constellation, this file creates a constellation node deployment deployment.The file defines 4 containers, init container which gets all the secrets from the vault, mysql-init caontainer , mysql-db and a quorum container.
  - *service.yaml* This template is used as a basic manifest for creating a service endpoint for our deployment.The file basically specifies service type and kind of service ports for the tessera node.
  - *configmap.yaml* The ConfigMap API resource provides mechanisms to inject containers with configuration data while keeping containers agnostic of Kubernetes.Here it is used to store tessera config data.
  - *ingress.yaml* Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource. This file contains those resources.

### Chart.yaml

- This file contains the information about the chart such as apiversion, appversion ,name etc.

### values.yaml

- This file contains the default configuration values for the chart.

## 6.4.7 Hyperledger Besu Charts

The structure below represents the Chart structure for Hyperledger Besu components in the Blockchain Automation Framework implementation.

```
|hyperledger-besu
|-- charts
|   |-- node_orion
```

### Pre-requisites

helm to be installed and configured on the cluster.

## node\_orion (besu node chart with orion transaction manager)

### About

This folder consists of Hyperledger-Besu node charts which is used by the ansible playbook for the deployment of the node. This folder contains a template folder, a chart file and a value file.

### Folder Structure

```
|node_orion
|-- templates
|   |-- _helpers.tpl
|   |-- configmap.yaml
|   |-- deployment.yaml
|   |-- service.yaml
|-- Chart.yaml
|-- values.yaml
```

### Charts description

#### templates

- This folder contains template structures which when combined with values, will generate valid Kuberetenes manifest files for Hyperledger-Besu node implementation.
- This folder contains following template files for node implementation
  - *helpers.tpl* This file doesn't output a *Kubernets manifest file as it begins with underscore ()*. And its a place to put template helpers that we can re-use throught the chart. That file is the default location for template partials, as we have defined a template to encapsulate a Kuberntes block label for node.
  - *configmap.yaml* The configmap contains the genesis file data encoded in base64 format.
  - *deployment.yaml* This file is used a basic manifest for creating a Kubernetes deployment. For the node, this file creates a deployment. The file defines where containers are defined and the respective Hyperledger-Besu images. It also contain the initial containers where the crypto material is fetched from the vault.
  - *service.yaml* This tempalate is used ts a basic manifest for creating service endpoints for our deployment. This service.yaml creates endpoints for the besu node.

## 6.5 Jenkins Automation

### 6.5.1 Jenkins Pipeline

Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

#### Jenkins in Blockchain Automation Framework

In Blockchain Automation Framework, although Jenkins is not mandatory, we have a [single Jenkinsfile](#) as a sample to help you setup CI/CD Pipelines.

### Pre-requisites

1. Setup Jenkins with slave configurations. Declare a slave-config called `ansible` with the Docker Image `hyperledgerlabs/baf-build:jenkins`.
2. A EKS Cluster (Managed on AWS) and its kubeconfig file available and accessible from the Jenkins server.
3. AWS user `jenkins` with CLI credentials with access to above EKS Cluster.
4. A Hashicorp Vault installation which is accessible from the Jenkins server.
5. A Git repo which will be added as multi-branch pipeline on Jenkins (this is a fork of this repo).
6. A separate `baf-configuration` git repo where the templated `network.yaml` for different platforms are stored. Details of this repo needs to be updated in pipeline Stage `Create Configuration File`.

### Branch Configuration

The Jenkinsfile is designed to ignore `develop` and `master` branches by default. So, create platform specific branches in your forked repo.

- `corda` for Opensource Corda
- `corda-ent` for Enterprise Corda
- `fabric` for Hyperledger Fabric
- `besu` for Hyperledger Besu
- `indy` for Hyperledger Indy
- `quorum` for Quorum

Your `baf-configuration` repo should have the corresponding folders and files as demanded/configured in Stage `Create Configuration File`.

### Jenkins Secrets

Following secrets must be stored in Jenkins which is configured in the environment section. This can be re-named/updated in the Jenkinsfile according to your needs.

- `sownak-innersource`: is the Git Token and password to access the Git repos.
- `aws_demo_kubeconfig`: is the Kubeconfig file for AWS EKS cluster.
- `jenkins_gitops_key`: is the Gitops private key which has Read-Write access to the Git repos.
- `nexus_user`: is the Service User and Password for access to Nexus for Cordapps (only used in Corda).
- `aws_demo_vault_key`: is the private key to enable ssh access to Hashicorp Vault Server.
- `aws_demo_vault_token`: is the Root Token for Hashicorp Vault.
- `gmaps_key`: is the Google Maps API key for frontend (only used when deploying Supplychain application).
- `aws_jenkins`: is the AWS credentials for jenkins user on AWS IAM.

## Environment Changes

Following `environment` variables need to be updated in Jenkinsfile for your own environment

- `VAULT_SERVER`=[vault server ip address or domain name reachable from this server]
- `VAULT_PORT`=[vault server port]
- `VAULT_BASTION`=[vault bastion server address]
- `VAULT_PRIVATE_IP`=[vault server private ip address]

## Parameters

These can be changed when running manually, the automated Jenkins pipeline always use the default option):

1. `FORCE_ACTION` (default: no) To force rebuild `[ci skip]` commits in case of previous failure.
2. `RESET_ACTION` (default: yes) To have the option to NOT reset the network when running the pipeline.
3. `APIONLY_ACTION` (default: no) To run only API test on existing live network in case of previous failure.
4. `FABRIC_VERSION` (default: 1\_4\_4) To select the Fabric version.
5. `FABRIC_CONSENSUS` (default: raft) To select the Fabric consensus.
6. `CORDA_VERSION` (default: 4\_4) To select the Corda Opensource version.
7. `QUORUM_VERSION` (default: 2\_5\_0) To select the Quorum version (only 2\_5\_0 is supported for now)
8. `QUORUM_CONSENSUS` (default: ibft) To change the Quorum consensus.
9. `QUORUM_TM` (default: tessera) To change the Quorum Transaction manager.
10. `INDY_VERSION` (default: 1\_11\_0) To change the Indy version.
  - Default Corda Enterprise version is 4\_4. This is hardcoded in the jenkinsfile.
  - Default Besu settings are: Version 1\_4\_4, Consensus IBFT, Transaction Manager Orion.

## Setup on Jenkins

Configure Multi-branch pipeline with the forked repo as the source. In case you create the branches later, scan the pipeline to get new branches on Jenkins.

## Jenkins Stages

1. `Checkout SCM`: Manually checkout the branch and check for `[ci skip]` commits as they are skipped.
2. `Prepare build environment`: Creates the build directory and sets up the necessary files for build like `gitops.pem`, `vault.pem`, `kubeconfig`, `test.jsons`. Also creates the ssh-tunnel connection to Hashicorp Vault server.
3. `<branch>-settings`: Set env variables `CONSENSUS`, `VERSION` and `TM` based on the branch i.e. based on the DLT platform.
4. `Create Configuration File`: Downloads the config file (`main network.yaml`, `addorg.yaml` and `application.yaml`) depending on the `BRANCH_NAME`, `CONSENSUS`, `VERSION` and `TM` from `baf-configuration` and adds the secret parameters.
5. `Reset existing network`: Resets the network based on `application.yaml` as that should contain all the orgs.

6. `Deploy network`: Deploys the network based on `main network.yaml`.
7. `Add a new node`: Adds a new organization to the above network. This is not enabled for Indy currently.
8. `Deploy Supplychain-App`: Deploys the supplychain app. Not enabled for Indy. Corda Enterprise and Besu are in the future roadmap.
9. `Deploy Identity-App`: Deploys the Identity app. Only for Indy.
10. `Run SupplyChain API tests`: Runs Supplychain API test using newman. This step has a try-catch so that the whole pipeline does not fail if only API tests fail. Re-run the tests manually if only API tests fail. Not enabled for Indy. Corda Enterprise and Besu are in the future roadmap.
11. `Run Identity API tests`: Runs Identity API test using newman. This step has a try-catch so that the whole pipeline does not fail if only API tests fail. Re-run the tests manually if only API tests fail. Only for Indy.  
1 `Manual Approval for resetting the deployment`: Waits for 20 minutes before resetting the network. If you want to keep the network for demo, Abort at this stage.
12. `Reset network again`: Resets the network after the 20 minutes is over or you chose to reset. Keeps the network running if the previous step was aborted.

This section shows the sample applications that are provisioned by the Blockchain Automation Framework. If you haven't already, follow the [Getting Started](#) to setup the network for your desired DLT/Blockchain platform. We have provided sample applications to be deployed using the Blockchain Automation Framework.

## 7.1 Supplychain

### 7.1.1 Use case description

Welcome to the Supply Chain application which allows nodes to track products or goods along their chain of custody, providing everyone along the way with relevant data to their product. The implementation has been done for Hyperledger Fabric, Quorum and R3 Corda. The three will slightly differ in behavior but follow the same principles. There are two types of items that can be tracked, products and containers. Products are defined as such:

The creator of the product will be marked as its initial custodian. As a custodian, a node is able to package and unpackage goods. Packaging a good stores an item in an existing ContainerState structured as such:

Products being packaged will have their trackingID added to the contents list of the container. The Product will be updated when its container is updated. If a product is contained it can no longer be handled directly (ie transfer ownership of a single product while still in a container with others).

Any of the participants can scan the QR code to claim custodianship. History can be extracted via transactions stored on the ledger/ within the vault.

As mentioned before, items are identified by their QR code. These codes are meant to be generated about a product and used to interact with a product. The QR will store a parsable JSON body with the following format:

Also containers are identified by their QR code. These codes are meant to be generated about a container and use to interact with a container. The QR will store a parsable JSON body with the following format:

### 7.1.2 Prerequisites

- The supplychain application requires that nodes have subject names that include a location field in the x.509 name formatted as such: L=<lat>/<long>/<city>
- Fabric or Corda network of 1 or more organizations

### 7.1.3 Setup Guide

The setup process has been automated using Ansible scripts, GitOps, and Helm charts. The files have all been provided to use and require the user to populate the network.yaml file accordingly.

The playbooks are as listed:

- platforms\r3-corda\configuration\deploy\_cordapps.yaml
- platforms\hyperledger-fabric\configuration\chaincode-install-instantiate.yaml
- examples\supplychain-app\configuration\deploy-supplychain-app.yaml

These playbooks make use of several roles defined within

- create/corda/api\_components
- create/corda/api\_values
- create/fabric/api\_components
- create/fabric/api\_values
- create/frontend

They are responsible for creating the helm release files for the platform specific rest service, the nodejs api abstraction layer, and the front end.

### Generating QR code

Products and containers are identified by their QR code. These codes need to be generated for interacting with products and containers. A module for generating QR is provided along with this sample. The QR generating module can be found on the location:

```
sample-home/generate/qr
```

Module for generating QR code is divided into two sections. The first section is generating QR code for the product. Codes from the first section are meant to be generated about a product and are used to interact with the product.

Following are the required fields in the QR Code generation form for the products along with the format:

The second section is provided for generating the QR code for the containers. Codes from this section are meant to be generated about a container and used to interact with a container.

Following are the required fields in the QR Code generation form for the containers along with the format:

## 7.2 Indy RefApp

### 7.2.1 Use case description

Welcome to the Indy Ref App which allows nodes to implement the concept of digital identities using blockchain. There are 3 components

- Alice: Alice is the end user and a student.
- Faber: Faber is the university.
- Indy Webserver

In this usecase, Alice obtains a Credential from Faber College regarding the transcript. A connection is build between Faber College and Alice (onboarding process).Faber College creates and sends a Credential Offer to Alice. Alice creates a Credential Request and sends it to Faber College.Faber College creates the Credential for Alice. Alice now receives the Credential and stores it in her wallet.

### 7.2.2 Pre-requisites

A network with 2 organizations:





- Authority
  - 1 Trustee
- University
  - 4 Steward nodes
  - 1 Endorser A Docker repository

Find more at [Indy-Ref-App](#)













## BAF current roadmap

Legend of annotations:








Mark	Description
	work to do
	work completed
	on-going work
	stretch goal
	on hold
	depricated
	in use / valid

## 8.1 General





-  Minikube support for existing platforms
-  Improve the existing `readthedocs` documentations
-  **BAF DLT and software tools upgrade:**
  -  Helm3 upgrade
  -  HAProxy upgrade to latest stable
  -  EKS upgrade to 1.16

-  HL Fabric version upgrade
-  Ambassador upgrade to latest stable
-  Flux upgrade for helm3 operator support
-  Java version upgrades




## 8.2 Platforms

-  Besu network provisioning
-  Support for R3 Corda Enterprise Version
-  Hyperledger Indy network provisioning
-  Quorum network provisioning
-  Support for Hyperledger Fabric v2.0
-  Key storage and management
-  Implementing couch DB













## 8.3 Application

-  Hyperledger Besu reference application
-  R3 Corda Enterprise reference application
-  Quorum supplychain application integration
-  Hyperledger Indy reference application

## 8.4 Architecture

-  Besu architecture blueprint
-  R3 Corda Enterprise Version architecture blueprint
-  Couch DB architecture blueprint

## 8.5 Histroic DLT/Blockchain support releases

Feature Name	Release	Status
Hyperledger Fabric v2.0	0.5.0	
R3 Corda v4.4	0.5.0	
Hyperledger Indy v1.11.0	0.5.0	
Quorum v2.5.0	0.4.1	
Quorum v2.1.1	0.4.0	
Quorum Architecture blueprint	0.4.0	
Hyperledger Indy v1.9.0	0.4.0	
Hyperledger Fabric v1.4.4	0.3.1	
Hyperledger Indy Architecture blueprint	0.3.0	
R3 Corda v4.1	0.3.0	
Hyperledger Fabric v1.4.0	0.2.0	
R3 Corda v4.0	0.2.0	



## Architecture Reference

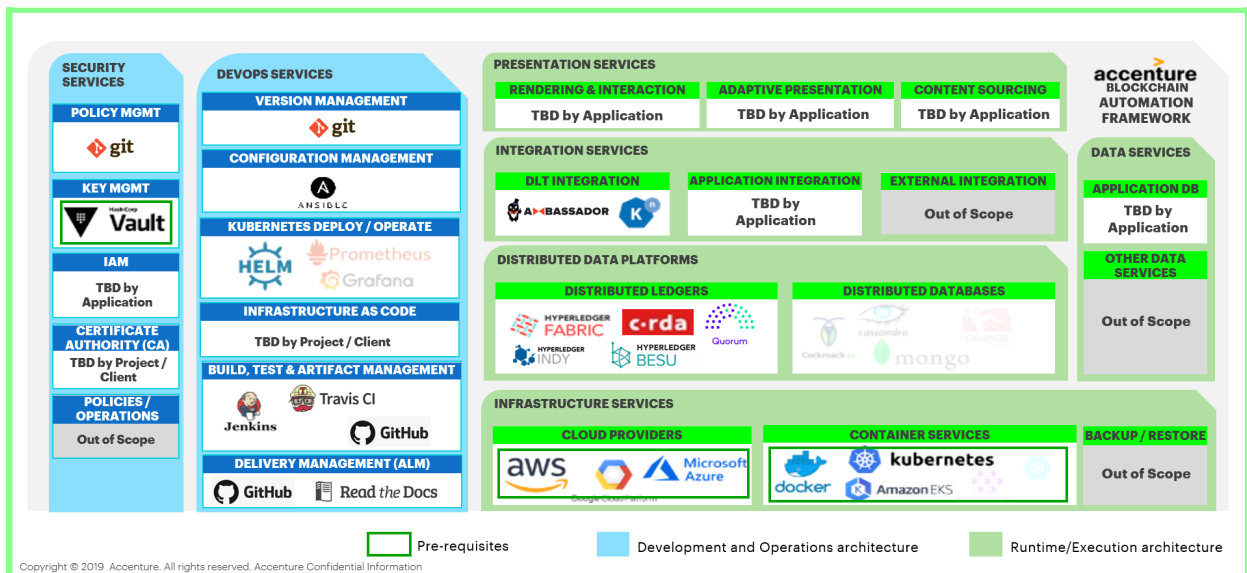


Fig. 1: Figure: Blockchain Automation Framework Physical Architecture

## 9.1 Security Services

These are the services to enable the security of cryptographic keys, users, nodes and transactions along with the infrastructure supporting those services.

### 9.1.1 Policy Management

Policy management is the process of creating, communicating, and maintaining policies and procedures within an organization. Policy Management is a key feature used in development as well as operational phase of any product as it dictates who has what control in the dev/test/prod environment(s).

