链滴

Hyperledger Fabric-1.4.1 核心配置 (configtx.yaml,core.yaml,orderer.yaml)

作者：jockming112

原文链接：https://ld246.com/article/1608532362676

来源网站：链滴

以下配置均来自源码，没有做过修改。这几个文件给出了fabric的完整配置以及各个配置项的相关说
。

configtx.yaml

```
---
################################################################################
#########
#
#   ORGANIZATIONS
#
#   This section defines the organizational identities that can be referenced
#   in the configuration profiles.
#
################################################################################
#########
Organizations:

    # SampleOrg defines an MSP using the sampleconfig. It should never be used
    # in production but may be used as a template for other definitions.
    - &SampleOrg
        # Name is the key by which this org will be referenced in channel
        # configuration transactions.
        # Name can include alphanumeric characters as well as dots and dashes.
        Name: SampleOrg

        # ID is the key by which this org's MSP definition will be referenced.
        # ID can include alphanumeric characters as well as dots and dashes.
        ID: SampleOrg

        # MSPDir is the filesystem path which contains the MSP configuration.
        MSPDir: msp

        # Policies defines the set of policies at this level of the config tree
        # For organization policies, their canonical path is usually
        #   /Channel/<Application|Orderer>/<OrgName>/<PolicyName>
        Policies: &SampleOrgPolicies
            Readers:
                Type: Signature
                Rule: "OR('SampleOrg.member')"
                # If your MSP is configured with the new NodeOUs, you might
                # want to use a more specific rule like the following:
                # Rule: "OR('SampleOrg.admin', 'SampleOrg.peer', 'SampleOrg.client')"
            Writers:
                Type: Signature
                Rule: "OR('SampleOrg.member')"
                # If your MSP is configured with the new NodeOUs, you might
                # want to use a more specific rule like the following:
                # Rule: "OR('SampleOrg.admin', 'SampleOrg.client')"
            Admins:
                Type: Signature
                Rule: "OR('SampleOrg.admin')"
```

```
        # AnchorPeers defines the location of peers which can be used for
        # cross-org gossip communication. Note, this value is only encoded in
        # the genesis block in the Application section context.
        AnchorPeers:
            - Host: 127.0.0.1
              Port: 7051

################################################################################
#########
#
#   CAPABILITIES
#
#   This section defines the capabilities of fabric network. This is a new
#   concept as of v1.1.0 and should not be utilized in mixed networks with
#   v1.0.x peers and orderers.  Capabilities define features which must be
#   present in a fabric binary for that binary to safely participate in the
#   fabric network.  For instance, if a new MSP type is added, newer binaries
#   might recognize and validate the signatures from this type, while older
#   binaries without this support would be unable to validate those
#   transactions.  This could lead to different versions of the fabric binaries
#   having different world states.  Instead, defining a capability for a channel
#   informs those binaries without this capability that they must cease
#   processing transactions until they have been upgraded.  For v1.0.x if any
#   capabilities are defined (including a map with all capabilities turned off)
#   then the v1.0.x peer will deliberately crash.
#
################################################################################
#########
Capabilities:
    # Channel capabilities apply to both the orderers and the peers and must be
    # supported by both.
    # Set the value of the capability to true to require it.
    Channel: &ChannelCapabilities
        # V1.3 for Channel is a catchall flag for behavior which has been
        # determined to be desired for all orderers and peers running at the v1.3.x
        # level, but which would be incompatible with orderers and peers from
        # prior releases.
        # Prior to enabling V1.3 channel capabilities, ensure that all
        # orderers and peers on a channel are at v1.3.0 or later.
        V1_3: true

    # Orderer capabilities apply only to the orderers, and may be safely
    # used with prior release peers.
    # Set the value of the capability to true to require it.
    Orderer: &OrdererCapabilities
        # V1.1 for Orderer is a catchall flag for behavior which has been
        # determined to be desired for all orderers running at the v1.1.x
        # level, but which would be incompatible with orderers from prior releases.
        # Prior to enabling V1.1 orderer capabilities, ensure that all
        # orderers on a channel are at v1.1.0 or later.
        V1_1: true

    # Application capabilities apply only to the peer network, and may be safely
    # used with prior release orderers.
```

```
    # Set the value of the capability to true to require it.
    Application: &ApplicationCapabilities
        # V1.3 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.3.
        V1_3: true
        # V1.2 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.2 (note, this need not be set if
        # later version capabilities are set)
        V1_2: false
        # V1.1 for Application enables the new non-backwards compatible
        # features and fixes of fabric v1.1 (note, this need not be set if
        # later version capabilities are set).
        V1_1: false

################################################################################
#########
#
#   APPLICATION
#
#   This section defines the values to encode into a config transaction or
#   genesis block for application-related parameters.
#
################################################################################
#########
Application: &ApplicationDefaults
    ACLs: &ACLsDefault
        # This section provides defaults for policies for various resources
        # in the system. These "resources" could be functions on system chaincodes
        # (e.g., "GetBlockByNumber" on the "qscc" system chaincode) or other resources
        # (e.g.,who can receive Block events). This section does NOT specify the resource's
        # definition or API, but just the ACL policy for it.
        #
        # User's can override these defaults with their own policy mapping by defining the
        # mapping under ACLs in their channel definition

        #---Lifecycle System Chaincode (lscc) function to policy mapping for access control---#

        # ACL policy for lscc's "getid" function
        lscc/ChaincodeExists: /Channel/Application/Readers

        # ACL policy for lscc's "getdepspec" function
        lscc/GetDeploymentSpec: /Channel/Application/Readers

        # ACL policy for lscc's "getccdata" function
        lscc/GetChaincodeData: /Channel/Application/Readers

        # ACL Policy for lscc's "getchaincodes" function
        lscc/GetInstantiatedChaincodes: /Channel/Application/Readers

        #---Query System Chaincode (qscc) function to policy mapping for access control---#

        # ACL policy for qscc's "GetChainInfo" function
        qscc/GetChainInfo: /Channel/Application/Readers
```

```
        # ACL policy for qscc's "GetBlockByNumber" function
        qscc/GetBlockByNumber: /Channel/Application/Readers

        # ACL policy for qscc's  "GetBlockByHash" function
        qscc/GetBlockByHash: /Channel/Application/Readers

        # ACL policy for qscc's "GetTransactionByID" function
        qscc/GetTransactionByID: /Channel/Application/Readers

        # ACL policy for qscc's "GetBlockByTxID" function
        qscc/GetBlockByTxID: /Channel/Application/Readers

        #---Configuration System Chaincode (cscc) function to policy mapping for access control
--#

        # ACL policy for cscc's "GetConfigBlock" function
        cscc/GetConfigBlock: /Channel/Application/Readers

        # ACL policy for cscc's "GetConfigTree" function
        cscc/GetConfigTree: /Channel/Application/Readers

        # ACL policy for cscc's "SimulateConfigTreeUpdate" function
        cscc/SimulateConfigTreeUpdate: /Channel/Application/Readers

        #---Miscellanesous peer function to policy mapping for access control---#

        # ACL policy for invoking chaincodes on peer
        peer/Propose: /Channel/Application/Writers

        # ACL policy for chaincode to chaincode invocation
        peer/ChaincodeToChaincode: /Channel/Application/Readers

        #---Events resource to policy mapping for access control###---#

        # ACL policy for sending block events
        event/Block: /Channel/Application/Readers

        # ACL policy for sending filtered block events
        event/FilteredBlock: /Channel/Application/Readers

