PassQueen
a.k.a dumpsqueen.com

BTA Certified Blockchain Developer – Hyperledger

Blockchain CBDH

Version Demo

Total Demo Questions: 22

Total Premium Questions: 226

Buy Premium PDF

https://passqueen.com

support@passqueen.com

QUESTION NO: 1

What is the purpose of a configuration block in a Hyperledger Fabric channel?

A. To record the channel configuration

B. To store private chaincode source code

C. To hold peer world-state values

D. To issue enrollment certificates

ANSWER: A

Explanation:

To record the channel configuration is correct because a configuration block contains the channel configuration that defines organizations, MSPs, policies, capabilities, ordering-service settings, and other channel-level settings. Peers use this configuration when processing channel transactions and validating configuration changes.

Configuration blocks are part of the channel ledger and are created when a channel is formed or when a valid channel configuration update is committed. They are not used to store ordinary application asset data. See the Hyperledger Fabric channel configuration documentation.

QUESTION NO: 2

In Hyperledger Composer resources are declared which three ways? (Choose three.)

A. Assets, orderers, transactions, and events

B. Concepts

C. Assets, participants, transactions, and events

D. Collections

E. Non Enumerated types

F. Enumerated types

ANSWER: B C F

Explanation:

Hyperledger Composer business-network models use Composer Modeling Language files, conventionally stored with the `.cto` extension, to declare the domain model. The model supports declarations for the operational resources of a business network: assets, participants, transactions, and events. Assets represent things of value or interest that are tracked on the blockchain; participants represent people, organizations, or systems; transactions describe requested operations; and events communicate that a significant business occurrence has taken place.

The same modeling language also supports concepts. A concept defines a reusable structured type that can be embedded within another declaration, enabling related properties to be grouped into a meaningful business structure. Enumerated types are likewise declared in model files when a property must be restricted to a fixed set of named values, such as lifecycle states or classifications. Together, these declaration categories provide the principal modeling constructs used to describe a Composer business network. The Composer documentation's model-language reference lists asset, participant, transaction, event, concept, and enum declarations and shows their syntax and intended use. See the Composer Modeling Language reference and the Composer business network documentation.

QUESTION NO: 3

The ledger system in Hyperledger Fabric uses what database by default?Â

A. CouchDB

B. LevelDB

C. MySQL

D. MS SQL

E. PostGres SQL

ANSWER: B

Explanation:

LevelDB is the default database used by a Hyperledger Fabric peer for its world state, which is the current value of all ledger data represented as key-value pairs. It is an embedded key-value database that runs within the peer process, so it does not require deploying or administering a separate database service. This default makes Fabric straightforward to set up while providing efficient reads and writes for chaincode data accessed by key. Fabric's ledger consists of both an immutable blockchain history and the world state; LevelDB specifically provides the default world-state database implementation. A peer can instead be configured to use CouchDB when an application needs rich JSON queries, but that configuration is optional rather than the default. The Fabric documentation identifies LevelDB as the default state database and describes CouchDB as an alternative external database for JSON document queries. See the Hyperledger Fabric ledger documentation and the CouchDB as a state database documentation for the supported state-database choices and their usage.

QUESTION NO: 4

Please review the chaincode below and select what "import" is specifying.

A. Menu Choices

B. Installation requirements

C. Dependencies

D. Logging requests

ANSWER: C

Explanation:

Dependencies is correct. In Go chaincode, an `import` declaration identifies the packages whose exported identifiers the source file needs to use. A chaincode file commonly imports standard-library packages, such as `fmt` or `encoding/json`, along with Hyperledger Fabric libraries such as the Fabric Contract API. These imported packages provide the functionality required by the contract implementation, including formatting, JSON serialization, transaction context handling, contract definitions, and ledger interaction APIs.

Go treats imported packages as dependencies of the source file: the compiler resolves them during the build, and module tooling records and retrieves the required external modules and their versions where applicable. In Hyperledger Fabric, chaincode is packaged and built with the libraries referenced by its source and module metadata, so declaring imports is the direct source-level mechanism that makes the required package APIs available to the chaincode. The Go language specification defines import declarations as declaring package dependencies, and Fabric's chaincode documentation shows Go smart-contract implementations importing Fabric Contract API packages. See the Go specification on import declarations and the Hyperledger Fabric Contract API documentation.

QUESTION NO: 5

What component of Hyperledger Composer captures the core data in a business network including the business model, transaction logic, and access controls?

A. Business Network Adapter

B. Business Network Interface

C. Business Network Card

D. Business Network Archive

E. Business Network API

ANSWER: D

Explanation:

Business Network Archive is correct because it is the deployable package that contains the complete definition of a Hyperledger Composer business network. A Business Network Archive, conventionally distributed as a `.bna` file, bundles the model files that define assets, participants, transactions, and events; script files containing transaction processor logic; and access-control files that specify which participants can perform particular actions on resources. It can also include optional query definitions and package metadata. This packaging makes the archive the portable, versioned unit used to install and deploy a business network to a Composer runtime connected to Hyperledger Fabric.