In the Blockchain Automation Framework (BAF), Policy Management is provided by the Git repository. BAF uses *GitOps* for deployment and operations, hence all policies are defined in the Git repository. Git branches with appropriate rights to users is maintained for releases in each environment. Read/write access, admin access to git repository, access to add access keys in repository, pull request based merge in main branch are some of the key features that is used in BAF.

### 9.1.2 Key Management

Key Management is the process of overseeing the generation, exchange, storage, use and destruction of cryptographic keys. Key Management is an important consideration for blockchain as all transactions in blockchain are signed using digital keys. Loss of keys can lead to financial loss as well as brand impact to the organization conducting the transaction.

The Blockchain Automation Framework uses *Hashicorp Vault* to hold secrets that are used by the DLT/Blockchain platform. A secret is anything that you want to tightly control access to (e.g. API keys, passwords, certificates). Vault provides a unified interface to any secret, while providing tight access control and recording a detailed audit log. *Hashicorp Vault* provides an abstraction on top of a Cloud KMS and does not create Cloud Platform lock-in. See the *Platform-Specific Reference Guides* for specific details on the structure of the Vault. Vault is a pre-requisite for BAF and should be configured and available before the automation is triggered.

### 9.1.3 Identity and Access Management (IAM)

Identity and Access Management (IAM) is the process of defining and managing the access privileges of network users and determining how users are granted or denied those privileges. IAM is the front door for all blockchain applications and hence has to be designed upfront to reduce risk. Strong authentication techniques and user level permissioning will help shift left some of the security concerns.

The Blockchain Automation Framework does not provide IAM controls. This is to be developed and applied by the application/use-case.

### 9.1.4 Certificate Authority (CA)

A Certificate Authority dispenses certificates to different actors. These certificates are digitally signed by the CA and bind together the actor with the actor's public key (and optionally with a comprehensive list of properties). As a result, if one trusts the CA (and knows its public key), it can trust that the specific actor is bound to the public key included in the certificate, and owns the included attributes, by validating the CA's signature on the actor's certificate.

For test and dev environments, the Blockchain Automation Framework generates certificates and keys (for all Platforms) and also provides CA servers (Fabric only).

For production use, generation of certificates, keys and CA servers via the Blockchain Automation Framework is not recommended. The existing certificates and keys can be placed in Vault in the paths described under subsections of *Platform-Specific Reference Guides*.

### 9.1.5 Policies/Operations

Policies/Operations refers to the actual security policies that an organization may/should have governing their business processes, operations and management.

This part of the reference architecture is out of scope for the Blockchain Automation Framework.

## 9.2 DevOps Services

These services enable the development of on-ledger (e.g. smart contracts) or off-ledger services based on SDK's and IDE's (e.g. Web APIs) including the maintenance, monitoring and administration of the distributed ledger and its on- and off-ledger services.

### 9.2.1 Version Management

Version Management capabilities enable change control of smart contract and decentralized applications. This enables developers and operators to track different version of the code as well as releases.

The Blockchain Automation Framework utilizes Git as the version management tool.

### 9.2.2 Configuration Management

Configuration management involves automation of scripts and ad-hoc practices in a consistent, reliable and secure way. Configuration Management enables operators to set-up DLT/Blockchain networks idempotently by using minimum configuration changes.

The Blockchain Automation Framework utilizes Ansible for configuration management. Ansible features a state driven, goal oriented resource model that describes the desired state of computer systems and services, not the paths to get them to this state. No matter what state a system is in, Ansible understands how to transform it to the desired state (and also supports a “dry run” mode to preview needed changes). This allows reliable and repeatable IT infrastructure configuration, avoiding the potential failures from scripting and script-based solutions that describe explicit and often irreversible actions rather than the end goal.

### 9.2.3 Kubernetes Deploy/Operate

Kubernetes Deploy/Operate consists of the services that are used to deploy desired state of various services on Kubernetes clusters. It is also used for maintenance and operations of these services.

The Blockchain Automation Framework uses Helm to achieve this. Helm uses a packaging format called charts. A chart is a collection of files that describe a related set of Kubernetes resources. A single chart might be used to deploy something simple, like a memcached pod, or something complex, like a full web app stack with HTTP servers, databases, caches, and so on, which in our case, is the desired blockchain platform. While using helm, we can deploy a set of services and deployments together as a release.

### 9.2.4 Infrastructure as Code

Infrastructure as Code (IaC) is the process of managing and provisioning cloud hardware through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools. IaC can be versioned and hence, maintained easily and can be used to deploy cloud environments idempotently.

This part of the reference architecture is out of scope for the Blockchain Automation Framework.

### 9.2.5 Build, Test, and Artifact Management

Build, Test, and Artifact Management capabilities enable continuous delivery management by ensuring automation of the build and deployment of artefacts.

The Blockchain Automation Framework (BAF) uses TravisCI for running static tests, building and storing of Docker images. Jenkins Pipelines (as code) are also available for continuous deployment/reset of DLT network. Artefact management is not implemented yet, but GitHub Releases can be used for this.

### 9.2.6 Delivery Management

Delivery Management is the process where all software, artifacts and data from disparate tools used to move a product or feature from initial idea to max adoption are integrated into a unified common data layer, with the key information connected and easily accessible, giving each individual and team an unprecedented level of insight into bottlenecks and inefficiencies dramatically improving the speed at which better software gets to users safely.

As it is opensource and a hyperledger-labs project, the Blockchain Automation Framework integrates with GitHub for reporting and tracking new features, bugs/issues and releases. BAF uses ReadTheDocs for sharing documentation. In specific implementations, the Blockchain Automation Framework can be integrated with tools like Jira and Confluence.

## 9.3 Presentation Services

The presentation services specify how the application will be provided to the end-user. It also defines the on-ledger and off-ledger services and capabilities via different channels.

This part of the reference architecture is out of scope for Blockchain Automation Framework (BAF) and will be determined by the application using BAF.

## 9.4 Integration Services

These are combination of the services to interact with on- and off-ledger services via APIs or ledger protocols including runtime and operations services.

### 9.4.1 DLT Integration

DLT integration refers to how the presentation services will talk to the DLT Platform. This will depend on the presentation service as such.

The Blockchain Automation Framework provides a sample application *Supplychain*, which uses Express Nodejs API as the integration layer to talk to the underlying DLT platform. Each DLT/Blockchain platform also enables this by providing SDKs or APIs themselves.

### 9.4.2 Application Integration

Application Integration refers to how the application will talk to different components of the same application.

This part of the reference architecture is out of scope for the Blockchain Automation Framework (BAF) and will be determined by the application using BAF.

### 9.4.3 External Integration

External integration is required when the blockchain application interfaces with systems outside of the application or DLT platform.

This part of the reference architecture is out of scope for the Blockchain Automation Framework (BAF) and will be determined by the application using BAF.

## 9.5 Distributed Data Platforms

Distributed Data Platforms form the core of any distributed architecture solution. The Blockchain Automation Framework (BAF) aims to support both Distributed Ledgers and Distributed Databases. BAF currently supports DLT/Blockchain Platforms: [Corda](#), [Hyperledger Fabric](#), [Hyperledger Indy](#), [Hyperledger Besu](#), and [Quorum](#).

## 9.6 Infrastructure Services

Infrastructure services refer to the various services needed to run or deploy the different services of a distributed ledger architecture.

### 9.6.1 Cloud Providers

A Cloud Provider is a company that delivers cloud computing based services with features like scalability and easy maintainance.

The Blockchain Automation Framework is built on Kubernetes, so will run on any Cloud provider providing Kubernetes as a service; this includes private and hybrid clouds.

### 9.6.2 Container Services

Container services allows users to deploy and manage containers using container based virtualization. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

The Blockchain Automation Framework (BAF) uses 2 containerization technologies: Docker and Kubernetes. Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications. Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.

### 9.6.3 Backup/Restore

Disaster recovery involves a set of policies, tools and procedures to enable the recovery of vital technology infrastructure and systems following a natural or human-induced disaster. Even though blockchain applications are self replicating, complete auto recovery is not always possible. Therefore it is important to have guidelines around backing up the data in a distributed store and restoring it using a conventional restoring mechanism. Backup is the process of copying and archiving data. Restore is the process of returning data that has been lost, stolen or damaged from secondary storage.

This part of the reference architecture is out of scope for the Blockchain Automation Framework.

## 9.7 Other Data Services

Data services are related to on-ledger storage and data processing.

This part of the reference architecture is out of scope for the Blockchain Automation Framework.

## 9.8 Platform-Specific Reference Guides

### 9.8.1 Corda Enterprise Architecture Reference

#### Kubernetes

#### Peer Nodes

The following diagram shows how Corda peer nodes will be deployed on your Kubernetes instance.

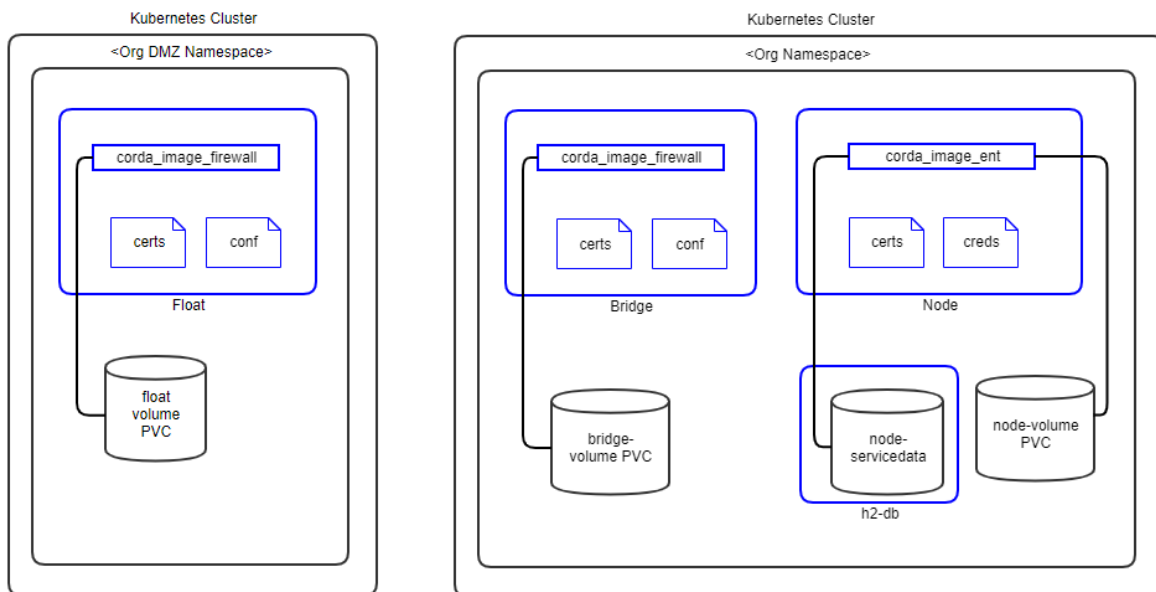


Figure:

R3 Corda Enterprise Kubernetes Deployment - Peers

#### Notes:

1. Pods are shown in blue in the diagram.
2. Certificates are mounted as in-memory volumes from the *Vault*.
3. The h2 database is a separate pod running in the same namespace. In future release, PostgreSQL will be implemented as well.
4. All storage uses a Kubernetes Persistent Volume.
5. Release 0.6.0.0 does not implement Corda firewall components. These will be implemented in later releases based on demand.

## Support Services

The following diagram shows how the Corda Enterprise Network Map Services (**Identity Manager**, **Networkmap**, **Signer** and **Notary**) will be deployed on your Kubernetes instance(s).

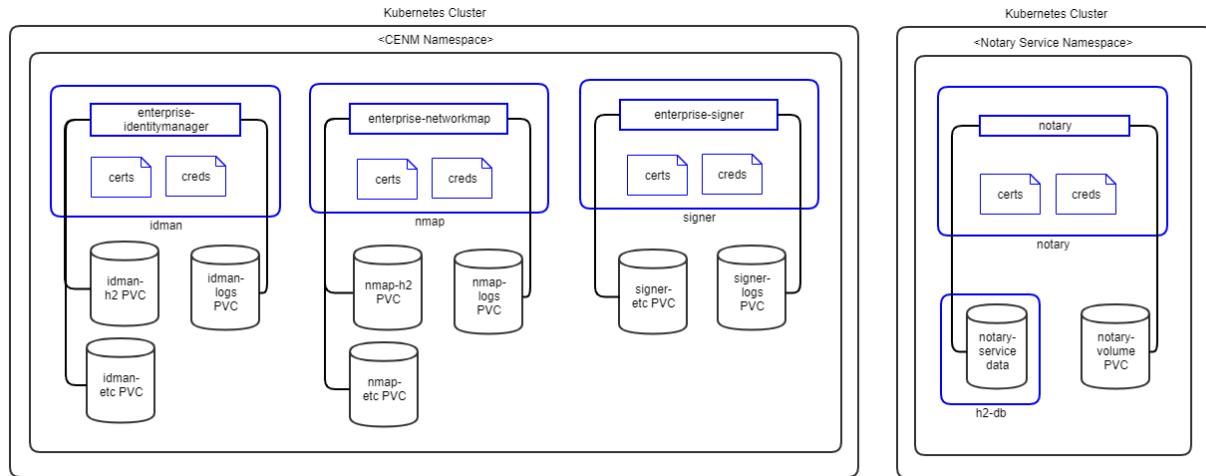


Figure:

### R3 Corda Kubernetes Deployment - CENM Services

#### Notes:

1. Pods are shown in blue in the diagram.
2. Certificates are mounted as in-memory volumes from the *Vault*.
3. All CENM pods (except Notary) have separate H2 volume for data storage. In future release, PostgreSQL will be implemented as well.
4. Notary service has a separate H2 pod for data storage. In future release, PostgreSQL will be implemented as well.
5. All storage uses a Kubernetes Persistent Volume.
6. Release 0.6.0.0 implements Notary in the same namespace as other CENM services. They will be separated when HA Notary is implemented in later releases.

## Components

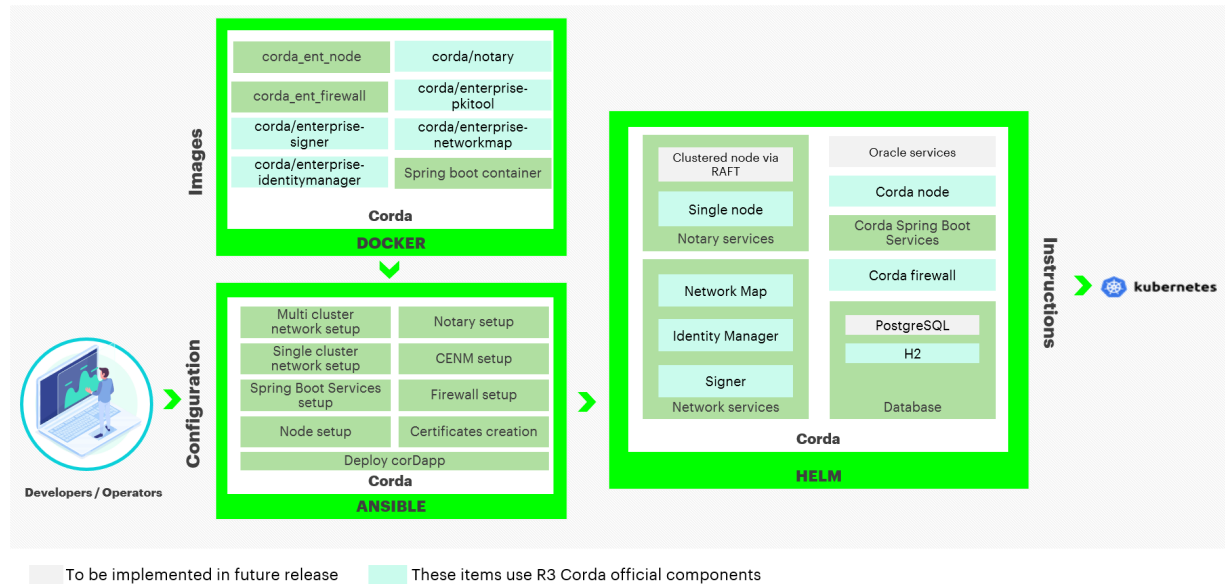


Figure:

### Corda Enterprise Components

## Docker Images

For Corda Enterprise, the `corda_ent_node` and `corda_ent_firewall` docker images should be built and put in a private docker registry. Please follow [these instructions](#) to build docker images for Corda Enterprise.