    # Organizations lists the orgs participating on the application side of the
    # network.
    Organizations:

    # Policies defines the set of policies at this level of the config tree
    # For Application policies, their canonical path is
    #   /Channel/Application/<PolicyName>
    Policies: &ApplicationDefaultPolicies
        Readers:
            Type: ImplicitMeta
            Rule: "ANY Readers"
        Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
```

```
        Admins:
            Type: ImplicitMeta
            Rule: "MAJORITY Admins"

    # Capabilities describes the application level capabilities, see the
    # dedicated Capabilities section elsewhere in this file for a full
    # description
    Capabilities:
        <<: *ApplicationCapabilities

################################################################################
#########
#
#   ORDERER
#
#   This section defines the values to encode into a config transaction or
#   genesis block for orderer related parameters.
#
################################################################################
#########
Orderer: &OrdererDefaults

    # Orderer Type: The orderer implementation to start.
    # Available types are "solo" and "kafka".
    OrdererType: solo

    # Addresses here is a nonexhaustive list of orderers the peers and clients can
    # connect to. Adding/removing nodes from this list has no impact on their
    # participation in ordering.
    # NOTE: In the solo case, this should be a one-item list.
    Addresses:
        - 127.0.0.1:7050

    # Batch Timeout: The amount of time to wait before creating a batch.
    BatchTimeout: 2s

    # Batch Size: Controls the number of messages batched into a block.
    # The orderer views messages opaquely, but typically, messages may
    # be considered to be Fabric transactions.  The 'batch' is the group
    # of messages in the 'data' field of the block.  Blocks will be a few kb
    # larger than the batch size, when signatures, hashes, and other metadata
    # is applied.
    BatchSize:

        # Max Message Count: The maximum number of messages to permit in a
        # batch.  No block will contain more than this number of messages.
        MaxMessageCount: 500

        # Absolute Max Bytes: The absolute maximum number of bytes allowed for
        # the serialized messages in a batch. The maximum block size is this value
        # plus the size of the associated metadata (usually a few KB depending
        # upon the size of the signing identities). Any transaction larger than
        # this value will be rejected by ordering. If the "kafka" OrdererType is
        # selected, set 'message.max.bytes' and 'replica.fetch.max.bytes' on
```

```
    # the Kafka brokers to a value that is larger than this one.
    AbsoluteMaxBytes: 10 MB

    # Preferred Max Bytes: The preferred maximum number of bytes allowed
    # for the serialized messages in a batch. Roughly, this field may be considered
    # the best effort maximum size of a batch. A batch will fill with messages
    # until this size is reached (or the max message count, or batch timeout is
    # exceeded).  If adding a new message to the batch would cause the batch to
    # exceed the preferred max bytes, then the current batch is closed and written
    # to a block, and a new batch containing the new message is created.  If a
    # message larger than the preferred max bytes is received, then its batch
    # will contain only that message.  Because messages may be larger than
    # preferred max bytes (up to AbsoluteMaxBytes), some batches may exceed
    # the preferred max bytes, but will always contain exactly one transaction.
    PreferredMaxBytes: 2 MB

# Max Channels is the maximum number of channels to allow on the ordering
# network. When set to 0, this implies no maximum number of channels.
MaxChannels: 0

Kafka:
    # Brokers: A list of Kafka brokers to which the orderer connects. Edit
    # this list to identify the brokers of the ordering service.
    # NOTE: Use IP:port notation.
    Brokers:
        - kafka0:9092
        - kafka1:9092
        - kafka2:9092

# EtcdRaft defines configuration which must be set when the "etcdraft"
# orderertype is chosen.
EtcdRaft:
    # The set of Raft replicas for this network. For the etcd/raft-based
    # implementation, we expect every replica to also be an OSN. Therefore,
    # a subset of the host:port items enumerated in this list should be
    # replicated under the Orderer.Addresses key above.
    Consenters:
        - Host: raft0.example.com
          Port: 7050
          ClientTLSCert: path/to/ClientTLSCert0
          ServerTLSCert: path/to/ServerTLSCert0
        - Host: raft1.example.com
          Port: 7050
          ClientTLSCert: path/to/ClientTLSCert1
          ServerTLSCert: path/to/ServerTLSCert1
        - Host: raft2.example.com
          Port: 7050
          ClientTLSCert: path/to/ClientTLSCert2
          ServerTLSCert: path/to/ServerTLSCert2

    # Options to be specified for all the etcd/raft nodes. The values here
    # are the defaults for all new channels and can be modified on a
    # per-channel basis via configuration updates.
    Options:
```

```
            # TickInterval is the time interval between two Node.Tick invocations.
            TickInterval: 500ms

            # ElectionTick is the number of Node.Tick invocations that must pass
            # between elections. That is, if a follower does not receive any
            # message from the leader of current term before ElectionTick has
            # elapsed, it will become candidate and start an election.
            # ElectionTick must be greater than HeartbeatTick.
            ElectionTick: 10

            # HeartbeatTick is the number of Node.Tick invocations that must
            # pass between heartbeats. That is, a leader sends heartbeat
            # messages to maintain its leadership every HeartbeatTick ticks.
            HeartbeatTick: 1

            # MaxInflightBlocks limits the max number of in-flight append messages
            # during optimistic replication phase.
            MaxInflightBlocks: 5

            # SnapshotIntervalSize defines number of bytes per which a snapshot is taken
            SnapshotIntervalSize: 20 MB

    # Organizations lists the orgs participating on the orderer side of the
    # network.
    Organizations:

    # Policies defines the set of policies at this level of the config tree
    # For Orderer policies, their canonical path is
    #   /Channel/Orderer/<PolicyName>
    Policies:
        Readers:
            Type: ImplicitMeta
            Rule: "ANY Readers"
        Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
        Admins:
            Type: ImplicitMeta
            Rule: "MAJORITY Admins"
        # BlockValidation specifies what signatures must be included in the block
        # from the orderer for the peer to validate it.
        BlockValidation:
            Type: ImplicitMeta
            Rule: "ANY Writers"

    # Capabilities describes the orderer level capabilities, see the
    # dedicated Capabilities section elsewhere in this file for a full
    # description
    Capabilities:
        <<: *OrdererCapabilities

################################################################################
#########
#
```