In practice, developers create or update the business network definition, package those artifacts into the archive, and deploy that archive through Composer tooling. The resulting runtime then applies the supplied business model, invokes the transaction logic, and enforces the included access rules. Hyperledger Composer documentation describes the Business Network Archive as the archive containing the definitions for a business network, including models, transaction processor functions, and access control rules. See the Hyperledger Composer business network documentation and the Business Network Definition reference.

QUESTION NO: 6

The use of cryptographic hashing with blockchain provides for which of the following?

A. Providing for flexibility in security design

B. Providing for ease of analytical insight

C. Ensuring data blocks are mutable

D. Ensuring data blocks are immutable

ANSWER: D

Explanation:

Ensuring data blocks are immutable is correct because cryptographic hashes create a tamper-evident chain of records. Each block includes a hash calculated from its contents and typically includes the hash of the preceding block. As a result, modifying a transaction or other data in an earlier block changes that block's hash. The changed hash no longer matches the value recorded by its successor, and the mismatch propagates through subsequent blocks. Network peers can therefore detect that the ledger history has been altered.

In a distributed blockchain network, preserving an altered history would also require producing a replacement sequence of valid blocks and satisfying the network's ordering or consensus requirements. This makes previously committed data practically immutable: it is not merely protected by access controls, but cryptographically linked so that unauthorized changes are evident and infeasible to reconcile with the accepted ledger. Hyperledger Fabric similarly describes its blockchain as a hash-linked sequence of blocks, where each block references the prior block's hash. See the Hyperledger Fabric blockchain documentation and NIST's overview of cryptographic hash functions.

QUESTION NO: 7

Implicit private data collections are available for each organization that is a member of a channel.

Which two statements about an implicit organization-specific collection are correct? (Select two.)

A. Its collection name uses the organization's MSP ID.

B. Only peers from that organization store the private data.

C. Every channel peer stores the private data.

D. It requires a collection-definition file before it can be used.

E. It can be accessed by organizations that are not channel members.

ANSWER: A B

Explanation:

Its collection name uses the organization's MSP ID is correct. An implicit collection is named using the pattern `_implicit_org_<MSP_ID>`, allowing chaincode to identify the organization-specific collection.

Only peers from that organization store the private data is also correct. The collection is associated with one organization, so its eligible peers can receive and retain the private data. Other channel organizations do not receive the private value through the collection, although the channel ledger can contain hashes that support validation.

Implicit collections do not require a collection-definition file, unlike explicitly defined private data collections. See the Hyperledger Fabric private data architecture documentation and the private data documentation.

QUESTION NO: 8

Chaincode in Hyperledger Fabric is a decentralized transactional program which is running on the validating nodes. Chaincode implements the Chaincode interface in particular, Init and Invoke functions.

Which two statements about Chaincode is correct? (Select two.)

A. Init is called during instantiate transaction after the chaincode container has been established for the first time, allowing the chaincode to initialize its internal data

B. Invoke is called to update or query the ledger after a proposal transaction. Update state variables are committed to the ledger before the transaction is committed

C. Init is called during Instantiate transaction after the chaincode ledger has been established for the first time, allowing the chaincode to initialize its internal data

D. Invoke is called to update or query the ledger in a proposal transaction. Updated state variables are not committed to the ledger until the transaction is committed.

ANSWER: A D

Explanation:

"Init is called during instantiate transaction after the chaincode container has been established for the first time, allowing the chaincode to initialize its internal data" is correct in the Fabric chaincode lifecycle model reflected by the question. The `Init` function is invoked when a chaincode is instantiated and provides the initialization entry point for application state, such as creating initial assets or setting configuration values. Fabric's Go chaincode guidance describes `Init` as the initialization function invoked during chaincode instantiation.

"Invoke is called to update or query the ledger in a proposal transaction. Updated state variables are not committed to the ledger until the transaction is committed." is also correct. During proposal simulation, chaincode execution produces a response together with a read-write set. State changes are only proposed at that stage; they become durable ledger and world-state updates only after endorsement, ordering, validation, and commit processing. This separation supports Fabric's endorsement and validation model and prevents simulation from directly changing ledger state. See the Hyperledger Fabric chaincode documentation and the peer transaction-flow documentation.

QUESTION NO: 9

The gossip data dissemination protocol performs which three functions? (Choose three.)

A. Manages peer discovery and channel membership

B. Disseminates ledger data across all peers on the channel

C. Manages channel membership only

D. Sync ledger state across all peers on any channel

E. Sync ledger state across all peers on the channel

F. Manages peer discovery only

ANSWER: A B E

Explanation:

Hyperledger Fabric's gossip protocol is the peer-to-peer communication layer used by peers within a channel. It supports peer discovery and maintains awareness of eligible peers and their channel membership, allowing channel-scoped communication to be formed and maintained without requiring every peer to communicate directly with every other peer. Gossip also disseminates ledger-related information, including blocks received from the ordering service, throughout the peers participating in that channel. This distribution is resilient because peers relay data to other peers rather than depending on a single delivery path.