The official Corda images are available on [Docker Hub](#). These are evaluation only, for production implementation, please acquire licensed images from R3, upload them into your private docker registry and update the tags accordingly.

Following Corda Docker Images are used and needed by the Blockchain Automation Framework.

- Corda Network Map Service (Built as per [these instructions](#))
- Corda Identity Manager Service
- Corda Signer
- Corda PKITool (Built as per [these instructions](#))
- Corda Notary (Built as per [these instructions](#))
- Corda Node (Built as per [these instructions](#))
- Corda Firewall (Built as per [these instructions](#))

## Ansible Playbooks

Detailed information on ansible playbooks can be referred [here](#) and the execution process can be referred [here](#).

## Helm Charts

Detailed information on helm charts can be referred [here](#).

## Vault Configuration WIP

The Blockchain Automation Framework stores their `crypto` and `credentials` immediately within the secret secrets engine.

- `secrets/notary/credentials/database` - Contains password for notary database for admin and user:

```
sa="newh2pass" notaryUser1="xyz1234" notaryUser2="xyz1236"
```

- `secrets/notary/credentials/keystore` - Contains password for notary keystore:

```
keyStorePassword="newpass" trustStorePassword="newpass" defaultTrustStorePassword
↪ "=trustpass" defaultKeyStorePassword="cordacadepass" sslkeyStorePassword="sslpass"
↪ ssltrustStorePassword="sslpass"
```

- `secrets/notary/credentials/networkmappassword` - Contains password for networkmap:

```
sa="admin"
```

- `secrets/notary/credentials/rpcusers` - Contains password for rpc users:

```
notaryoperations="usera" notaryoperations1="usera" notaryoperations2="usera"
↪ notaryadmin="usera"
```

- `secrets/notary/credentials/vaultroottoken` - Contains password for vault root token in the format:

```
rootToken="<vault.root_token>"
```

- `secrets/<org-name>/credentials/database` - Contains password for notary database for admin and user:

```
sa="newh2pass" <org-name>User1="xyz1234" <org-name>User2="xyz1236"
```

- `secrets/<org-name>/credentials/keystore` - Contains password for notary keystore:

```
keyStorePassword="newpass" trustStorePassword="newpass" defaultTrustStorePassword
↪ "=trustpass" defaultKeyStorePassword="cordacadepass" sslkeyStorePassword="sslpass"
↪ ssltrustStorePassword="sslpass"
```

- `secrets/<org-name>/credentials/networkmappassword` - Contains password for networkmap:

```
sa="admin"
```

- `secrets/<org-name>/credentials/rpcusers` - Contains password for rpc users:

```
<org-name>operations="usera" <org-name>operations1="usera" <org-name>operations2=
↪ "usera" <org-name>admin="usera"
```

- `secrets/<org-name>/credentials/vaultroottoken` - Contains password for vault root token in the format:

```
rootToken="<vault.root_token>"
```

The complete Corda Enterprise Certificate and key paths in the vault can be referred [here](#).

## 9.8.2 Certificate Paths on Vault for Corda Enterprise

All values must be Base64 encoded files as BAF decodes them.

**For CENM**

**For Node/Peer Organization**

## 9.8.3 Corda Opensource Architecture Reference

**Kubernetes**

**Peer Nodes**

The following diagram shows how Corda peer nodes will be deployed on your Kubernetes instance.

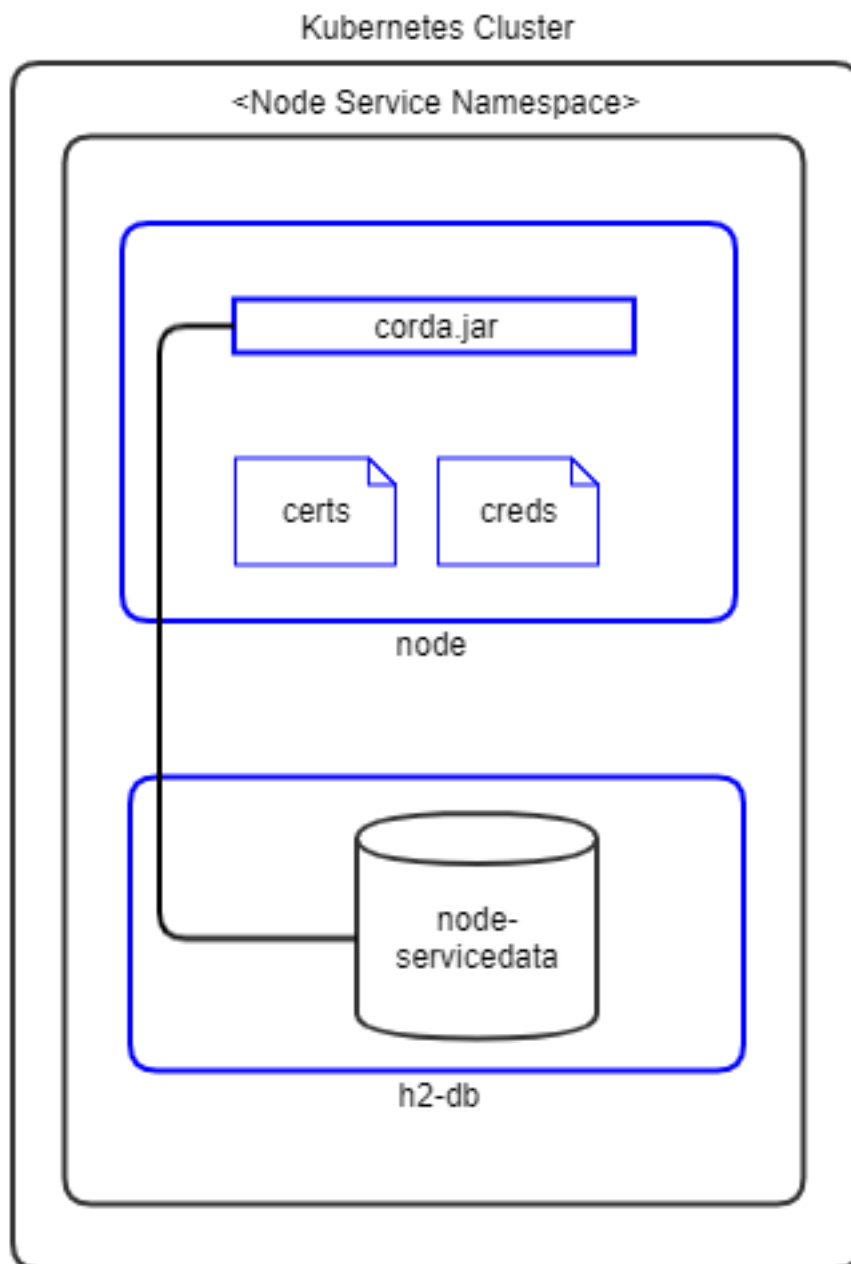


Figure: R3 Corda Ku-

bernetes Deployment - Peers

**Notes:**

1. Pods are shown in blue in the diagram.
2. Certificates are mounted as in-memory volumes from the *vault*.
3. The node-pod runs *corda.jar*.
4. The h2 database is a separate pod running in the same namespace
5. All storage uses a Kubernetes Persistent Volume.

## Support Services

The following diagram shows how the Corda Support Services (**Doorman**, **Networkmap** and **Notary**) will be deployed on your Kubernetes instance.

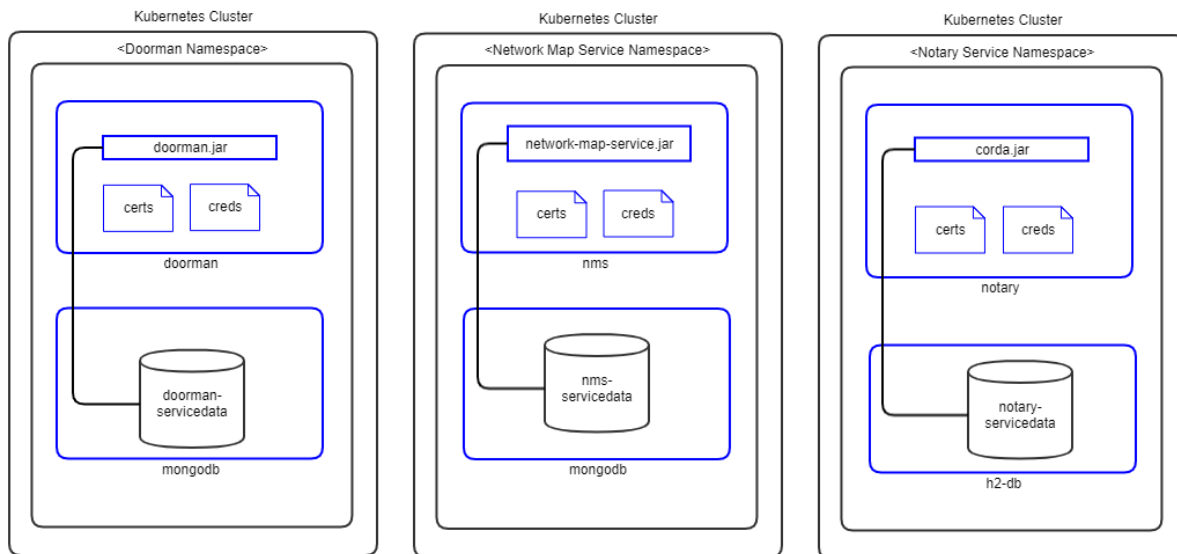


Figure:

### R3 Corda Kubernetes Deployment - Support Services

#### Notes:

1. Pods are shown in blue in the diagram.
2. Certificates are mounted as in-memory volumes from the *vault*.
3. Doorman and Networkmap services have a separate MongoDB pod for data storage.
4. Notary service has a separate H2 pod for data storage.
5. All storage uses a Kubernetes Persistent Volume.

## Components

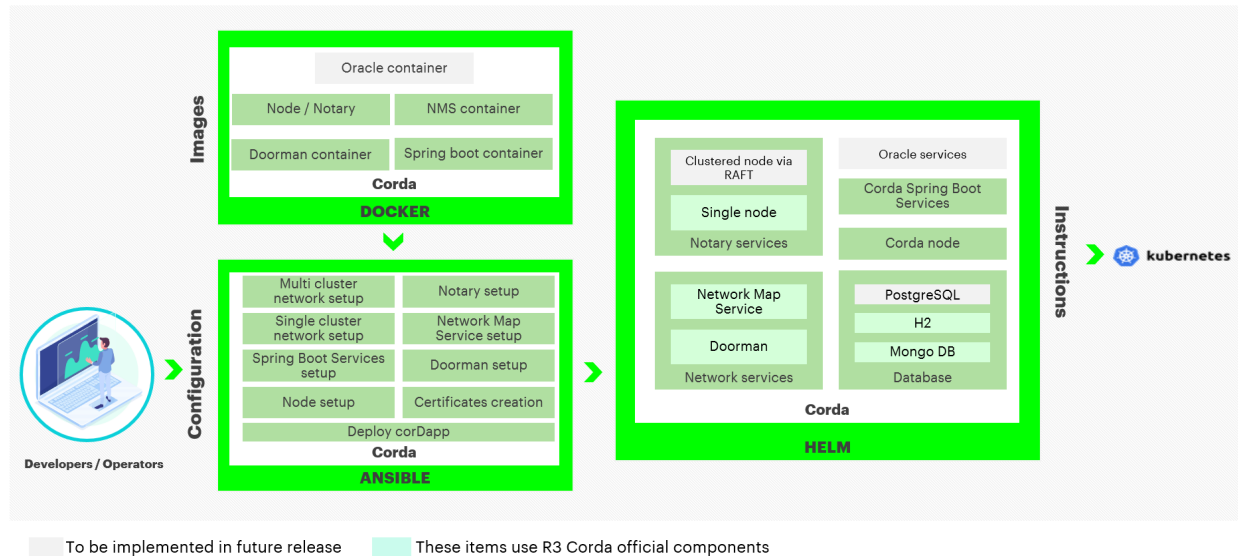


Figure:

### Corda Components

## Docker Images

The Blockchain Automation Framework creates/provides a set of Corda Docker images that can be found in the [Hyperledger-Labs repository](#) or can be built as per [configuring prerequisites](#). The following Corda Docker Images are used and needed by the Blockchain Automation Framework.

- Corda Network Map Service
- Corda Doorman Service
- Corda Node

## Ansible Playbooks

Detailed information on ansible playbooks can be referred [here](#) and the execution process can be referred [here](#)

## Helm Charts

Detailed information on helm charts can be referred [here](#)

## Vault Configuration

The Blockchain Automation Framework stores their `crypto` and `credentials` immediately within the secret secrets engine.

- `secrets/doorman/credentials/mongodb` - Contains password for doorman mongodb database.

```
mongodbPassword="admin"
```

- `secrets/doorman/credentials/userpassword` - Contains password for doorman mongodb database user:

```
sa="newdbnm"
```

- secrets/networkmap/credentials/mongodb - Contains password for networkmap mongodb database:

```
mongodbPassword="newdbnm"
```

- secrets/networkmap/credentials/userpassword - Contains password for networkmap mongodb database user:

```
sa="admin"
```

- secrets/notary/credentials/database - Contains password for notary database for admin and user:

```
sa="newh2pass" notaryUser1="xyz1234" notaryUser2="xyz1236"
```

- secrets/notary/credentials/keystore - Contains password for notary keystore:

```
keyStorePassword="newpass" trustStorePassword="newpass" defaultTrustStorePassword  
↪="trustpass" defaultKeyStorePassword="cordacadepass" sslkeyStorePassword="sslpass"  
↪ssltrustStorePassword="sslpass"
```

- secrets/notary/credentials/networkmappassword - Contains password for networkmap:

```
sa="admin"
```

- secrets/notary/credentials/rpcusers - Contains password for rpc users:

```
notaryoperations="usera" notaryoperations1="usera" notaryoperations2="usera"  
↪notaryadmin="usera"
```

- secrets/notary/credentials/vaultroottoken - Contains password for vault root token in the format:

```
rootToken="<vault.root_token>"
```

- secrets/<org-name>/credentials/database - Contains password for notary database for admin and user:

```
sa="newh2pass" <org-name>User1="xyz1234" <org-name>User2="xyz1236"
```

- secrets/<org-name>/credentials/keystore - Contains password for notary keystore:

```
keyStorePassword="newpass" trustStorePassword="newpass" defaultTrustStorePassword  
↪="trustpass" defaultKeyStorePassword="cordacadepass" sslkeyStorePassword="sslpass"  
↪ssltrustStorePassword="sslpass"
```

- secrets/<org-name>/credentials/networkmappassword - Contains password for networkmap:

```
sa="admin"
```

- secrets/<org-name>/credentials/rpcusers - Contains password for rpc users:

```
<org-name>operations="usera" <org-name>operations1="usera" <org-name>operations2=  
↪"usera" <org-name>admin="usera"
```

- secrets/<org-name>/credentials/vaultroottoken - Contains password for vault root token in the format:

```
rootToken="<vault.root_token>"
```

The complete Certificate and key paths in the vault can be referred [here](#)

## 9.8.4 Certificate Paths on Vault for Corda Network

### For Networkmap

### For Doorman

### For Notary organization

### For Node/Peer Organization

## 9.8.5 Hyperledger Fabric Architecture Reference

### Kubernetes

#### Peer Nodes

The following diagram shows how Hyperledger Fabric peer nodes will be deployed on your Kubernetes instance.