```
#   CHANNEL
#
#   This section defines the values to encode into a config transaction or
#   genesis block for channel related parameters.
#
################################################################################
#########
Channel: &ChannelDefaults
    # Policies defines the set of policies at this level of the config tree
    # For Channel policies, their canonical path is
    #   /Channel/<PolicyName>
    Policies:
        # Who may invoke the 'Deliver' API
        Readers:
            Type: ImplicitMeta
            Rule: "ANY Readers"
        # Who may invoke the 'Broadcast' API
        Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
        # By default, who may modify elements at this config level
        Admins:
            Type: ImplicitMeta
            Rule: "MAJORITY Admins"


    # Capabilities describes the channel level capabilities, see the
    # dedicated Capabilities section elsewhere in this file for a full
    # description
    Capabilities:
        <<: *ChannelCapabilities

################################################################################
#########
#
#   PROFILES
#
#   Different configuration profiles may be encoded here to be specified as
#   parameters to the configtxgen tool. The profiles which specify consortiums
#   are to be used for generating the orderer genesis block. With the correct
#   consortium members defined in the orderer genesis block, channel creation
#   requests may be generated with only the org member names and a consortium
#   name.
#
################################################################################
#########
Profiles:

    # SampleSingleMSPSolo defines a configuration which uses the Solo orderer,
    # and contains a single MSP definition (the MSP sampleconfig).
    # The Consortium SampleConsortium has only a single member, SampleOrg.
    SampleSingleMSPSolo:
        <<: *ChannelDefaults
        Orderer:
```

```
        <<: *OrdererDefaults
        Organizations:
            - *SampleOrg
    Consortiums:
        SampleConsortium:
            Organizations:
                - *SampleOrg

# SampleSingleMSPKafka defines a configuration that differs from the
# SampleSingleMSPSolo one only in that it uses the Kafka-based orderer.
SampleSingleMSPKafka:
    <<: *ChannelDefaults
    Orderer:
        <<: *OrdererDefaults
        OrdererType: kafka
        Organizations:
            - *SampleOrg
    Consortiums:
        SampleConsortium:
            Organizations:
                - *SampleOrg

# SampleInsecureSolo defines a configuration which uses the Solo orderer,
# contains no MSP definitions, and allows all transactions and channel
# creation requests for the consortium SampleConsortium.
SampleInsecureSolo:
    <<: *ChannelDefaults
    Orderer:
        <<: *OrdererDefaults
    Consortiums:
        SampleConsortium:
            Organizations:

# SampleInsecureKafka defines a configuration that differs from the
# SampleInsecureSolo one only in that it uses the Kafka-based orderer.
SampleInsecureKafka:
    <<: *ChannelDefaults
    Orderer:
        OrdererType: kafka
        <<: *OrdererDefaults
    Consortiums:
        SampleConsortium:
            Organizations:

# SampleDevModeSolo defines a configuration which uses the Solo orderer,
# contains the sample MSP as both orderer and consortium member, and
# requires only basic membership for admin privileges. It also defines
# an Application on the ordering system channel, which should usually
# be avoided.
SampleDevModeSolo:
    <<: *ChannelDefaults
    Orderer:
        <<: *OrdererDefaults
        Organizations:
```

```
            - <<: *SampleOrg
              Policies:
                  <<: *SampleOrgPolicies
                  Admins:
                      Type: Signature
                      Rule: "OR('SampleOrg.member')"
    Application:
        <<: *ApplicationDefaults
        Organizations:
            - <<: *SampleOrg
              Policies:
                  <<: *SampleOrgPolicies
                  Admins:
                      Type: Signature
                      Rule: "OR('SampleOrg.member')"
    Consortiums:
        SampleConsortium:
            Organizations:
                - <<: *SampleOrg
                  Policies:
                      <<: *SampleOrgPolicies
                      Admins:
                          Type: Signature
                          Rule: "OR('SampleOrg.member')"

# SampleDevModeKafka defines a configuration that differs from the
# SampleDevModeSolo one only in that it uses the Kafka-based orderer.
SampleDevModeKafka:
    <<: *ChannelDefaults
    Orderer:
        <<: *OrdererDefaults
        OrdererType: kafka
        Organizations:
            - <<: *SampleOrg
              Policies:
                  <<: *SampleOrgPolicies
                  Admins:
                      Type: Signature
                      Rule: "OR('SampleOrg.member')"
    Application:
        <<: *ApplicationDefaults
        Organizations:
            - <<: *SampleOrg
              Policies:
                  <<: *SampleOrgPolicies
                  Admins:
                      Type: Signature
                      Rule: "OR('SampleOrg.member')"
    Consortiums:
        SampleConsortium:
            Organizations:
                - <<: *SampleOrg
                  Policies:
                      <<: *SampleOrgPolicies
```

```
                Admins:
                    Type: Signature
                    Rule: "OR('SampleOrg.member')"

    # SampleSingleMSPChannel defines a channel with only the sample org as a
    # member. It is designed to be used in conjunction with SampleSingleMSPSolo
    # and SampleSingleMSPKafka orderer profiles.  Note, for channel creation
    # profiles, only the 'Application' section and consortium # name are
    # considered.
    SampleSingleMSPChannel:
        Consortium: SampleConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *SampleOrg

    # SampleDevModeEtcdRaft defines a configuration that differs from the
    # SampleDevModeSolo one only in that it uses the etcd/raft-based orderer.
    SampleDevModeEtcdRaft:
        <<: *ChannelDefaults
        Orderer:
            <<: *OrdererDefaults
            OrdererType: etcdraft
            Organizations:
                - <<: *SampleOrg
                  Policies:
                      <<: *SampleOrgPolicies
                      Admins:
                          Type: Signature
                          Rule: "OR('SampleOrg.member')"
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - <<: *SampleOrg
                  Policies:
                      <<: *SampleOrgPolicies
                      Admins:
                          Type: Signature
                          Rule: "OR('SampleOrg.member')"
        Consortiums:
            SampleConsortium:
                Organizations:
                    - <<: *SampleOrg
                      Policies:
                          <<: *SampleOrgPolicies
                          Admins:
                              Type: Signature
                              Rule: "OR('SampleOrg.member')"
```

core.yaml

```
###############################################################################
#
```

```
#    Peer section
#
############################################################################
########
peer:

    # The Peer id is used for identifying this Peer instance.
    id: jdoe

    # The networkId allows for logical seperation of networks
    networkId: dev

    # The Address at local network interface this Peer will listen on.
    # By default, it will listen on all network interfaces
    listenAddress: 0.0.0.0:7051

    # The endpoint this peer uses to listen for inbound chaincode connections.
    # If this is commented-out, the listen address is selected to be
    # the peer's address (see below) with port 7052
    # chaincodeListenAddress: 0.0.0.0:7052

    # The endpoint the chaincode for this peer uses to connect to the peer.
    # If this is not specified, the chaincodeListenAddress address is selected.
    # And if chaincodeListenAddress is not specified, address is selected from
    # peer listenAddress.
    # chaincodeAddress: 0.0.0.0:7052

    # When used as peer config, this represents the endpoint to other peers
    # in the same organization. For peers in other organization, see
    # gossip.externalEndpoint for more info.
    # When used as CLI config, this means the peer's endpoint to interact with
    address: 0.0.0.0:7051

    # Whether the Peer should programmatically determine its address
    # This case is useful for docker containers.
    addressAutoDetect: false

    # Setting for runtime.GOMAXPROCS(n). If n < 1, it does not change the
    # current setting
    gomaxprocs: -1

    # Keepalive settings for peer server and clients
    keepalive:
        # MinInterval is the minimum permitted time between client pings.
        # If clients send pings more frequently, the peer server will
        # disconnect them
        minInterval: 60s
        # Client keepalive settings for communicating with other peer nodes
        client:
            # Interval is the time between pings to peer nodes.  This must
            # greater than or equal to the minInterval specified by peer
            # nodes
            interval: 60s
            # Timeout is the duration the client waits for a response from
```

```
        # peer nodes before closing the connection
        timeout: 20s
    # DeliveryClient keepalive settings for communication with ordering
    # nodes.
    deliveryClient:
        # Interval is the time between pings to ordering nodes.  This must
        # greater than or equal to the minInterval specified by ordering
        # nodes.
        interval: 60s
        # Timeout is the duration the client waits for a response from
        # ordering nodes before closing the connection
        timeout: 20s


# Gossip related configuration
gossip:
    # Bootstrap set to initialize gossip with.
    # This is a list of other peers that this peer reaches out to at startup.
    # Important: The endpoints here have to be endpoints of peers in the same
    # organization, because the peer would refuse connecting to these endpoints
    # unless they are in the same organization as the peer.
    bootstrap: 127.0.0.1:7051

    # NOTE: orgLeader and useLeaderElection parameters are mutual exclusive.
    # Setting both to true would result in the termination of the peer
    # since this is undefined state. If the peers are configured with
    # useLeaderElection=false, make sure there is at least 1 peer in the
    # organization that its orgLeader is set to true.

    # Defines whenever peer will initialize dynamic algorithm for
    # "leader" selection, where leader is the peer to establish
    # connection with ordering service and use delivery protocol
    # to pull ledger blocks from ordering service. It is recommended to
    # use leader election for large networks of peers.
    useLeaderElection: true
    # Statically defines peer to be an organization "leader",
    # where this means that current peer will maintain connection
    # with ordering service and disseminate block across peers in
    # its own organization
    orgLeader: false

    # Interval for membershipTracker polling
    membershipTrackerInterval: 5s

    # Overrides the endpoint that the peer publishes to peers
    # in its organization. For peers in foreign organizations
    # see 'externalEndpoint'
    endpoint:
    # Maximum count of blocks stored in memory
    maxBlockCountToStore: 100
    # Max time between consecutive message pushes(unit: millisecond)
    maxPropagationBurstLatency: 10ms
    # Max number of messages stored until a push is triggered to remote peers
    maxPropagationBurstSize: 10
```

```
        # Number of times a message is pushed to remote peers
        propagateIterations: 1
        # Number of peers selected to push messages to
        propagatePeerNum: 3
        # Determines frequency of pull phases(unit: second)
        # Must be greater than digestWaitTime + responseWaitTime
        pullInterval: 4s
        # Number of peers to pull from
        pullPeerNum: 3
        # Determines frequency of pulling state info messages from peers(unit: second)
        requestStateInfoInterval: 4s
        # Determines frequency of pushing state info messages to peers(unit: second)
        publishStateInfoInterval: 4s
        # Maximum time a stateInfo message is kept until expired
        stateInfoRetentionInterval:
        # Time from startup certificates are included in Alive messages(unit: second)
        publishCertPeriod: 10s
        # Should we skip verifying block messages or not (currently not in use)
        skipBlockVerification: false
        # Dial timeout(unit: second)
        dialTimeout: 3s
        # Connection timeout(unit: second)
        connTimeout: 2s
        # Buffer size of received messages
        recvBuffSize: 20
        # Buffer size of sending messages
        sendBuffSize: 200
        # Time to wait before pull engine processes incoming digests (unit: second)
        # Should be slightly smaller than requestWaitTime
        digestWaitTime: 1s
        # Time to wait before pull engine removes incoming nonce (unit: milliseconds)
        # Should be slightly bigger than digestWaitTime
        requestWaitTime: 1500ms
        # Time to wait before pull engine ends pull (unit: second)
        responseWaitTime: 2s
        # Alive check interval(unit: second)
        aliveTimeInterval: 5s
        # Alive expiration timeout(unit: second)
        aliveExpirationTimeout: 25s
        # Reconnect interval(unit: second)
        reconnectInterval: 25s
        # This is an endpoint that is published to peers outside of the organization.
        # If this isn't set, the peer will not be known to other organizations.
        externalEndpoint:
        # Leader election service configuration
        election:
            # Longest time peer waits for stable membership during leader election startup (unit:
econd)
            startupGracePeriod: 15s
            # Interval gossip membership samples to check its stability (unit: second)
            membershipSampleInterval: 1s
            # Time passes since last declaration message before peer decides to perform leader el
ction (unit: second)
            leaderAliveThreshold: 10s
```

```
            # Time between peer sends propose message and declares itself as a leader (sends de
laration message) (unit: second)
            leaderElectionDuration: 5s

        pvtData:
            # pullRetryThreshold determines the maximum duration of time private data correspo
ding for a given block
            # would be attempted to be pulled from peers until the block would be committed wi
hout the private data
            pullRetryThreshold: 60s
            # As private data enters the transient store, it is associated with the peer's ledger's hei
ht at that time.
            # transientstoreMaxBlockRetention defines the maximum difference between the curr
nt ledger's height upon commit,
            # and the private data residing inside the transient store that is guaranteed not to be
urged.
            # Private data is purged from the transient store when blocks with sequences that are
multiples
            # of transientstoreMaxBlockRetention are committed.
            transientstoreMaxBlockRetention: 1000
            # pushAckTimeout is the maximum time to wait for an acknowledgement from each p
er
            # at private data push at endorsement time.
            pushAckTimeout: 3s
            # Block to live pulling margin, used as a buffer
            # to prevent peer from trying to pull private data
            # from peers that is soon to be purged in next N blocks.
            # This helps a newly joined peer catch up to current
            # blockchain height quicker.
            btlPullMargin: 10
            # the process of reconciliation is done in an endless loop, while in each iteration recon
iler tries to
            # pull from the other peers the most recent missing blocks with a maximum batch size
limitation.
            # reconcileBatchSize determines the maximum batch size of missing private data that
will be reconciled in a
            # single iteration.
            reconcileBatchSize: 10
            # reconcileSleepInterval determines the time reconciler sleeps from end of an iteration
until the beginning
            # of the next reconciliation iteration.
            reconcileSleepInterval: 1m
            # reconciliationEnabled is a flag that indicates whether private data reconciliation is e
able or not.