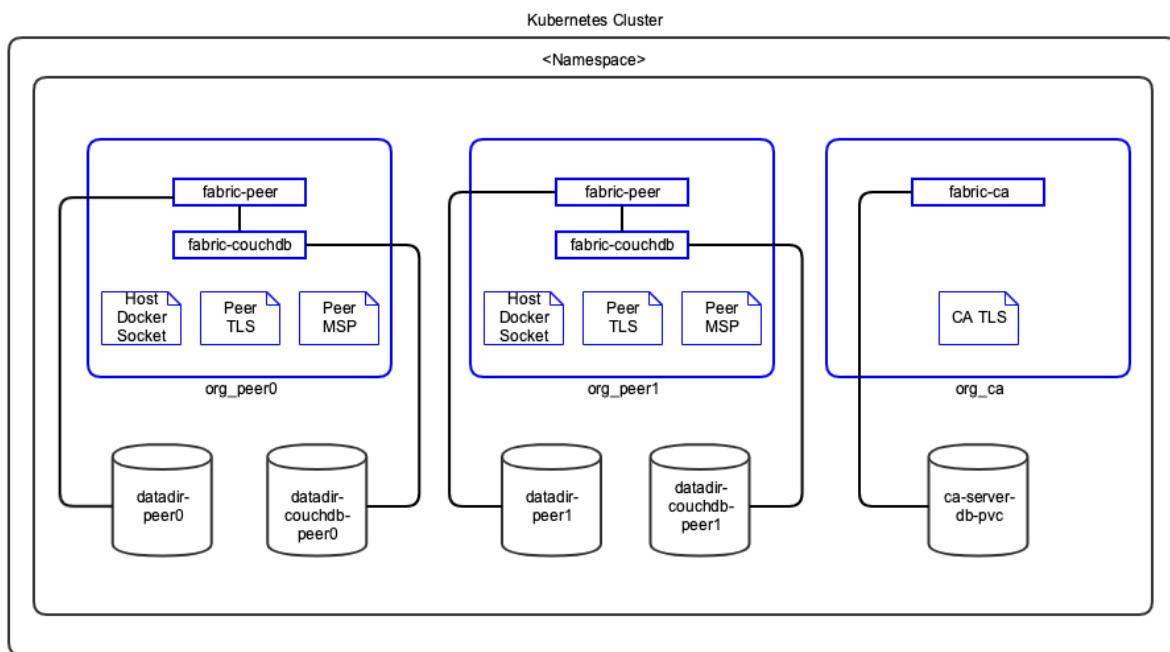


Figure:

Hyperledger Fabric Kubernetes Deployment - Peers

**Notes:**

1. Pods are shown in blue in the diagram.
2. Each peer pod will have both `fabric-peer` and `fabric-couchdb` containers running. Since they are in the same pod, Kubernetes always schedules them on the same VM and they can communicate to each other through localhost. This guarantees minimal latency between them.
3. Host VM's Docker socket is attached to peer pod so it can create chaincode containers. Kubernetes is not aware of these containers.
4. TLS and MSP certificates are mounted as in-memory volumes from the *Vault*.
5. The storage uses a Kubernetes Persistent Volume.

**Orderer Nodes**

The following diagram shows how Hyperledger Fabric orderer will be deployed on your Kubernetes instance.

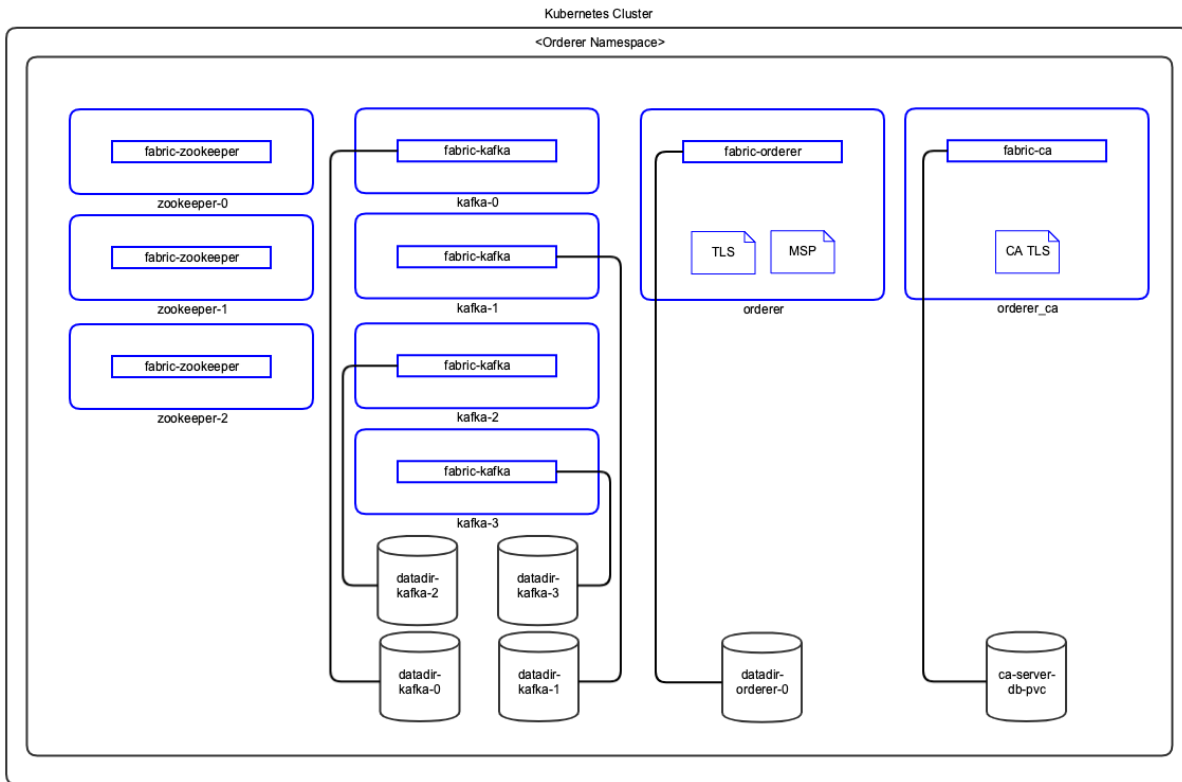


Figure:

Hyperledger Fabric Kubernetes Deployment - Orderer

**Notes:**

1. Pods are shown in blue in the diagram.
2. TLS and MSP certificates are mounted as in-memory volumes from the *Vault*.
3. The storage uses a Kubernetes Persistent Volume.

## Components

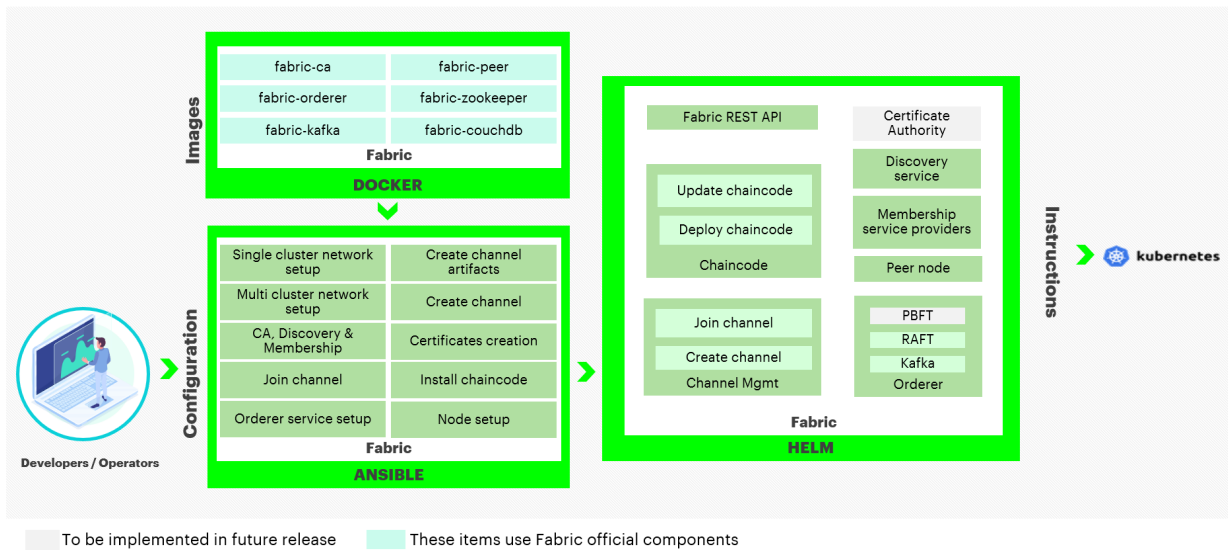


Figure:

Hyperledger Fabric Components

## Docker Images

The Blockchain Automation Framework uses the officially published Hyperledger Fabric Docker images from [hub.docker.com](https://hub.docker.com). The following Hyperledger Fabric Docker Images are used by the Blockchain Automation Framework.

- `fabric-ca` - Hyperledger Fabric Certificate Authority
- `fabric-couchdb` - CouchDB for Hyperledger Fabric Peer
- `fabric-kafka` - Kafka for Hyperledger Fabric Orderer
- `fabric-orderer` - Hyperledger Fabric Orderer
- `fabric-peer` - Hyperledger Fabric Peer
- `fabric-zookeeper` - Zookeeper for Hyperledger Fabric Orderer

## Ansible Playbooks

Detailed information on ansible playbooks can be referred [here](#) and the execution process can be referred [here](#)

## Helm Charts

Detailed information on helm charts can be referred [here](#)

## Vault Configuration

The Blockchain Automation Framework stores their `crypto` and `credentials` immediately within the secret secrets engine.

- `secret/credentials/ordererOrganizations/<orderer-org>/ca` - Contains password for the Orderer CA Bootstrap user in the format:

```
user="${ORDERER_NAMESPACE}-adminpw
```

- `secret/credentials/peerOrganizations/<org1>/ca` - Contains password for the Org Peers CA Bootstrap user in the format:

```
user="${NAMESPACE}-adminpw
```

- `secret/credentials/peerOrganizations/<org1>/<peern>couchdb` - Contains the password for the Peer's CouchDB user in the format:

```
pass="${NAMESPACE}-peer-${n}-adminpw
```

The complete Certificate and key paths in the vault can be referred [here](#).

## 9.8.6 Certificate Paths on Vault for Fabric Network

### 9.8.7 For each channel

Path	Key (for Vault)	Type
/secret/crypto/ordererOrganizations/	genesisBlock	Genesis

#### For each orderer organization

---

#### For each peer organization

## 9.8.8 Hyperledger Indy Architecture Reference

### Kubernetes

### Peer Nodes

The following diagram shows how Hyperledger Indy peer nodes will be deployed on your Kubernetes instance.

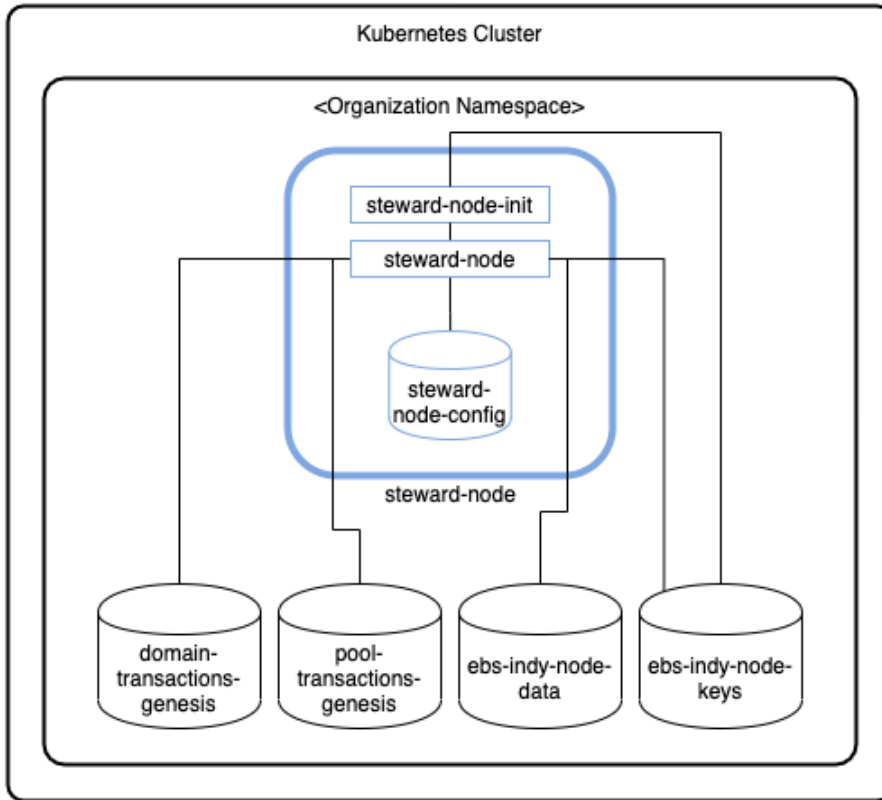


Figure:

Hyperledger Indy Kubernetes Deployment - Peers

**Notes:**

1. Pods are shown in blue in the diagram.
2. Each StatefulSet will have `steward-node-init` for initialization (read crypto from Vault) and `steward-node` containers running. Since they are in the same pod, Kubernetes always schedules them on the same VM and they can communicate to each other through localhost. This guarantees minimal latency between them.
3. The storage uses a Kubernetes Persistent Volume.

## Components

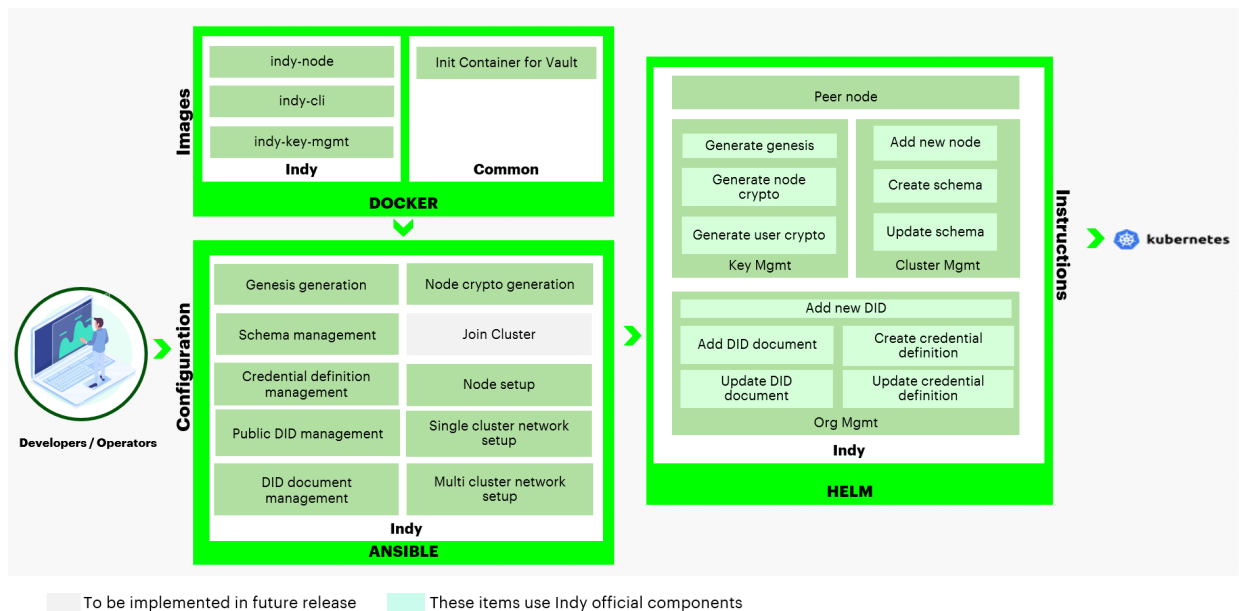


Figure:

Hyperledger Indy Components

## Docker Images

The Blockchain Automation Framework creates/provides own Docker images, which are based on Ubuntu and consist with official Hyperledger Indy libraries (indy-plenum and indy-node).

- **indy-cli** - Docker image contains Indy CLI, which is used to issue transactions against an Indy pool.
- **indy-key-mgmt** - Docker image for indy key management, which generates identity crypto and stores it into Vault or displays it onto the terminal in json format.
- **indy-node** - Docker image of an Indy node (runs using a Steward identity).

## Ansible Playbooks

Detailed information on ansible playbooks can be referred [here](#) and the execution process can be referred [here](#).

## Helm Charts

Detailed information on helm charts can be referred [here](#).

## Vault Configuration

The Blockchain Automation Framework stores their `crypto` immediately within the secret secrets engine. The `crypto` is stored by each organization under `/org_name_lowercase` - it contains private/public keys, dids and seeds.

The complete key paths in the vault can be referred [here](#).