            reconciliationEnabled: true

        # Gossip state transfer related configuration
        state:
            # indicates whenever state transfer is enabled or not
            # default value is true, i.e. state transfer is active
            # and takes care to sync up missing blocks allowing
            # lagging peer to catch up to speed with rest network
            enabled: true
            # checkInterval interval to check whether peer is lagging behind enough to
```

```
        # request blocks via state transfer from another peer.
        checkInterval: 10s
        # responseTimeout amount of time to wait for state transfer response from
        # other peers
        responseTimeout: 3s
        # batchSize the number of blocks to request via state transfer from another peer
        batchSize: 10
        # blockBufferSize reflect the maximum distance between lowest and
        # highest block sequence number state buffer to avoid holes.
        # In order to ensure absence of the holes actual buffer size
        # is twice of this distance
        blockBufferSize: 100
        # maxRetries maximum number of re-tries to ask
        # for single state transfer request
        maxRetries: 3

# TLS Settings
# Note that peer-chaincode connections through chaincodeListenAddress is
# not mutual TLS auth. See comments on chaincodeListenAddress for more info
tls:
    # Require server-side TLS
    enabled:  false
    # Require client certificates / mutual TLS.
    # Note that clients that are not configured to use a certificate will
    # fail to connect to the peer.
    clientAuthRequired: false
    # X.509 certificate used for TLS server
    cert:
        file: tls/server.crt
    # Private key used for TLS server (and client if clientAuthEnabled
    # is set to true
    key:
        file: tls/server.key
    # Trusted root certificate chain for tls.cert
    rootcert:
        file: tls/ca.crt
    # Set of root certificate authorities used to verify client certificates
    clientRootCAs:
        files:
          - tls/ca.crt
    # Private key used for TLS when making client connections.  If
    # not set, peer.tls.key.file will be used instead
    clientKey:
        file:
    # X.509 certificate used for TLS when making client connections.
    # If not set, peer.tls.cert.file will be used instead
    clientCert:
        file:

# Authentication contains configuration parameters related to authenticating
# client messages
authentication:
    # the acceptable difference between the current server time and the
    # client's time as specified in a client request message
```

```
    timewindow: 15m

# Path on the file system where peer will store data (eg ledger). This
# location must be access control protected to prevent unintended
# modification that might corrupt the peer operations.
fileSystemPath: /var/hyperledger/production

# BCCSP (Blockchain crypto provider): Select which crypto implementation or
# library to use
BCCSP:
    Default: SW
    # Settings for the SW crypto provider (i.e. when DEFAULT: SW)
    SW:
        # TODO: The default Hash and Security level needs refactoring to be
        # fully configurable. Changing these defaults requires coordination
        # SHA2 is hardcoded in several places, not only BCCSP
        Hash: SHA2
        Security: 256
        # Location of Key Store
        FileKeyStore:
            # If "", defaults to 'mspConfigPath'/keystore
            KeyStore:
    # Settings for the PKCS#11 crypto provider (i.e. when DEFAULT: PKCS11)
    PKCS11:
        # Location of the PKCS11 module library
        Library:
        # Token Label
        Label:
        # User PIN
        Pin:
        Hash:
        Security:
        FileKeyStore:
            KeyStore:

# Path on the file system where peer will find MSP local configurations
mspConfigPath: msp

# Identifier of the local MSP
# ----!!!!IMPORTANT!!!-!!!IMPORTANT!!!-!!!IMPORTANT!!!!----
# Deployers need to change the value of the localMspId string.
# In particular, the name of the local MSP ID of a peer needs
# to match the name of one of the MSPs in each of the channel
# that this peer is a member of. Otherwise this peer's messages
# will not be identified as valid by other nodes.
localMspId: SampleOrg

# CLI common client config options
client:
    # connection timeout
    connTimeout: 3s

# Delivery service related config
deliveryclient:
```

```
        # It sets the total time the delivery service may spend in reconnection
        # attempts until its retry logic gives up and returns an error
        reconnectTotalTimeThreshold: 3600s

        # It sets the delivery service <-> ordering service node connection timeout
        connTimeout: 3s

        # It sets the delivery service maximal delay between consecutive retries
        reConnectBackoffThreshold: 3600s

    # Type for the local MSP - by default it's of type bccsp
    localMspType: bccsp

    # Used with Go profiling tools only in none production environment. In
    # production, it should be disabled (eg enabled: false)
    profile:
        enabled:     false
        listenAddress: 0.0.0.0:6060

    # The admin service is used for administrative operations such as
    # control over logger levels, etc.
    # Only peer administrators can use the service.
    adminService:
        # The interface and port on which the admin server will listen on.
        # If this is commented out, or the port number is equal to the port
        # of the peer listen address - the admin service is attached to the
        # peer's service (defaults to 7051).
        #listenAddress: 0.0.0.0:7055

    # Handlers defines custom handlers that can filter and mutate
    # objects passing within the peer, such as:
    #   Auth filter - reject or forward proposals from clients
    #   Decorators  - append or mutate the chaincode input passed to the chaincode
    #   Endorsers   - Custom signing over proposal response payload and its mutation
    # Valid handler definition contains:
    #   - A name which is a factory method name defined in
    #     core/handlers/library/library.go for statically compiled handlers
    #   - library path to shared object binary for pluggable filters
    # Auth filters and decorators are chained and executed in the order that
    # they are defined. For example:
    # authFilters:
    #   -
    #     name: FilterOne
    #     library: /opt/lib/filter.so
    #   -
    #     name: FilterTwo
    # decorators:
    #   -
    #     name: DecoratorOne
    #   -
    #     name: DecoratorTwo
    #     library: /opt/lib/decorator.so
    # Endorsers are configured as a map that its keys are the endorsement system chaincodes
hat are being overridden.
```

```
    # Below is an example that overrides the default ESCC and uses an endorsement plugin tha
 has the same functionality
    # as the default ESCC.
    # If the 'library' property is missing, the name is used as the constructor method in the built
n library similar
    # to auth filters and decorators.
    # endorsers:
    #   escc:
    #     name: DefaultESCC
    #     library: /etc/hyperledger/fabric/plugin/escc.so
    handlers:
        authFilters:
          -
            name: DefaultAuth
          -
            name: ExpirationCheck    # This filter checks identity x509 certificate expiration
        decorators:
          -
            name: DefaultDecorator
        endorsers:
          escc:
            name: DefaultEndorsement
            library:
        validators:
          vscc:
            name: DefaultValidation
            library:

    #    library: /etc/hyperledger/fabric/plugin/escc.so
    # Number of goroutines that will execute transaction validation in parallel.
    # By default, the peer chooses the number of CPUs on the machine. Set this
    # variable to override that choice.
    # NOTE: overriding this value might negatively influence the performance of
    # the peer so please change this value only if you know what you're doing
    validatorPoolSize:

    # The discovery service is used by clients to query information about peers,
    # such as - which peers have joined a certain channel, what is the latest
    # channel config, and most importantly - given a chaincode and a channel,
    # what possible sets of peers satisfy the endorsement policy.
    discovery:
        enabled: true
        # Whether the authentication cache is enabled or not.
        authCacheEnabled: true
        # The maximum size of the cache, after which a purge takes place
        authCacheMaxSize: 1000
        # The proportion (0 to 1) of entries that remain in the cache after the cache is purged du
 to overpopulation
        authCachePurgeRetentionRatio: 0.75
        # Whether to allow non-admins to perform non channel scoped queries.
        # When this is false, it means that only peer admins can perform non channel scoped qu
ries.