### 9.8.9 Certificate Paths on Vault for Indy Network

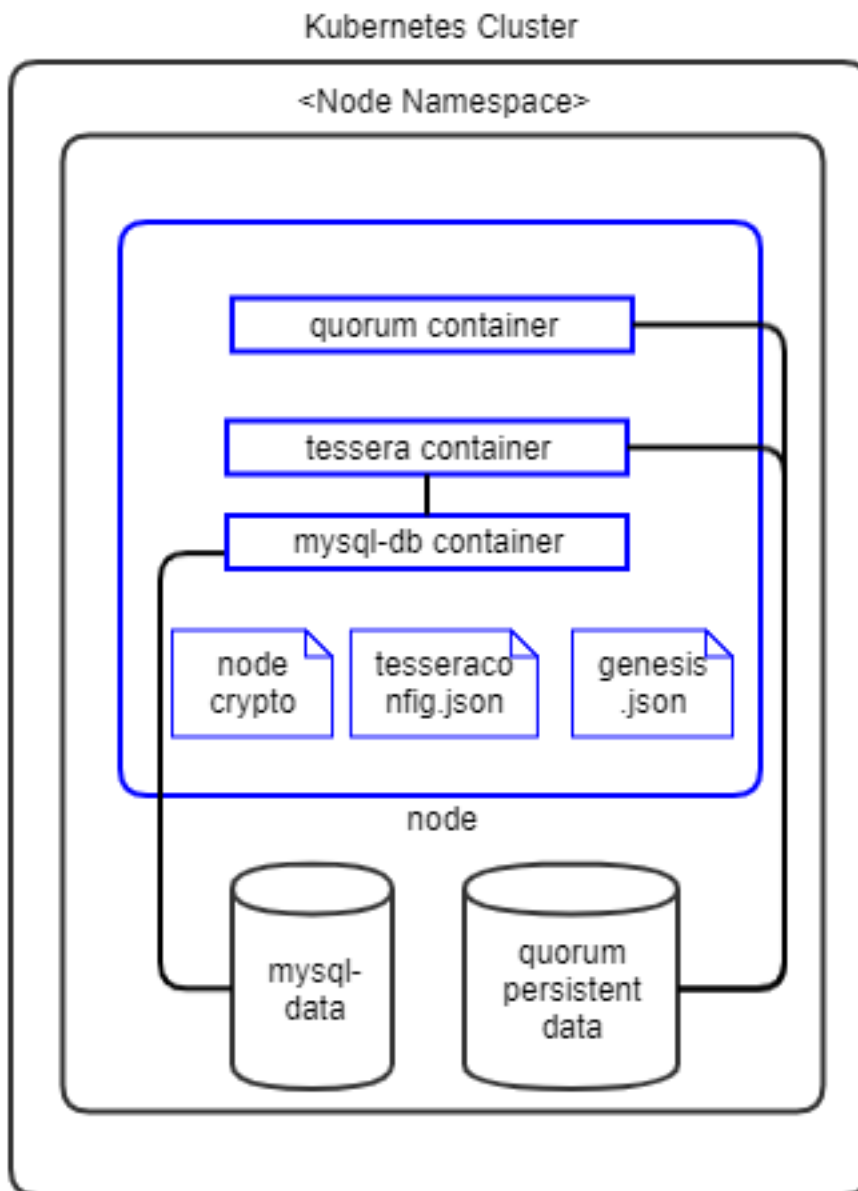
For each organization

### 9.8.10 Quorum Architecture Reference

Kubernetes

Nodes with Tessera

The following diagram shows how Quorum peer nodes with Tessera TM will be deployed on your Kubernetes instance.



bernetes Deployment - Tessera Peers

Figure: Quorum Ku-

### Notes:

1. Pods are shown in blue in the diagram.
2. Each peer pod will have three init-containers: `certificates-init` to read crypto from Vault, `mysql-init` to initialize MySQL DB and `quorum-genesis-init-container` to generate genesis block.
3. Each peer pod will then have three containers: `mysql-db`, `tessera` and `quorum` containers running. Since they are in the same pod, Kubernetes always schedules them on the same VM and they can communicate to each other through localhost. This guarantees minimal latency between them.
4. The storage uses a Kubernetes Persistent Volume.

### Nodes with Constellation

The following diagram shows how Quorum peer nodes with Constellation TM will be deployed on your Kubernetes instance.

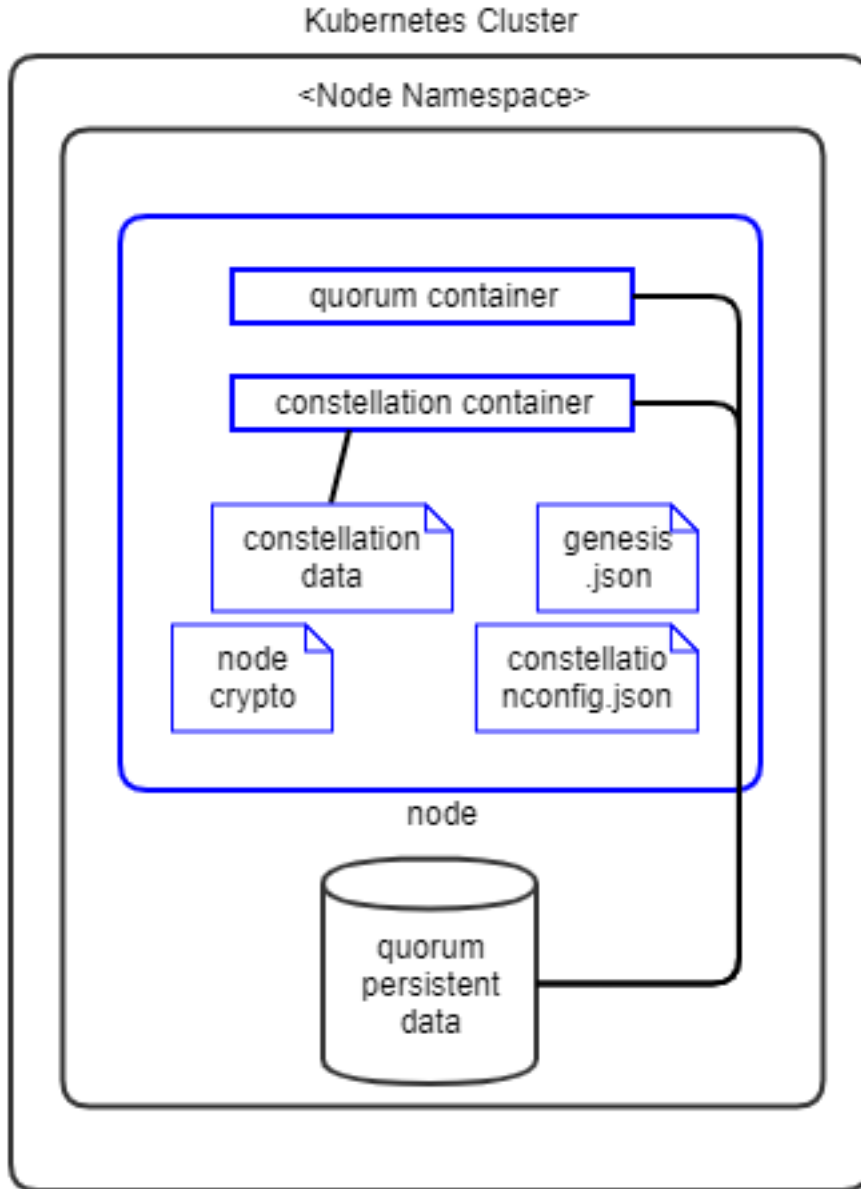


Figure: Quorum Ku-

bernetes Deployment - Constellation Peers

**Notes:**

1. Pods are shown in blue in the diagram.
2. Each peer pod will have two init-containers: `certificates-init` to read crypto from Vault and `quorum-genesis-init-container` to generate genesis block.
3. Each peer pod will then have two containers: `constellation` and `quorum` containers running. Since they are in the same pod, Kubernetes always schedules them on the same VM and they can communicate to each other through localhost. This guarantees minimal latency between them.
4. The storage uses a Kubernetes Persistent Volume.

## Components

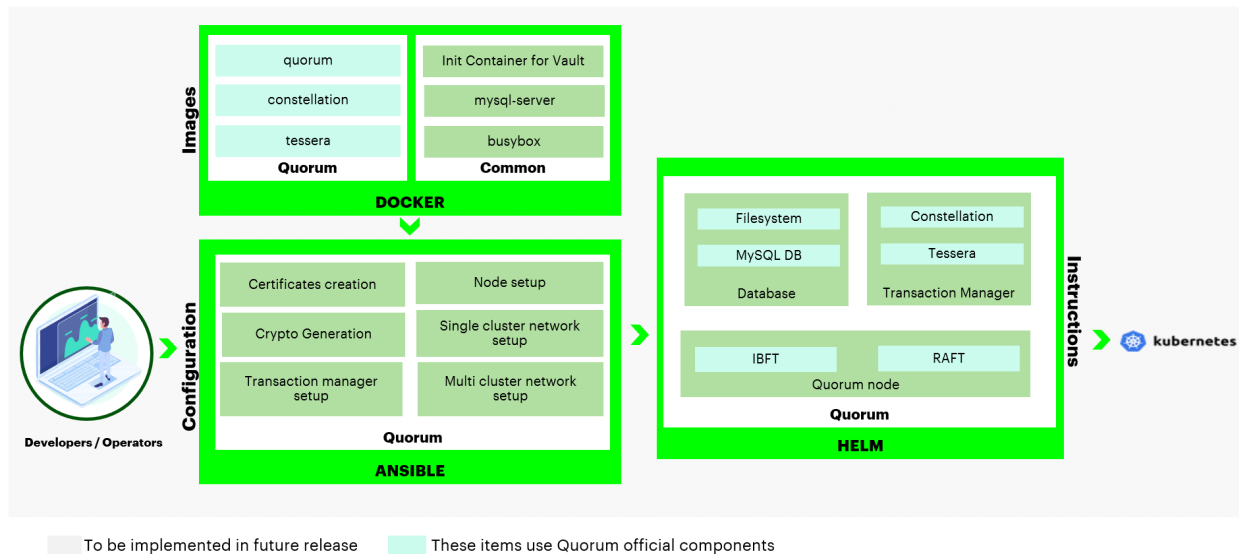


Figure:

### Quorum Components

## Docker Images

The Blockchain Automation Framework uses the officially published Quorum Docker images from [hub.docker.com](https://hub.docker.com). The following Quorum Images are used by the Blockchain Automation Framework.

- **quorum** - Quorum Peer Node
- **tessera** - Tessera Transaction Manager
- **constellation** - Constellation Transaction Manager

Additionally, following common images are also used:

- **busybox** - Used for DB initialization
- **mysql-server** - Used as the DB for Tessera Transaction Manager
- **alpine-utils** - Used as a utility to get crypto from Hashicorp Vault server

## Ansible Playbooks

Detailed information on ansible playbooks can be referred [here](#) and the execution process can be referred [here](#).

## Helm Charts

Detailed information on helm charts can be referred [here](#).

## Vault Configuration

The Blockchain Automation Framework stores their `crypto` immediately in the Hashicorp Vault secrets engine. The crypto is stored by each organization under path `secret/org_namespace` - it contains node keys, keystore, passwords, TM keys, and CA certificates for proxy connections.

The complete key paths in the Vault can be referred [here](#).

### 9.8.11 Certificate Paths on Vault for Quorum Network

**For IBFT/ RAFT**

**For Tessera/Constellation**

**For Root Certificates**

---

Details of Variables

### 9.8.12 Hyperledger Besu Architecture Reference

**Kubernetes**

**Nodes with Orion Transaction Manager**

The following diagram shows how Besu peer nodes with Orion TM will be deployed on your Kubernetes instance.

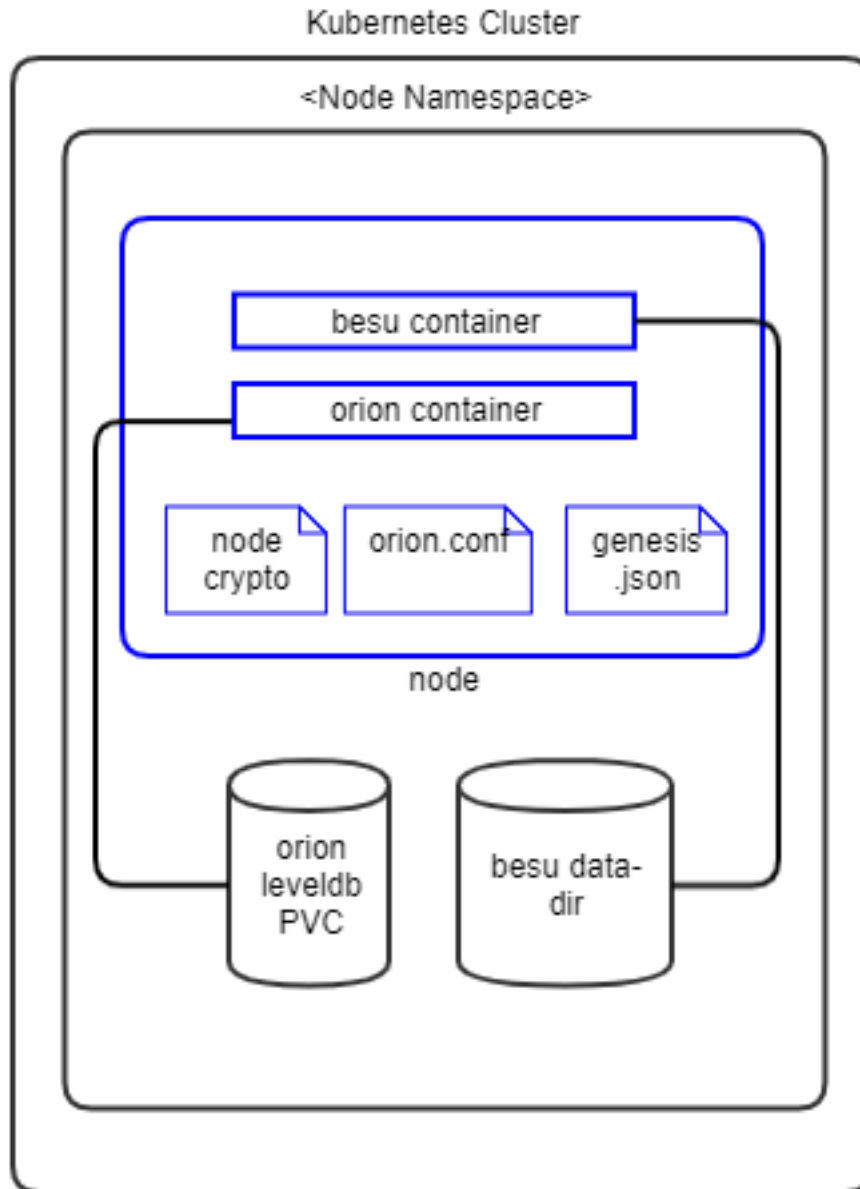


Figure: Hyperledger

## Besu Kubernetes Deployment - Orion Peers

**Notes:**

1. Pods are shown in blue in the diagram.
2. Each peer pod will have two init-containers: `certificates-init` to read crypto from Vault and `liveness-check` to check that if the bootnode endpoint is available, only when bootnode is used.
3. Each peer pod will then have two containers: `orion` and `besu` running. Since they are in the same pod, Kubernetes always schedules them on the same VM and they can communicate to each other through localhost. This guarantees minimal latency between them.
4. The storage uses a Kubernetes Persistent Volume.
5. In future releases, the levelDB PVC will be replaced by a containerised database.

## Validator Nodes

The following diagram shows how Besu Validator nodes will be deployed on your Kubernetes instance.