        orgMembersAllowedAccess: false
############################################################################
```

```
########
#
#    VM section
#
############################################################################
########
vm:

    # Endpoint of the vm management system.  For docker can be one of the following in gene
al
    # unix:///var/run/docker.sock
    # http://localhost:2375
    # https://localhost:2376
    endpoint: unix:///var/run/docker.sock

    # settings for docker vms
    docker:
        tls:
            enabled: false
            ca:
                file: docker/ca.crt
            cert:
                file: docker/tls.crt
            key:
                file: docker/tls.key

        # Enables/disables the standard out/err from chaincode containers for
        # debugging purposes
        attachStdout: false

        # Parameters on creating docker container.
        # Container may be efficiently created using ipam & dns-server for cluster
        # NetworkMode - sets the networking mode for the container. Supported
        # standard values are: `host`(default),`bridge`,`ipvlan`,`none`.
        # Dns - a list of DNS servers for the container to use.
        # Note:  `Privileged` `Binds` `Links` and `PortBindings` properties of
        # Docker Host Config are not supported and will not be used if set.
        # LogConfig - sets the logging driver (Type) and related options
        # (Config) for Docker. For more info,
        # https://docs.docker.com/engine/admin/logging/overview/
        # Note: Set LogConfig using Environment Variables is not supported.
        hostConfig:
            NetworkMode: host
            Dns:
              # - 192.168.0.1
            LogConfig:
                Type: json-file
                Config:
                    max-size: "50m"
                    max-file: "5"
            Memory: 2147483648

############################################################################
########
```

```
#
#    Chaincode section
#
################################################################################
########
chaincode:

    # The id is used by the Chaincode stub to register the executing Chaincode
    # ID with the Peer and is generally supplied through ENV variables
    # the `path` form of ID is provided when installing the chaincode.
    # The `name` is used for all other requests and can be any string.
    id:
        path:
        name:

    # Generic builder environment, suitable for most chaincode types
    builder: $(DOCKER_NS)/fabric-ccenv:latest

    # Enables/disables force pulling of the base docker images (listed below)
    # during user chaincode instantiation.
    # Useful when using moving image tags (such as :latest)
    pull: false

    golang:
        # golang will never need more than baseos
        runtime: $(BASE_DOCKER_NS)/fabric-baseos:$(ARCH)-$(BASE_VERSION)

        # whether or not golang chaincode should be linked dynamically
        dynamicLink: false

    car:
        # car may need more facilities (JVM, etc) in the future as the catalog
        # of platforms are expanded.  For now, we can just use baseos
        runtime: $(BASE_DOCKER_NS)/fabric-baseos:$(ARCH)-$(BASE_VERSION)

    java:
        # This is an image based on java:openjdk-8 with addition compiler
        # tools added for java shim layer packaging.
        # This image is packed with shim layer libraries that are necessary
        # for Java chaincode runtime.
        runtime: $(DOCKER_NS)/fabric-javaenv:$(ARCH)-$(PROJECT_VERSION)

    node:
        # need node.js engine at runtime, currently available in baseimage
        # but not in baseos
        runtime: $(BASE_DOCKER_NS)/fabric-baseimage:$(ARCH)-$(BASE_VERSION)

    # Timeout duration for starting up a container and waiting for Register
    # to come through. 1sec should be plenty for chaincode unit tests
    startuptimeout: 300s

    # Timeout duration for Invoke and Init calls to prevent runaway.
    # This timeout is used by all chaincodes in all the channels, including
    # system chaincodes.
```

```
    # Note that during Invoke, if the image is not available (e.g. being
    # cleaned up when in development environment), the peer will automatically
    # build the image, which might take more time. In production environment,
    # the chaincode image is unlikely to be deleted, so the timeout could be
    # reduced accordingly.
    executetimeout: 30s

    # There are 2 modes: "dev" and "net".
    # In dev mode, user runs the chaincode after starting peer from
    # command line on local machine.
    # In net mode, peer will run chaincode in a docker container.
    mode: net

    # keepalive in seconds. In situations where the communiction goes through a
    # proxy that does not support keep-alive, this parameter will maintain connection
    # between peer and chaincode.
    # A value <= 0 turns keepalive off
    keepalive: 0

    # system chaincodes whitelist. To add system chaincode "myscc" to the
    # whitelist, add "myscc: enable" to the list below, and register in
    # chaincode/importsysccs.go
    system:
        cscc: enable
        lscc: enable
        escc: enable
        vscc: enable
        qscc: enable

    # System chaincode plugins:
    # System chaincodes can be loaded as shared objects compiled as Go plugins.
    # See examples/plugins/scc for an example.
    # Plugins must be white listed in the chaincode.system section above.
    systemPlugins:
      # example configuration:
      # - enabled: true
      #   name: myscc
      #   path: /opt/lib/myscc.so
      #   invokableExternal: true
      #   invokableCC2CC: true

    # Logging section for the chaincode container
    logging:
      # Default level for all loggers within the chaincode container
      level:  info
      # Override default level for the 'shim' logger
      shim:   warning
      # Format for the chaincode container logs
      format: '%{color}%{time:2006-01-02 15:04:05.000 MST} [%{module}] %{shortfunc} -> %{le
el:.4s} %{id:03x}%{color:reset} %{message}'

###############################################################################
########
#
```

```
#    Ledger section - ledger configuration encompasses both the blockchain
#    and the state
#
###############################################################################
########
ledger:

  blockchain:

  state:
    # stateDatabase - options are "goleveldb", "CouchDB"
    # goleveldb - default state database stored in goleveldb.
    # CouchDB - store state database in CouchDB
    stateDatabase: goleveldb
    # Limit on the number of records to return per query
    totalQueryLimit: 100000
    couchDBConfig:
       # It is recommended to run CouchDB on the same server as the peer, and
       # not map the CouchDB container port to a server port in docker-compose.
       # Otherwise proper security must be provided on the connection between
       # CouchDB client (on the peer) and server.
       couchDBAddress: 127.0.0.1:5984
       # This username must have read and write authority on CouchDB
       username:
       # The password is recommended to pass as an environment variable
       # during start up (eg CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD).
       # If it is stored here, the file must be access control protected
       # to prevent unintended users from discovering the password.
       password:
       # Number of retries for CouchDB errors
       maxRetries: 3
       # Number of retries for CouchDB errors during peer startup
       maxRetriesOnStartup: 12
       # CouchDB request timeout (unit: duration, e.g. 20s)
       requestTimeout: 35s
       # Limit on the number of records per each CouchDB query
       # Note that chaincode queries are only bound by totalQueryLimit.
       # Internally the chaincode may execute multiple CouchDB queries,
       # each of size internalQueryLimit.
       internalQueryLimit: 1000
       # Limit on the number of records per CouchDB bulk update batch
       maxBatchUpdateSize: 1000
       # Warm indexes after every N blocks.
       # This option warms any indexes that have been
       # deployed to CouchDB after every N blocks.
       # A value of 1 will warm indexes after every block commit,
       # to ensure fast selector queries.
       # Increasing the value may improve write efficiency of peer and CouchDB,
       # but may degrade query response time.
       warmIndexesAfterNBlocks: 1
       # Create the _global_changes system database
       # This is optional.  Creating the global changes database will require
       # additional system resources to track changes and maintain the database
       createGlobalChangesDB: false
```

```
  history:
    # enableHistoryDatabase - options are true or false
    # Indicates if the history of key updates should be stored.
    # All history 'index' will be stored in goleveldb, regardless if using
    # CouchDB or alternate database for the state.
    enableHistoryDatabase: true

################################################################################
########
#
#    Operations section
#
################################################################################
########
operations:
    # host and port for the operations server
    listenAddress: 127.0.0.1:9443

    # TLS configuration for the operations endpoint
    tls:
        # TLS enabled
        enabled: false

        # path to PEM encoded server certificate for the operations server
        cert:
            file:

        # path to PEM encoded server key for the operations server
        key:
            file:

        # most operations service endpoints require client authentication when TLS
        # is enabled. clientAuthRequired requires client certificate authentication
        # at the TLS layer to access all resources.
        clientAuthRequired: false

        # paths to PEM encoded ca certificates to trust for client authentication
        clientRootCAs:
            files: []

################################################################################
########
#
#    Metrics section
#
################################################################################
########
metrics:
    # metrics provider is one of statsd, prometheus, or disabled
    provider: disabled

    # statsd configuration
    statsd:
```

```
    # network type: tcp or udp
    network: udp

    # statsd server address
    address: 127.0.0.1:8125

    # the interval at which locally cached counters and gauges are pushed
    # to statsd; timings are pushed immediately
    writeInterval: 10s

    # prefix is prepended to all emitted statsd metrics
    prefix:
```

orderer.yaml

```
---
################################################################################
#########
#
#   Orderer Configuration
#
#   - This controls the type and configuration of the orderer.
#
################################################################################
#########
General:

    # Ledger Type: The ledger type to provide to the orderer.
    # Two non-production ledger types are provided for test purposes only:
    #  - ram: An in-memory ledger whose contents are lost on restart.
    #  - json: A simple file ledger that writes blocks to disk in JSON format.
    # Only one production ledger type is provided:
    #  - file: A production file-based ledger.
    LedgerType: file

    # Listen address: The IP on which to bind to listen.
    ListenAddress: 127.0.0.1

    # Listen port: The port on which to bind to listen.
    ListenPort: 7050

    # TLS: TLS settings for the GRPC server.
    TLS:
        Enabled: false
        # PrivateKey governs the file location of the private key of the TLS certificate.
        PrivateKey: tls/server.key
        # Certificate governs the file location of the server TLS certificate.
        Certificate: tls/server.crt
        RootCAs:
          - tls/ca.crt
        ClientAuthRequired: false
        ClientRootCAs:
    # Keepalive settings for the GRPC server.
    Keepalive:
```

```
        # ServerMinInterval is the minimum permitted time between client pings.
        # If clients send pings more frequently, the server will
        # disconnect them.
        ServerMinInterval: 60s
        # ServerInterval is the time between pings to clients.
        ServerInterval: 7200s
        # ServerTimeout is the duration the server waits for a response from
        # a client before closing the connection.
        ServerTimeout: 20s
    # Cluster settings for ordering service nodes that communicate with other ordering service
nodes
    # such as Raft based ordering service.
    Cluster:
        # SendBufferSize is the maximum number of messages in the egress buffer.
        # Consensus messages are dropped if the buffer is full, and transaction
        # messages are waiting for space to be freed.
        SendBufferSize: 10
        # ClientCertificate governs the file location of the client TLS certificate
        # used to establish mutual TLS connections with other ordering service nodes.
        ClientCertificate:
        # ClientPrivateKey governs the file location of the private key of the client TLS certificate.
        ClientPrivateKey:
        # The below 4 properties should be either set together, or be unset together.
        # If they are set, then the orderer node uses a separate listener for intra-cluster
        # communication. If they are unset, then the general orderer listener is used.
        # This is useful if you want to use a different TLS server certificates on the
        # client-facing and the intra-cluster listeners.

        # ListenPort defines the port on which the cluster listens to connections.
        ListenPort:
        # ListenAddress defines the IP on which to listen to intra-cluster communication.
        ListenAddress:
        # ServerCertificate defines the file location of the server TLS certificate used for intra-clus
er
        # communication.
        ServerCertificate:
        # ServerPrivateKey defines the file location of the private key of the TLS certificate.
        ServerPrivateKey:
    # Genesis method: The method by which the genesis block for the orderer
    # system channel is specified. Available options are "provisional", "file":
    #   - provisional: Utilizes a genesis profile, specified by GenesisProfile,
    #                  to dynamically generate a new genesis block.
    #   - file: Uses the file provided by GenesisFile as the genesis block.
    GenesisMethod: provisional

    # Genesis profile: The profile to use to dynamically generate the genesis
    # block to use when initializing the orderer system channel and
    # GenesisMethod is set to "provisional". See the configtx.yaml file for the
    # descriptions of the available profiles. Ignored if GenesisMethod is set to
    # "file".
    GenesisProfile: SampleInsecureSolo

    # Genesis file: The file containing the genesis block to use when
    # initializing the orderer system channel and GenesisMethod is set to
```

```
    # "file". Ignored if GenesisMethod is set to "provisional".
    GenesisFile: genesisblock

    # LocalMSPDir is where to find the private crypto material needed by the
    # orderer. It is set relative here as a default for dev environments but
    # should be changed to the real location in production.
    LocalMSPDir: msp

    # LocalMSPID is the identity to register the local MSP material with the MSP
    # manager. IMPORTANT: The local MSP ID of an orderer needs to match the MSP
    # ID of one of the organizations defined in the orderer system channel's
    # /Channel/Orderer configuration. The sample organization defined in the
    # sample configuration provided has an MSP ID of "SampleOrg".
    LocalMSPID: SampleOrg

    # Enable an HTTP service for Go "pprof" profiling as documented at:
    # https://golang.org/pkg/net/http/pprof
    Profile:
        Enabled: false
        Address: 0.0.0.0:6060

    # BCCSP configures the blockchain crypto service providers.
    BCCSP:
        # Default specifies the preferred blockchain crypto service provider
        # to use. If the preferred provider is not available, the software
        # based provider ("SW") will be used.
        # Valid providers are:
        #  - SW: a software based crypto provider
        #  - PKCS11: a CA hardware security module crypto provider.
        Default: SW

        # SW configures the software based blockchain crypto provider.
        SW:
            # TODO: The default Hash and Security level needs refactoring to be
            # fully configurable. Changing these defaults requires coordination
            # SHA2 is hardcoded in several places, not only BCCSP
            Hash: SHA2
            Security: 256
            # Location of key store. If this is unset, a location will be
            # chosen using: 'LocalMSPDir'/keystore
            FileKeyStore:
                KeyStore:

    # Authentication contains configuration parameters related to authenticating
    # client messages
    Authentication:
        # the acceptable difference between the current server time and the
        # client's time as specified in a client request message
        TimeWindow: 15m

################################################################################
#########
#
#   SECTION: File Ledger
```

```
#
#   - This section applies to the configuration of the file or json ledgers.
#
################################################################################
#########
FileLedger:

    # Location: The directory to store the blocks in.
    # NOTE: If this is unset, a new temporary location will be chosen every time
    # the orderer is restarted, using the prefix specified by Prefix.
    Location: /var/hyperledger/production/orderer