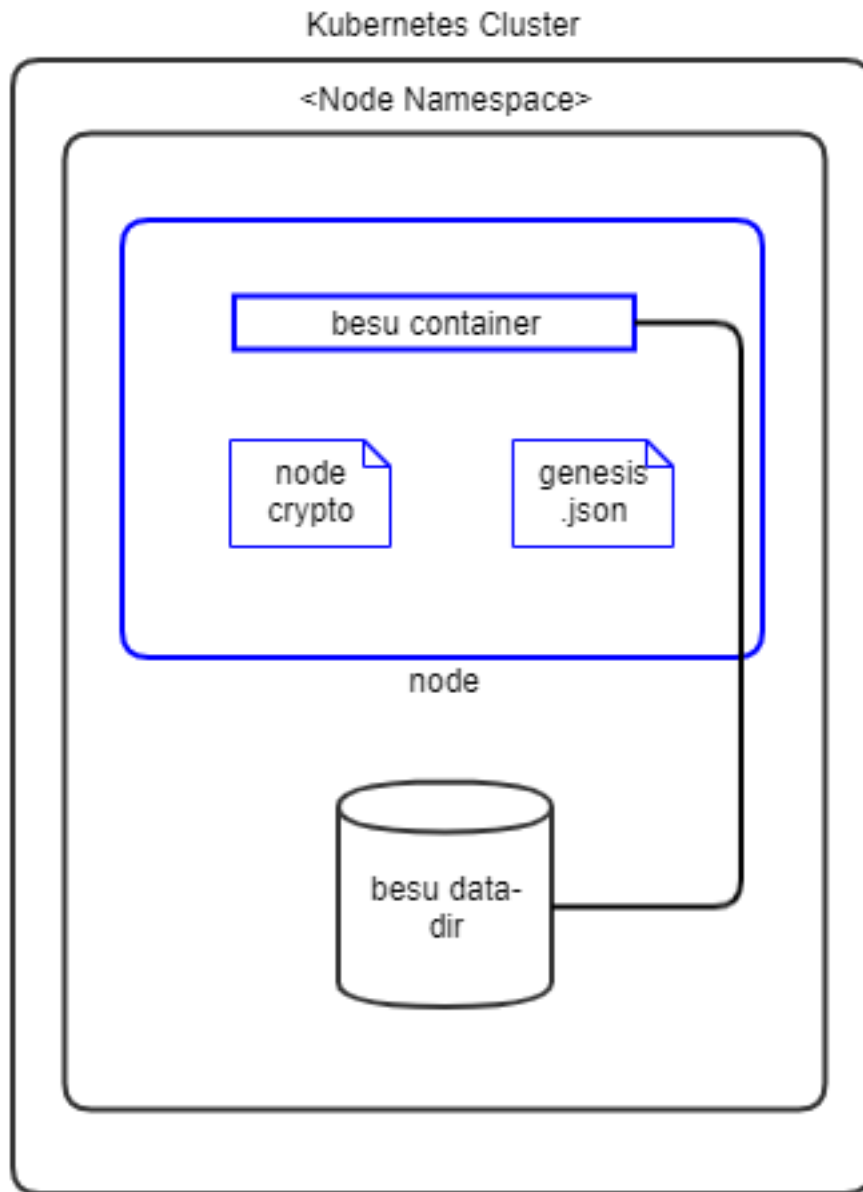


Figure: Hyperledger

Besu Kubernetes Deployment - Validators

### Notes:

1. Pods are shown in blue in the diagram.
2. Each peer pod will have one init-containers: `certificates-init` to read crypto from Vault.
3. Each peer pod will then have one container `besu` running.
4. The storage uses a Kubernetes Persistent Volume for storing the `besu data-dir`.

## Components

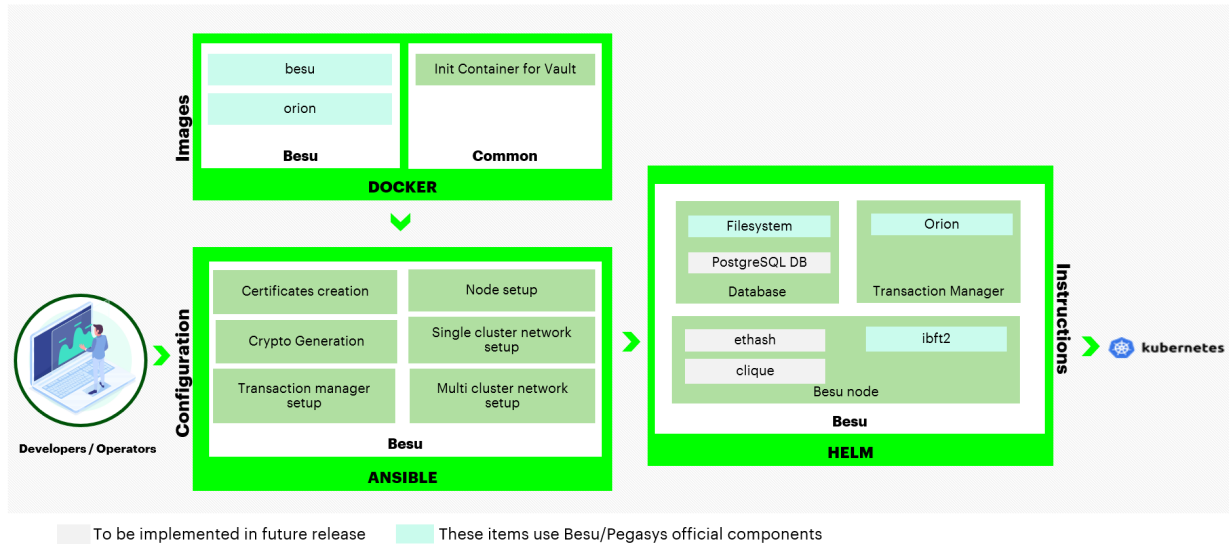


Figure:

Hyperledger Besu Components

## Docker Images

The Blockchain Automation Framework uses the officially published Besu Docker images from [hub.docker.com](https://hub.docker.com). The following Besu Images are used by the Blockchain Automation Framework.

- [besu](#) - Besu Peer and Validator Node
- [orion](#) - Orion Transaction Manager

Additionally, following common images are also used:

- [alpine-utils](#) - Used as a utility to get crypto from Hashicorp Vault server

## Ansible Playbooks

Detailed information on ansible playbooks can be referred [here](#) and the execution process can be referred [here](#).

## Helm Charts

Detailed information on helm charts can be referred [here](#).

## Vault Configuration

The Blockchain Automation Framework stores their `crypto` immediately in the Hashicorp Vault secrets engine. The crypto is stored by each organization under path `secret/org_namespace` - it contains node keys, keystore, passwords, TM keys, and CA certificates for proxy connections.

The complete key paths in the Vault can be referred [here](#).

### 9.8.13 Certificate Paths on Vault for Hyperledger Besu Network

For IBFT2 WIP

For Orion

For Root Certificates

---

Details of Variables



Below are various debugging commands that can be used

### 10.1 Kubectl related debugging

- To setup KUBECONFIG environment variable

```
export KUBECONFIG=PATH_TO_CLUSTER_KUBECONFIG_FILE  
Ex. export KUBECONFIG=~/.kube/config  
/root/.kube/config is the default KUBECONFIG path
```

- To check the cluster config file being used

```
kubectl config view
```

- To check the current context

```
kubectl config current-context
```

- To get all pods in a namespace

```
kubectl get pods -n NAMESPACE  
Ex. kubectl get pods -n supplychain-net
```

- To get all pods in a cluster

```
kubectl get pods --all-namespaces
```

- To check description of resource type (pod/service/pvc/HelmRelease)

```
kubectl describe RESOURCE_TYPE RESOURCE_NAME -n NAMESPACE  
Ex. kubectl describe pvc ca-server-db-svc -n carrier-net  
Ex. kubectl describe sa vault-reviewer -n carrier-net
```

- To check logs of pod

```
kubectl logs POD_NAME -n NAMESPACE  
Ex. kubectl logs flux-dev-123r45 -n default
```

- To check logs of container within a pod

```
kubectl logs POD_NAME -c CONTAINER_NAME -n NAMESPACE  
Ex. kubectl logs ca-123r45 -c ca-certs-init -n carrier-net
```

- To execute a command in a running pod

```
kubectl exec POD_NAME -n NAMESPACE -- COMMAND_TO_EXECUTE  
Ex. kubectl exec ca-tools-12345 -n carrier-net -- ls -a
```

- To execute a command in a container of a pod

```
kubectl exec POD_NAME -c CONTAINER_NAME -n NAMESPACE -- COMMAND_TO_EXECUTE  
Ex. kubectl exec ca-tools-12345 -c ca-tools -n carrier-net -- ls -a
```

## 10.2 Vault related debugging

- To access vault

```
export VAULT_ADDR=  
export VAULT_TOKEN=  
vault read PATH_IN_VAULT  
Ex. vault read /secret/crypto/ordererOrganizations/carrier-net/ca/carrier-net-CA.  
↪key
```

- To list all enabled secrets engines with detailed output

```
vault secrets list -detailed
```

- To enable an auth method at a given path

```
vault auth enable -path PATH  
Ex. vault auth enable -path authpath
```

- To delete data on a given path in the key/value secrets engine

```
vault kv delete PATH  
Ex. vault kv delete secret/creds
```

## 10.3 Helm related debugging

- To list down all helm releases

```
helm ls
```

- To upgrade helm client and Tiller server to the same version

```
helm init --upgrade
```

- To delete an existing helm installation

```
helm del --purge HELM_RELEASE_NAME
Ex. helm del --purge carrier-ca
```

## 10.4 Docker related debugging

- To login to docker registry

```
docker login --username USERNAME --password PASSWORD URL
Ex. docker login --username abcd --password abcd index.docker.io/hyperledgerlabs
```

- To pull images from docker registry

```
docker pull IMAGE_NAME:TAG
Ex. docker pull alpineutils:1.0
```

- To push images to docker registry

```
docker push IMAGE_NAME:TAG
Ex. docker push alpineutilstest:1.0
```

- To build an image from Dockerfile

```
cd FOLDER_TO_DOCKERFILE
docker build -t IMAGE_NAME:TAG -f DOCKERFILE_PATH PATH_TO_BUILD_CONTEXT
Ex. docker build -t alpineutilstest:1.0 -f Dockerfile .
```

## 10.5 Quorum related debugging

To login to a quorum node

```
kubectl exec -it POD_NAME -n POD_NAMESPACE -c quorum -- geth attach "http://
↪localhost:RPC_PORT"
Ex. kubectl exec -it carrier-0 -n carrier-ns -c quorum -- geth attach "http://
↪localhost:8546"
```

Get all the participants present in the network after logging into the node (for raft consensus based cluster)

```
raft.cluster
```

Get node information (after logging into the node)

```
admin.nodeInfo
```

Get the peers attached to the current node (after login into the node)

```
admin.peers
```

Get the account details (after logging into the node)

```
eth.accounts
```

Get retrieves the list of authorized validators at the specified block (for ibft consensus based cluster)

```
istanbul.getValidators(blockHashOrBlockNumber)
```

## 10.6 Indy related debugging

To access indy cli, in any terminal

```
indy-cli
```

To create a pool

```
pool create local-pool gen_txn_file=<path of the genesis file>
```

To connect the pool

```
pool connect <pool name>
```

To create a wallet

```
wallet create <wallet name> <key>
```

To open a wallet

```
wallet open <wallet name> <key>
```

To list the wallets

```
wallet list
```

To delete a wallet

```
wallet delete <wallet name>
```

To create a new did

```
did import <did file>
```

```
did new
```

To create a pool

```
pool create <pool name> gen_txn_file=<pool_genesis_path>
```

To open a pool

```
pool connect <pool name>
```

To list the pool

```
pool list
```

To execute a transaction on ledger

```
ledger nym did=<did name> verkey=<key detail> role=<role name>
```

To get the transaction details

```
ledger get-nym did=<did name>
```



---

Frequently Asked Questions

---

## 11.1 1.FAQs for Getting Started

### 11.1.1 Who are the target users?

In this project, it is assumed that a user would fall into either a category of *Operators* or *Developers*. However, this is not saying that technicians such as Solution/Tech Archs who have more expertise in wider areas are not eligible users, e.g. Blockchain or Distributed Ledger Technology (DLT). On the contrary, a user who has proper technical knowledge on those areas will find the usage of the Blockchain Automation Framework (BAF) repository mentioned in the tutorial on this website to be more straightforward. For people new to these areas, they might find a deep learning curve before using or even contributing back to this repository. If a user is from a non-tech background, but would still like to find out how BAF could accelerate set-up of a new production-scale DLT network, the *Introduction* section is the right start point.

(1) **Operators:** An operator is a System Operator that would work as a Deployment Manager, who has strong technical knowledge on cloud architecture and DevOps but basic DLT. An operator might be a decision maker in a new DLT/Blockchain project, and would be responsible for the ongoing stability of the organization's resources and services as well as set-up and maintenance of one or more applications for the organization.

A **common scenario** that an operator would like to leverage the Blockchain Automation Framework repository might be that s/he has been asked to use a DLT/Blockchain technology for a business case, but s/he does not know where/how to start. S/he might have limited budget, and might not have all the technical skills in the team and was overwhelmed by the time it would take for the solution to be created.

**Unique values** in scenarios like this provisioned by the Blockchain Automation Framework repository are: (a) efficiency and rapid deployment (b) consistent quality (c) open-source (d) cloud infrastructure independence (e) optimization via scalability, modularity and security and (f) accelerated go-to-market.

Essentially, an operator would be able to set up a large-size DLT/Blockchain network in a production environment by using this repository as per the tutorials in this website along with the instructions in the readme files in the repository. The network requirements such as which DLT/Blockchain platform (e.g. Fabric/Corda) and which cloud platform (e.g. AWS/GCP/Azure etc) would be used should have been pre-determined already before using this repository. The operator would ensure that the Blockchain Automation Framework repo is set up and deployed properly. Eventually,

BAF would speed up the whole DLT/Blockchain network set-up process and would require less DLT/Blockchain developers enabling the operator to retain the budgets and man-power for other activities.

(2) **Developers:** A developer can be a DevOps or Full Stack Developer who would have knowledge on multiple programming languages, basic knowledge of DLT/Blockchain networks and smart contracts, Ansible and DevOps. Daily work might include developing applications and using DevOps tools.

A **common scenario** that a developer would like to use this repo might be that s/he would like to gain knowledge on production-scale DLT/Blockchain development, but might not have enough technical skills and experiences yet. Learning knowledge from the existing poorly-designed architecture would be time-consuming and fruitless.

The Blockchain Automation Framework provisions its **unique values** to the developer that s/he now has an opportunity to learn how different sets of cutting-edge technologies leveraged in this repository are combined in use such as reusable architecture patterns, reusable assets including APIs or microservices design. The architecture design in this repository has been fully tested and demonstrated as a high-quality one known for a fact that it has been being improved continuously through the technical experts' rich experiences. The developer could try to use this repository to set up a small-size DLT/Blockchain network to see how it works and gradually pick up new skills across Blockchain, DevOps etc.

Furthermore, the developer could even show the maturity of skills to contribute back to this project. Contributions can include but not limited to (1) suggest or add new functionalities (2) fix various bugs and (3) organize hackthon or developer events for the Blockchain Automation Framework in the future.

### 11.1.2 What is the Blockchain Automation Framework and how could it help me?

In simple words, the Blockchain Automation Framework works as an accelerator to help organizations set up a production-scale DLT network (currently supports Corda, Fabric, Indy, Besu and Quorum) with a single network.yaml file used for *Fabric* or *Corda* or *Quorum* to be configured in this project. It can work in managed Kubernetes Clusters which has been fully tested in AWS Elastic Kubernetes Services (EKS), and should also work in a non-managed Kubernetes Cluster in theory. For detailed information, please see the [Welcome page](#).

### 11.1.3 How do I find more about the Blockchain Automation Framework?

Normally, when a user sees information in this section, it means that s/he has already known the existence of the Blockchain Automation Framework project, at least this readthedocs website. Basically, this website provisions a high-level background information of how to use the Blockchain Automation Framework GitHub repository. For detailed step-by-step instructions, one should go to the Blockchain Automation Framework's GitHub repository and find the readme files for a further reading. Upon finishing reading the tutorials in this website, one should be able to analyse whether the Blockchain Automation Framework would be the right solution in your case and reach a decision to use it or not.

### 11.1.4 How much would Blockchain Automation Framework cost? How much would it cost to run Fulcurn on a cloud platform?

As an open source repository, there will be no cost at all to use the Blockchain Automation Framework. However, by running the Blockchain Automation Framework repository on a cloud platform, there might be cost by using a cloud platform and it will depend on which cloud services you are going to use.

### 11.1.5 Who can support me during this process and answer my questions?

One could raise questions in the Github repository and the Blockchain Automation Framework maintainers will give their best supports at early stages. Later on, when the open community matures, one would expect to get support from people in the community as well.

### **11.1.6 Is there any training provided? If so, what kind of training will be included?**

Unfortunately, there are no existing training for using the Blockchain Automation Framework yet, because we are not sure about the potential size of the community and what types of training people would look forward to. However, we do aware that trainings could happen, if there would be a large number of same or similar questions or issues raised by new users, and if we would have a large amount of requests like this in the future.

### **11.1.7 Can I add/remove one or more organisations as DLT nodes in a running DLT/Blockchain network by using the Blockchain Automation Framework?**

Yes, you can add additional nodes to a running DLT/Blockchain network using the Blockchain Automation Framework (BAF). Unfortunately, BAF does not support removing nodes in a running DLT/Blockchain network, but this significant feature is in our future roadmap, and we will add this feature in a future release.

### **11.1.8 Does the Blockchain Automation Framework support multiple versions of Fabric and Corda? What are the minimum versions for Fabric and Corda supported in the Blockchain Automation Framework?**

The Blockchain Automation Framework currently only supports version 1.4.0, 1.4.4 & 2.0.0 for Fabric and version 4.1 and 4.4 for Corda as minimum versions, and will only support future higher versions for Fabric and Corda. Corda Enterprise 4.4 is available as per Release 0.6.0.0. Please check the [latest releases](#) for version upgrades and deprecations.

## **11.2 2.FAQs for Operators Guide**

### **11.2.1 What is the minimal infrastructure set-up required to run the Blockchain Automation Framework?**

To run the Blockchain Automation Framework repository, you need to have a managed/non-managed Kubernetes clusters ready as well as an unsealed Hashicorp Vault service available.

### **11.2.2 What would be the recommended/required cloud service?**

We recommend to use Cloud Services such as Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP), as their managed Kubernetes clusters services are being or will be tested for this repository. We have fully tested this repository in AWS, and testing it on Azure and GCP is in our future roadmap.

### **11.2.3 Do I have to use AWS?**

No, AWS is not mandatory, but is recommended because it is the first cloud platform we have tested on. Theoretically, the Blockchain Automation Framework repository should work in any cloud platforms as long as a Kubernetes Cluster service is provisioned, but there is no 100% guarantee it will work, since there might be unseen/unknown features in these managed Kubernetes clusters environments we are not aware of.

### **11.2.4 Are there any pre-requisites to run the Blockchain Automation Framework?**

Yes, you can find them on this [page](#).

### 11.2.5 How to configure HashiCorp Vault and Kubernetes?

Please see this [page](#) for details.

### 11.2.6 I'm using Windows machine, can I run the Blockchain Automation Framework on it?

The Blockchain Automation Framework repository relies a lot on using [Ansible](#), which might not work in Windows machines. Please check Ansible website for more information.

### 11.2.7 How do I configure a DLT/Blockchain network?

The network.yaml file is the main file to be configured to set up a DLT/Blockchain network. [This page](#) gives the links for a user to pick up knowledge of how to configure this file for Fabric and Corda first (see the two “Configuration file specification” sections for each DLT/Blockchain platform). Having this knowledge will then enable a user to understand how to use this file in the “Setting up DLT network” section.

### 11.2.8 How can I test whether my DLT/Blockchain network are configured and deployed correctly?

Please see this [page](#) for details.

### 11.2.9 How/Where can I request for new features, bugs and get feedback?

One could request a new feature on the Github repository for now. In the future, people might use Jira or Slack to do the same as well.

### 11.2.10 Are CI/CD pipeline tools a mandatory to use the Blockchain Automation Framework?

No, CI/CD pipeline tools like Jenkins are not mandatory, but it could help a user automate the set-up or testing of a new DLT/Blockchain network in different environments, once a user has a good understanding of using it. We have the master Jenkinsfile in `automation` folder which can be taken as a template.

### 11.2.11 Is it required to run Ansible in a particular machine like AWS EC2?

No, a user should be able to run the Ansible command on any machine as long as Ansible command CLI is installed.

### 11.2.12 Is there an example ansible\_hosts file?

Yes, you can find an example ansible\_hosts file [here](#). The configuration in this file means that all Ansible commands will be run in the same machine that works as both an Ansible client and server machine.

### 11.2.13 Can I specify the tools versions such as kubectl, helm in this project?

Yes, you can specify tools versions like kubectl, helm, HashiCorp Vault, AWS-authenticator in the `playbook environment-setup.yaml`.

### 11.2.14 How would system react if we plan to update tools versions (e.g. kubectl, helm)?

Honestly speaking, we don't know. The latest version Blockchain Automation Framework has been tested on specific client versions of these tools, see below: (1) Kubectl: v1.14.2 for Kubernetes 1.14, v1.16.13 for Kubernetes 1.16 (2) Helm: v2.14.1 for Kubernetes 1.14, v2.16.10 for Kubernetes 1.16 (3) HashiCorp Vault: v1.0.1 (4) AWS-Authenticator: v1.10.3

It is assumed that newer versions of these tools would be backward compatible, which is beyond our control. One can raise a new ticket to the Blockchain Automation Framework GitHub repository, if any major updates would break the system down.

### 11.2.15 Why does the Flux K8s pod get a permission denied for this Blockchain Automation Framework GitHub repository?

This usually means that the private key that you have used in your network.yaml for gitops does not have access to the GitHub repository. The corresponding public key must be added to your GitHub Account (or other git repository that you are using). Details can be found [here](#).

### 11.2.16 Why does the flux-helm-operator keep on reporting “Failed to list \*v1beta1.HelmRelease: the server could not find the requested resource (get helmreleases.flux.weave.works)”?

The HelmRelease CustomResourceDefinition (CRD) was missing from the cluster, according to <https://github.com/fluxcd/flux>, the following command has to be used to deploy it:

```
kubectl apply -f https://raw.githubusercontent.com/fluxcd/flux/helm-0.10.1/deploy-  
helm/flux-helm-release-crd.yaml
```

## 11.3 3.FAQs for Developer Guide

### 11.3.1 How do I contribute to this project?

- Guide on BAF [contribution](#)
- Details on creating pull request on github can be found in this [link](#).

### 11.3.2 Where can I find the Blockchain Automation Framework's coding standards?

TBD

### 11.3.3 How can I engage in the Blockchain Automation Framework community for any events?

Connect us on [Rocket Chat](#)



### 12.1 General

This section lists the general terms that are used in the Blockchain Automation Framework.

#### 12.1.1 Ansible

Ansible is an open-source software provisioning, configuration management, and application-deployment tool. It runs on many Unix-like systems, and can configure both Unix-like systems as well as Microsoft Windows. It includes its own declarative language to describe system configuration. For more details, refer: [Ansible](#)

#### 12.1.2 AWS

Amazon Web Services is a subsidiary of Amazon that provides on-demand cloud computing platforms to individuals, companies, and governments, on a metered pay-as-you-go basis. For more details, refer: [AWS](#)

#### 12.1.3 AWS EKS

Amazon Elastic Container Service for Kubernetes (Amazon EKS) is a managed service that makes it easy for users to run Kubernetes on AWS without needing to stand up or maintain your own Kubernetes control plane. Since Amazon EKS is a managed service it handles tasks such as provisioning, upgrades, and patching. For more details, refer: [EKS](#)

#### 12.1.4 Blockchain as a Service (BaaS)

Blockchain-as-a-Service platform is a full-service cloud-based solution that enables developers, entrepreneurs, and enterprises to develop, test, and deploy blockchain applications and smart contracts that will be hosted on the BaaS platform.

### 12.1.5 Charts

Helm uses a packaging format called charts. A chart is a collection of files that describe a related set of Kubernetes resources. A single chart might be used to deploy something simple, like a memcached pod, or something complex, like a full web app stack with HTTP servers, databases, caches, and so on. For more details, refer: [Helm Charts](#)

### 12.1.6 CI/CD

CI and CD are two acronyms that are often mentioned when people talk about modern development practices. CI is straightforward and stands for continuous integration, a practice that focuses on making preparing a release easier. But CD can either mean continuous delivery or continuous deployment, and while those two practices have a lot in common, they also have a significant difference that can have critical consequences for a business.

### 12.1.7 CLI

A command-line interface (CLI) is a means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines).

### 12.1.8 Cluster

In Kubernetes, a cluster consists of at least one cluster master and multiple worker machines called nodes. For more details, refer: [Cluster](#)

### 12.1.9 Deployment

Software deployment is all of the activities that make a software system available for use. The general deployment process consists of several interrelated activities with possible transitions between them. These activities can occur at the producer side or at the consumer side or both.

### 12.1.10 DLT

Distributed Ledger Technology (DLT) is a digital system for recording the transaction of assets in which the transactions and their details are recorded in multiple places at the same time. Unlike traditional databases, distributed ledgers have no central data store or administration functionality. For more details, refer: [DLT](#)

### 12.1.11 Docker

Docker is a set of platform-as-a-service products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. For more details, refer: [Docker](#)

### 12.1.12 Flux

Flux is the operator that makes GitOps happen in a cluster. It ensures that the cluster config matches the one in git and automates your deployments. Flux enables continuous delivery of container images, using version control for each step to ensure deployment is reproducible, auditable and revertible. Deploy code as fast as your team creates it, confident that you can easily revert if required. For more details, refer: [Flux](#)

### 12.1.13 Git

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows For more details, refer: [GIT](#)

### 12.1.14 Gitops

GitOps is a method used for Continuous Delivery. It uses Git as a single source of truth for infrastructures like declarative infrastructure and the applications. For more details, refer: [Gitops](#)

### 12.1.15 HashiCorp Vault

HashiCorp Vault is a tool for securely accessing secrets. A secret is anything that you want to tightly control access to, such as API keys, passwords, or certificates. Vault provides a unified interface to any secret, while providing tight access control and recording a detailed audit log. For more details, refer: [Vault](#)

### 12.1.16 HashiCorp Vault Client

A Vault client is any stand-alone application or integrated add-in that connects to the vault server to access files and perform vault operations.

### 12.1.17 Helm

Helm is the first application package manager running atop Kubernetes. It allows describing the application structure through convenient helm-charts and managing it with simple commands. For more details, refer: [Helm](#)

### 12.1.18 Hosts

A Host is either a physical or virtual machine.

### 12.1.19 IAM user

An AWS Identity and Access Management (IAM) user is an entity that you create in AWS to represent the person or application that uses it to interact with AWS. A user in AWS consists of a name and credentials. For more details, refer: [IAM Users](#)

### 12.1.20 IOT

The Internet of Things is simply “A network of Internet connected objects able to collect and exchange data.” It is commonly abbreviated as IoT. In a simple way to put it, You have “things” that sense and collect data and send it to the internet. For more details, refer: [IOT](#)

### 12.1.21 Instance

A “cloud instance” refers to a virtual server instance from a public or private cloud network. In cloud instance computing, single hardware is implemented into software and run on top of multiple computers.

### 12.1.22 Jenkins

Jenkins is a free and open source automation server written in Java. Jenkins helps to automate the non-human part of the software development process, with continuous integration and facilitating technical aspects of continuous delivery. For more details, refer: [Jenkins](#)

### 12.1.23 Jenkins Master

Your main Jenkins server is the master machine called Jenkins Master. For more details, refer: [Jenkins Master](#)

### 12.1.24 Jenkins Slave

A slave is a Java executable that runs on a remote machine. For more details, refer: [Jenkins Slave](#)

### 12.1.25 Jenkins Stages

A stage block in Jenkins defines a conceptually distinct subset of tasks performed through the entire Pipeline (e.g. “Build”, “Test” and “Deploy” stages), which is used by many plugins to visualize or present Jenkins Pipeline status/progress.

### 12.1.26 Kubeconfig File

A kubeconfig file is a file used to configure access to Kubernetes when used in conjunction with the kubectl command line tool (or other clients). This is usually refer to an environment variable called KUBECONFIG.

### 12.1.27 Kubernetes

Kubernetes (K8s) is an open-source container-orchestration system for automating application deployment, scaling, and management. It was originally designed by Google, and is now maintained by the Cloud Native Computing Foundation. For more details, refer: [Kubernetes](#)

### 12.1.28 Kubernetes Node

A node is a worker machine in Kubernetes, previously known as a minion. A node may be a VM or physical machine, depending on the cluster. Each node contains the services necessary to run pods and is managed by the master components. The services on a node include the container runtime, kubelet and kube-proxy. For more details, refer: [Kubernetes Node](#)

### 12.1.29 Kubernetes Storage Class

A StorageClass in Kubernetes provides a way for administrators to describe the “classes” of storage they offer. Different classes might map to quality-of-service levels, or to backup policies, or to arbitrary policies determined by the cluster administrators. For more details, refer: [Storage class](#)

### 12.1.30 Kubernetes PersistentVolume (PV)

A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual pod that uses the PV. For more details, refer: [PVC](#)

### 12.1.31 Kubernetes Persistent Volume Claim (PVC)

A PVC, binds a persistent volume to a pod that requested it. When a pod wants access to a persistent disk, it will request access to the claim which will specify the size, access mode and/or storage classes that it will need from a Persistent Volume. For more details, refer: [PVC](#)

### 12.1.32 PGP signature

Pretty Good Privacy (PGP) is an encryption program that provides cryptographic privacy and authentication for data communication. PGP is used for signing, encrypting, and decrypting texts, e-mails, files, directories, and whole disk partitions. For more details, refer: [PGP](#)

### 12.1.33 Playbook

An Ansible playbook is an organized unit of scripts that defines work for a server configuration managed by the automation tool Ansible. For more details, refer: [Playbooks](#)

### 12.1.34 Pipeline

Jenkins Pipeline (or simply “Pipeline”) is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins. A continuous delivery pipeline is an automated expression of your process for getting software from version control right through to your users and customers. For more details, refer: [Pipeline](#)

### 12.1.35 Roles

Roles provide a framework for fully independent, or interdependent collections of variables, tasks, files, templates, and modules. In Ansible, the role is the primary mechanism for breaking a playbook into multiple files. This simplifies writing complex playbooks, and it makes them easier to reuse. For more details, refer: [Roles](#)

### 12.1.36 SCM

Supply Chain Management (SCM) is the broad range of activities required to plan, control and execute a product’s flow, from acquiring raw materials and production through distribution to the final customer, in the most streamlined and cost-effective way possible.

### 12.1.37 SHA256

SHA-256 stands for Secure Hash Algorithm – 256 bit and is a type of hash function commonly used in Blockchain. A hash function is a type of mathematical function which turns data into a fingerprint of that data called a hash. It’s like a formula or algorithm which takes the input data and turns it into an output of a fixed length, which represents the fingerprint of the data. For more details, refer: [SHA256](#)

### 12.1.38 Sphinx

Sphinx is a tool that makes it easy to create intelligent and beautiful documentation, written by Georg Brandl and licensed under the BSD license. It was originally created for the Python documentation, and it has excellent facilities for the documentation of software projects in a range of languages. For more details, refer: [Sphinx](#)

### 12.1.39 SSH

SSH, also known as Secure Shell or Secure Socket Shell, is a network protocol that gives users, particularly system administrators, a secure way to access a computer over an unsecured network. SSH also refers to the suite of utilities that implement the SSH protocol. For more details, refer: [SSH](#)

### 12.1.40 Template

- Ansible: A template in Ansible is a file which contains all your configuration parameters, but the dynamic values are given as variables. During the playbook execution, depending on the conditions like which cluster you are using, the variables will be replaced with the relevant values. For more details, refer: [Ansible Template](#)
- Helm Charts: In Helm Charts, Templates generate manifest files, which are YAML-formatted resource descriptions that Kubernetes can understand. For more details, refer: [Helm Charts Template](#)

### 12.1.41 Tiller

Tiller is the in-cluster component of Helm. It interacts directly with the Kubernetes API server to install, upgrade, query, and remove Kubernetes resources. It also stores the objects that represent releases. For more details, refer: [Tiller](#)

### 12.1.42 TLS

Transport Layer Security, and its now-deprecated predecessor, Secure Sockets Layer, are cryptographic protocols designed to provide communications security over a computer network. For more details, refer: [TLS](#)

### 12.1.43 YAML

YAML (“YAML Ain’t Markup Language”) is a human-readable data-serialization language. It is commonly used for configuration files and in applications where data is being stored or transmitted. YAML targets many of the same communications applications as Extensible Markup Language but has a minimal syntax which intentionally differs from SGML. For more details, refer: [YAML](#)

---

## 12.2 Hyperledger-Fabric

This section lists specific terms used in Hyperledger Fabric

### 12.2.1 CA

The Hyperledger Fabric CA is a Certificate Authority (CA) for Hyperledger Fabric. It provides features such as: registration of identities, or connects to LDAP as the user registry. For more details, refer: [CA](#)

### 12.2.2 CA Server

Fabric CA server is used to host one or more Certification Authorities (Fabric CA) for your Fabric Network (based on the MSPs)

### 12.2.3 Chaincode

Chaincode is a piece of code that is written in one of the supported languages such as Go or Java. It is installed and instantiated through an SDK or CLI onto a network of Hyperledger Fabric peer nodes, enabling interaction with that network's shared ledger. For more details, refer: [Chaincode](#)

### 12.2.4 Channel

A Hyperledger Fabric channel is a private “subnet” of communication between two or more specific network members, for the purpose of conducting private and confidential transactions. A channel is defined by members (organizations), anchor peers per member, the shared ledger, chaincode application(s) and the ordering service node(s). For more details, refer: [Channel](#)

### 12.2.5 Channel Artifacts

Artifacts in Hyperledger are channel configuration files which are required for the Hyperledger Fabric network. They are generated at the time of network creation. For more details, refer: [Channel Artifacts](#)

### 12.2.6 Instantiate

Instantiating a chaincode means to initialize it with initial values. For more details, refer: [Instantiating Chaincode](#)

### 12.2.7 MSP

Hyperledger Fabric includes a Membership Service Provider (MSP) component to offer an abstraction of all cryptographic mechanisms and protocols behind issuing and validating certificates, and user authentication. For more details, refer: [MSP](#)

### 12.2.8 Orderer

Orderer peer is considered as the central communication channel for the Hyperledger Fabric network. Orderer peer/node is responsible for consistent Ledger state across the network. Orderer peer creates the block and delivers that to all the peers For more details, refer: [Orderer](#)

### 12.2.9 Peer

Hyperledger Fabric is a permissioned blockchain network that gets set by the organizations that intend to set up a consortium. The organizations that take part in building the Hyperledger Fabric network are called the “members”. Each member organization in the blockchain network is responsible to set up their peers for participating in the network. All of these peers need are configured with appropriate cryptographic materials like Certificate Authority and other information. For more details, refer: [Peer](#)

### 12.2.10 Zkafka

Kafka is primarily a distributed, horizontally-scalable, fault-tolerant, commit log. A commit log is basically a data structure that only appends. No modification or deletion is possible, which leads to no read/write locks, and the worst case complexity  $O(1)$ . There can be multiple Kafka nodes in the blockchain network, with their corresponding Zookeeper ensemble. For more details, refer: [zkafka](#)

### 12.2.11 RAFT

RAFT is distributed crash Fault tolerance consensus algorithm which makes sure that in the event of failure, the system should be able to take a decision and process clients request. In technical term Raft is a consensus algorithm for managing a replicated log. Replicated log is a part of Replicated state machine. For more details, refer: [raft](#)

---

## 12.3 R3 Corda

This section lists specific terms used in R3 Corda.