    # The prefix to use when generating a ledger directory in temporary space.
    # Otherwise, this value is ignored.
    Prefix: hyperledger-fabric-ordererledger

################################################################################
#########
#
#   SECTION: RAM Ledger
#
#   - This section applies to the configuration of the RAM ledger.
#
################################################################################
#########
RAMLedger:

    # History Size: The number of blocks that the RAM ledger is set to retain.
    # WARNING: Appending a block to the ledger might cause the oldest block in
    # the ledger to be dropped in order to limit the number total number blocks
    # to HistorySize. For example, if history size is 10, when appending block
    # 10, block 0 (the genesis block!) will be dropped to make room for block 10.
    HistorySize: 1000

################################################################################
#########
#
#   SECTION: Kafka
#
#   - This section applies to the configuration of the Kafka-based orderer, and
#     its interaction with the Kafka cluster.
#
################################################################################
#########
Kafka:

    # Retry: What do if a connection to the Kafka cluster cannot be established,
    # or if a metadata request to the Kafka cluster needs to be repeated.
    Retry:
        # When a new channel is created, or when an existing channel is reloaded
        # (in case of a just-restarted orderer), the orderer interacts with the
        # Kafka cluster in the following ways:
        # 1. It creates a Kafka producer (writer) for the Kafka partition that
        # corresponds to the channel.
```

```
    # 2. It uses that producer to post a no-op CONNECT message to that
    # partition
    # 3. It creates a Kafka consumer (reader) for that partition.
    # If any of these steps fail, they will be re-attempted every
    # <ShortInterval> for a total of <ShortTotal>, and then every
    # <LongInterval> for a total of <LongTotal> until they succeed.
    # Note that the orderer will be unable to write to or read from a
    # channel until all of the steps above have been completed successfully.
    ShortInterval: 5s
    ShortTotal: 10m
    LongInterval: 5m
    LongTotal: 12h
    # Affects the socket timeouts when waiting for an initial connection, a
    # response, or a transmission. See Config.Net for more info:
    # https://godoc.org/github.com/Shopify/sarama#Config
    NetworkTimeouts:
      DialTimeout: 10s
      ReadTimeout: 10s
      WriteTimeout: 10s
    # Affects the metadata requests when the Kafka cluster is in the middle
    # of a leader election.See Config.Metadata for more info:
    # https://godoc.org/github.com/Shopify/sarama#Config
    Metadata:
      RetryBackoff: 250ms
      RetryMax: 3
    # What to do if posting a message to the Kafka cluster fails. See
    # Config.Producer for more info:
    # https://godoc.org/github.com/Shopify/sarama#Config
    Producer:
      RetryBackoff: 100ms
      RetryMax: 3
    # What to do if reading from the Kafka cluster fails. See
    # Config.Consumer for more info:
    # https://godoc.org/github.com/Shopify/sarama#Config
    Consumer:
      RetryBackoff: 2s
# Settings to use when creating Kafka topics.  Only applies when
# Kafka.Version is v0.10.1.0 or higher
Topic:
    # The number of Kafka brokers across which to replicate the topic
    ReplicationFactor: 3
# Verbose: Enable logging for interactions with the Kafka cluster.
Verbose: false

# TLS: TLS settings for the orderer's connection to the Kafka cluster.
TLS:

  # Enabled: Use TLS when connecting to the Kafka cluster.
  Enabled: false

  # PrivateKey: PEM-encoded private key the orderer will use for
  # authentication.
  PrivateKey:
    # As an alternative to specifying the PrivateKey here, uncomment the
```

```
      # following "File" key and specify the file name from which to load the
      # value of PrivateKey.
      #File: path/to/PrivateKey

    # Certificate: PEM-encoded signed public key certificate the orderer will
    # use for authentication.
    Certificate:
      # As an alternative to specifying the Certificate here, uncomment the
      # following "File" key and specify the file name from which to load the
      # value of Certificate.
      #File: path/to/Certificate

    # RootCAs: PEM-encoded trusted root certificates used to validate
    # certificates from the Kafka cluster.
    RootCAs:
      # As an alternative to specifying the RootCAs here, uncomment the
      # following "File" key and specify the file name from which to load the
      # value of RootCAs.
      #File: path/to/RootCAs

  # SASLPlain: Settings for using SASL/PLAIN authentication with Kafka brokers
  SASLPlain:
    # Enabled: Use SASL/PLAIN to authenticate with Kafka brokers
    Enabled: false
    # User: Required when Enabled is set to true
    User:
    # Password: Required when Enabled is set to true
    Password:

  # Kafka protocol version used to communicate with the Kafka cluster brokers
  # (defaults to 0.10.2.0 if not specified)
  Version:

################################################################################
#
#   Debug Configuration
#
#   - This controls the debugging options for the orderer
#
################################################################################
Debug:

  # BroadcastTraceDir when set will cause each request to the Broadcast service
  # for this orderer to be written to a file in this directory
  BroadcastTraceDir:

  # DeliverTraceDir when set will cause each request to the Deliver service
  # for this orderer to be written to a file in this directory
  DeliverTraceDir:

################################################################################
```

```
#
#   Operations Configuration
#
#   - This configures the operations server endpoint for the orderer
#
################################################################################
#########
Operations:
    # host and port for the operations server
    ListenAddress: 127.0.0.1:8443

    # TLS configuration for the operations endpoint
    TLS:
        # TLS enabled
        Enabled: false

        # Certificate is the location of the PEM encoded TLS certificate
        Certificate:

        # PrivateKey points to the location of the PEM-encoded key
        PrivateKey:

        # Most operations service endpoints require client authentication when TLS
        # is enabled. ClientAuthRequired requires client certificate authentication
        # at the TLS layer to access all resources.
        ClientAuthRequired: false

        # Paths to PEM encoded ca certificates to trust for client authentication
        ClientRootCAs: []

################################################################################
#########
#
#   Metrics  Configuration
#
#   - This configures metrics collection for the orderer
#
################################################################################
#########
Metrics:
    # The metrics provider is one of statsd, prometheus, or disabled
    Provider: disabled

    # The statsd configuration
    Statsd:
      # network type: tcp or udp
      Network: udp

      # the statsd server address
      Address: 127.0.0.1:8125

      # The interval at which locally cached counters and gauges are pushed
      # to statsd; timings are pushed immediately
      WriteInterval: 30s
```

```
    # The prefix is prepended to all emitted statsd metrics
    Prefix:

################################################################################
#########
#
#   Consensus Configuration
#
#   - This section contains config options for a consensus plugin. It is opaque
#     to orderer, and completely up to consensus implementation to make use of.
#
################################################################################
#########
Consensus:
    # The allowed key-value pairs here depend on consensus plugin. For etcd/raft,
    # we use following options:

    # WALDir specifies the location at which Write Ahead Logs for etcd/raft are
    # stored. Each channel will have its own subdir named after channel ID.
    WALDir: /var/hyperledger/production/orderer/etcdraft/wal

    # SnapDir specifies the location at which snapshots for etcd/raft are
    # stored. Each channel will have its own subdir named after channel ID.
    SnapDir: /var/hyperledger/production/orderer/etcdraft/snapshot
```