### 12.3.1 Compatibility Zone

Every Corda node is part of a “zone” (also sometimes called a Corda network) that is permissioned. Production deployments require a secure certificate authority. We use the term “zone” to refer to a set of technically compatible nodes reachable over a TCP/IP network like the internet. The word “network” is used in Corda but can be ambiguous with the concept of a “business network”, which is usually more like a membership list or subset of nodes in a zone that have agreed to trade with each other. For more details, refer [Compatibility Zone](#).

### 12.3.2 CorDapp

CorDapps (Corda Distributed Applications) are distributed applications that run on the Corda platform. The goal of a CorDapp is to allow nodes to reach agreement on updates to the ledger. They achieve this goal by defining flows that Corda node owners can invoke over RPC. For more details, refer: [CorDapp](#)

### 12.3.3 Corda Node

A Corda node is a JVM run-time environment with a unique identity on the network that hosts Corda services and CorDapps. For more details, refer [Corda Node](#).

### 12.3.4 Corda Web Server

A simple web server is provided that embeds the Jetty servlet container. The Corda web server is not meant to be used for real, production-quality web apps. Instead it shows one example way of using Corda RPC in web apps to provide a REST API on top of the Corda native RPC mechanism.

### 12.3.5 Doorman

The Doorman CA is a Certificate Authority R3 Corda. It is used for day-to-day key signing to reduce the risk of the root network CA's private key being compromised. This is equivalent to an intermediate certificate in the web PKI. For more details, refer [Doorman](#).

### 12.3.6 NetworkMap

The Network Map Service accepts digitally signed documents describing network routing and identifying information from nodes, based on the participation certificates signed by the Identity Service, and makes this information available to all Corda Network nodes. For more details, refer [Networkmap](#).

### 12.3.7 Notary

The Corda design separates correctness consensus from uniqueness consensus, and the latter is provided by one or more Notary Services. The Notary will digitally sign a transaction presented to it, provided no transaction referring to any of the same inputs has been previously signed by the Notary, and the transaction timestamp is within bounds. Business network operators and network participants may choose to enter into legal agreements which rely on the presence of such digital signatures when determining whether a transaction to which they are party, or upon the details of which they otherwise rely, is to be treated as 'confirmed' in accordance with the terms of the underlying agreement. For more details, refer [Corda Notaries](#).

---

## 12.4 Hyperledger-Indy

This section lists specific terms used in Hyperledger-Indy.

### 12.4.1 Admin DID

A decentralized identifier for Admin as defined by the DID Data Model and Generic Syntax specification.

### 12.4.2 Admin Seed

Seed can be any randomly chosen 32 byte value. There is no predefined format for the seed and it used to initializing keys. The seed used for Admin key is called an admin seed.

### 12.4.3 Agency

A service provider that hosts Cloud Agents and may provision Edge Agents on behalf of a Ledger's Entities.

### 12.4.4 Agent

A software program or process used by or acting on behalf of a Ledger's Entity to interact with other Agents or, via a Ledger's Client component, directly with the Ledger. Agents are of two types: Edge Agents run at the edge of the network on a local device, while Cloud Agents run remotely on a server or cloud hosting service. Agents typically have access to a Wallet in order to perform cryptographic operations on behalf of the Ledger's Entity they represent.

### 12.4.5 Dependent

An Individual who needs to depend on a Guardian to administer the Individual's Ledger Identities. Under a Trust Framework, all Dependents may have the right to become Independents. Mutually exclusive with Independent.

### 12.4.6 Developer

An Identity Owner that has legal accountability (in a scenario where there is a Trust Framework) for the functionality of an Agent, or for software that interacts with an Agent or the Ledger, to provide services to a Ledger Entity.

### 12.4.7 DID

A decentralized identifier as defined by the DID Data Model and Generic Syntax specification. DIDs enable interoperable decentralized self-sovereign identity management. An Identity Record is associated with exactly one DID. A DID is associated with exactly one DDO.

### 12.4.8 Domain Genesis

Domain genesis is a genesis file used to initialise the network and may populate network with some domain data.

### 12.4.9 Endorser

Endorser has the required rights to write on a ledger. Endorser submits a transaction on behalf of the original author.

### 12.4.10 Genesis Record

The first Identity Record written to the Ledger that describes a new Ledger Entity. For a Steward, the Genesis Record must be written by a Trustee. For an Independent Identity Owner, the Genesis Record must be written by a Trust Anchor. For a Dependent Identity Owner, the Genesis Record must be written by a Guardian.

### 12.4.11 Identity

A set of Identity Records, Claims, and Proofs that describes a Ledger Entity. To protect privacy: a) an Identity Owner may have more than one Ledger Identity, and b) only the Identity Owner and the Relying Party(s) with whom an Identity is shared knows the specific set of Identity Records, Claims, and Proofs that comprise that particular Identity.

### 12.4.12 Identity Owner

A Ledger Entity who can be held legally accountable. An Identity Owner must be either an Individual or an Organization. Identity owners can also be distinguished as Independent Identity Owner and Dependent Identity Owner based on the writer of the Genesis record, for an Independent Identity Owner the Genesis Record must be written by a Trust Anchor and in case of a Dependent Identity Owner the the Genesis Record must be written by a Guardian.

### **12.4.13 Identity Record**

A transaction on the Ledger that describes a Ledger Entity. Every Identity Record is associated with exactly one DID. The registration of a DID is itself an Identity Record. Identity Records may include Public Keys, Service Endpoints, Claim Definitions, Public Claims, and Proofs. Identity Records are Public Data.

### **12.4.14 Identity Role**

Each identity has a specific role in Indy described by one of four roles in Indy. These roles are Trustee, Steward, Endorser and Network Monitor.

### **12.4.15 Issuer Key**

The special type of cryptographic key necessary for an Issuer to issue a Claim that supports Zero Knowledge Proofs.

### **12.4.16 Ledger**

The ledger in Indy is Indy-plenum based. Provides a simple, python-based, immutable, ordered log of transactions backed by a merkle tree. For more details, refer [Indy-plenum](#)

### **12.4.17 NYM Transaction**

NYM record is created for a specific user, Trust Anchor, Sovrin Stewards or trustee. The transaction can be used for creation of new DIDs, setting and Key Rotation of verification key, setting and changing of roles.

### **12.4.18 Pairwise-Unique Identifier**

A Pseudonym used in the context of only one digital relationship (Connection). See also Pseudonym and Verinym.

### **12.4.19 Pool Genesis**

Pool genesis is a genesis file used to initialise the network and may populate network with some pool data.

### **12.4.20 Private Claim**

A Claim that is sent by the Issuer to the Holder's Agent to hold (and present to Relying Parties) as Private Data but which can be verified using Public Claims and Public Data. A Private Claim will typically use a Zero Knowledge Proof, however it may also use a Transparent Proof.

### **12.4.21 Private Data**

Data over which an Entity exerts access control. Private Data should not be stored on a Ledger even when encrypted. Mutually exclusive with Public Data.

### 12.4.22 Private Key

The half of a cryptographic key pair designed to be kept as the Private Data of an Identity Owner. In elliptic curve cryptography, a Private Key is called a signing key.

### 12.4.23 Prover

The Entity that issues a Zero Knowledge Proof from a Claim. The Prover is also the Holder of the Claim.

### 12.4.24 Pseudonym

A Blinded Identifier used to maintain privacy in the context on an ongoing digital relationship (Connection).

### 12.4.25 Steward

An Organization, within a Trust Framework, that operate a Node. A Steward must meet the Steward Qualifications and agree to the Steward Obligations defined in the a Trust Framework. All Stewards are automatically Trust Anchors.

### 12.4.26 Trust Anchor

An Identity Owner who may serve as a starting point in a Web of Trust. A Trust Anchor has two unique privileges: 1) to add new Identity Owners to a Network, and 2) to issue Trust Anchor Invitations. A Trust Anchor must meet the Trust Anchor Qualifications and agree to the Trust Anchor Obligations defined in a Trust Framework. All Trustees and Stewards are automatically Trust Anchors.

### 12.4.27 Verinym

A DID authorized to be written to an Indy-powered Ledger by a Trust Anchor so that it is directly or indirectly associated with the Legal Identity of the Identity Owner. Mutually exclusive with Anonym.

### 12.4.28 Wallet

A software module, and optionally an associated hardware module, for securely storing and accessing Private Keys, Master Secrets, and other sensitive cryptographic key material and optionally other Private Data used by an Entity on Indy. A Wallet may be either an Edge Wallet or a Cloud Wallet. In Indy infrastructure, a Wallet implements the emerging DKMS standards for interoperable decentralized cryptographic key management.

### 12.4.29 Zero Knowledge Proof

A Proof that uses special cryptography and a Master Secret to permit selective disclosure of information in a set of Claims. A Zero Knowledge Proof proves that some or all of the data in a set of Claims is true without revealing any additional information, including the identity of the Prover. Mutually exclusive with Transparent Proof.

## 12.5 Quorum

This section lists specific terms used in Quorum.

### 12.5.1 Constellation

Haskell implementation of a general-purpose system for submitting information in a secure way. it is comparable to a network of MTA (Message Transfer Agents) where messages are encrypted with PGP. Contains Node ( Private transaction manager ) and the Enclave.

### 12.5.2 Enode

Enode is a url which identifies a node, it is generated using the node keys.

### 12.5.3 Istanbul Tool

Istanbul tool is istanbul binary compiled from the code repository. The tool is used to generate the configuration files required for setting up the Quorum network with IBFT consensus.

### 12.5.4 Node Keys

Node keys consist of node private and node public keys. Those keys are required by the binaries provided by Quorum to boot the node and the network.

### 12.5.5 Private Transactions

Private Transactions are those Transactions whose payload is only visible to the network participants whose public keys are specified in the privateFor parameter of the Transaction . privateFor can take multiple addresses in a comma separated list.

### 12.5.6 Public Transactions

Public Transactions are those Transactions whose payload is visible to all participants of the same Quorum network. These are created as standard Ethereum Transactions in the usual way.

### 12.5.7 Quorum Node

Quorum Node is designed to be a lightweight fork of geth in order that it can continue to take advantage of the R&D that is taking place within the ever growing Ethereum community. Quorum Node is running geth, a Go-Ethereum client with rpc endpoints. It supports raft and IBFT pluggable consensus and private and permissioned transactions.

### 12.5.8 State

Quorum supports dual state, Public State(accessible by all nodes within the network) and Private State(only accessible by nodes with the correct permissions). The difference is made through the use of transactions with encrypted (private) and non-encrypted payloads (public). Nodes can determine if a transaction is private by looking at the v value of the signature. Public transactions have a v value of 27 or 28, private transactions have a value of 37 or 38.

### 12.5.9 Static nodes

Static nodes are nodes we keep reference to even if the node is not alive. So that when the nodes comes alive, then we can connect to it. Hostnames are permitted here, and are resolved once at startup. If a static peer goes offline and its IP address changes, then it is expected that that peer would re-establish the connection in a fully static network, or have discovery enabled.

### 12.5.10 Tessera

Java implementation of a general-purpose system for submitting information in a secure way. it is comparable to a network of MTA (Message Transfer Agents) where messages are encrypted with PGP. Contains Node ( Private transaction manager ) and The Enclave.

### 12.5.11 The Enclave

Distributed Ledger protocols typically leverage cryptographic techniques for transaction authenticity, participant authentication, and historical data preservation (i.e. through a chain of cryptographically hashed data.) In order to achieve a separation of concerns, as well as to provide performance improvements through parallelization of certain crypto-operations, much of the cryptographic work including symmetric key generation and data encryption/decryption is delegated to the Enclave.

### 12.5.12 Transaction Manager

Quorum's Transaction Manager is responsible for Transaction privacy. It stores and allows access to encrypted transaction data, exchanges encrypted payloads with other participant's Transaction Managers but does not have access to any sensitive private keys. It utilizes the Enclave for cryptographic functionality (although the Enclave can optionally be hosted by the Transaction Manager itself.)

Thank you for your interest to contribute to BAF!

We welcome contributions to the Blockchain Automation Framework Project in many forms, and there's always plenty to do!

First things first, please review the [Hyperledger Code of Conduct](#) before participating and please follow it in all your interactions with the project.

You can contribute to BAF, as a user or/and as a developer.

### 13.1 As a user:

[Making Feature/Enhancement Proposals](#)[Reporting bugs](#)

### 13.2 As a developer:

Consider picking up a “[help-wanted](#)” or “[good-first-issue](#)” task

If you can commit to full-time/part-time development, then please contact us on our [Rocketchat channel](#) to work through logistics!

Please visit the [Developer Guide](#) in the docs to learn how to make contributions to this exciting project.

### 13.3 Pull Request Process :

For source code integrity , Blockchain Automation Framework GitHub pull requests are accepted from forked repositories only. There are also quality standards identified and documented here that will be enhanced over time.

1. Fork BAF via Github UI
2. Clone the fork to your local machine

3. Complete the desired changes and where possible test locally (more detail to come here)
4. Commit your changesi) Make sure you sign your commit using `git commit -s` for more information see [here](#)ii) Make sure your commit message follows [Conventional Commits syntax](#); this aids in release notes generation
5. Push your changes to your feature branch
6. Initiate a pull request from your fork to the base repository ( develop branch , unless it is a critical bug, in that case initiate to the master branch)
7. Await DCO & linting quality checks ( CI to come soon ), as well as any feedback from reviewers.
8. Work on the feedbacks to revise the PR if there are any comments
9. If not, the PR gets approved , delete feature branch post the merge

---

**NOTE:** If you are a regular contributor , please make sure to take the latest pull from the develop branch everytime before making any pull request , master branch in case of a critical defect / bug .

---

This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

---

### Maintainers for the Blockchain Automation Framework

---

This file is the official list of maintainers for the Blockchain Automation Framework project. Changes to this list should be submitted by submitting a pull request that changes this file, and requesting reviews on that pull request from all of the current maintainers. This is the list of maintainers, including their github profiles for direct communications:

#### 14.1 License

The Blockchain Automation Framework source code files are made available under the Apache License, Version 2.0 (Apache-2.0), located in the LICENSE file. The Blockchain Automation Framework documentation files are made available under the Creative Commons Attribution 4.0 International License (CC-BY-4.0), available at <http://creativecommons.org/licenses/by/4.0/>.