In addition, gossip enables ledger-state synchronization for peers on the channel. A peer that has fallen behind, or that joins and needs to catch up, can use gossip-based communication to obtain missing blocks and align its local ledger with the channel's current ledger. These functions are deliberately channel-scoped: channel membership defines which peers may participate in the relevant gossip communication and ledger exchange. Hyperledger Fabric documents gossip as handling peer discovery, channel membership communication, block dissemination, and ledger synchronization. See the Hyperledger Fabric Gossip Data Dissemination documentation and the Fabric peers documentation.

QUESTION NO: 10

A peer commits a block that contains both valid and invalid transactions.

What happens to the invalid transactions? (Select two.)

A. They remain recorded in the blockchain.

B. They do not update the world state.

C. They are removed from the committed block.

D. They are automatically resubmitted by the orderer.

E. They update the world state with a pending status.

ANSWER: A B

Explanation:

They remain recorded in the blockchain is correct. A committed Fabric block preserves the ordered transaction records, including transactions that fail validation. Validation codes in block metadata identify whether a transaction is valid or invalid.

They do not update the world state is also correct. Only valid transactions have their write sets applied to the peer's world state. An invalid transaction remains visible for audit purposes but cannot alter the current ledger state.

This design provides an immutable transaction history while preventing transactions that fail endorsement-policy, MVCC, or other validation checks from changing business data. See the Hyperledger Fabric ledger documentation and transaction flow documentation.

QUESTION NO: 11

Blockchain services consists of three major components.

What are they? (Select three.)

A. Consensus Manager

B. Distributed Ledger

C. Peer to Peer Protocol

D. Reputation Manager

E. Membership Services

ANSWER: A B C

Explanation:

The core components are **Consensus Manager**, **Distributed Ledger**, and **Peer to Peer Protocol**. A distributed ledger provides the shared, tamper-evident record of transactions and current state that is replicated among participating nodes. In Hyperledger Fabric, the ledger consists of an immutable blockchain transaction log and a world-state database, allowing network participants to validate history and query current data consistently.

A peer-to-peer protocol provides the communication layer through which nodes exchange transactions, blocks, and state-related information without relying on a single central server. Fabric's gossip protocol is an example: it distributes ledger and channel data among eligible peers while supporting resilient, decentralized dissemination.

A consensus manager represents the mechanism responsible for reaching agreement on the transaction/block ordering and ensuring that participants process records in a consistent sequence. In Fabric, this responsibility is performed by the ordering service, whose consensus implementation establishes a definite order for transactions before blocks are distributed to peers. Together, these functions—recording shared data, communicating it across nodes, and agreeing on its order—form the essential service capabilities of a blockchain network. See the Hyperledger Fabric ledger documentation and the Fabric gossip protocol documentation.

QUESTION NO: 12

Exhibit.

```
func (t *BTAAsset) Init(stub shim.ChaincodeStubInterface) peer.Response {
  // Get the args from the transaction proposal
  args := stub.GetStringArgs()
  if len(args) != 3 {
    return shim.Error("Incorrect arguments. Expecting a key and a value")
  }

  err := stub.PutState(args[0], []byte(args[1]))
  if err != nil {
    return shim.Error(fmt.Sprintf("Failed to create asset: %s", args[0]))
  }
  return shim.Success(nil)
}
```

The function displayed is called

A. On the instantiation of the chaincode

B. On the 3rd call to the chaincode

C. On every invocation of the chaincode

D. On the teardown of the chaincode

ANSWER: A

Explanation:

The function displayed is called **on the instantiation of the chaincode**. In Hyperledger Fabric chaincode, the `Init` entry point is intended to perform initialization work for a chaincode instance, such as establishing initial ledger state, validating initialization arguments, or creating initial assets. Under the original Fabric chaincode lifecycle, Fabric invokes this function as part of the instantiate transaction; it may also be invoked during an upgrade transaction because an upgrade creates a new chaincode definition and can require initialization of the upgraded version. The initialization transaction is endorsed, ordered, validated, and committed like other ledger-affecting transactions, so any successful writes made by the initialization logic become part of the channel ledger. This differs from ordinary business operations, which are handled through the chaincode invocation entry point after deployment. Fabric's chaincode documentation describes the initialization and invocation entry points and their role in processing transactions. See Hyperledger Fabric: Chaincode for Developers and Hyperledger Fabric: Chaincode lifecycle.

QUESTION NO: 13

Which Hyperledger Fabric ordering mechanism is recommended for production use?

A. BFT

B. Kafka

C. SBFT

D. SOLO

E. Raft

ANSWER: E

Explanation:

Raft is the recommended ordering mechanism for production Hyperledger Fabric networks. It is Fabric's current built-in crash fault-tolerant ordering service and uses a leader-and-follower consensus model: one ordering node acts as leader to sequence transactions into blocks, while follower nodes replicate the ordered log. This provides availability when ordering nodes fail, provided that a majority of the ordering-node cluster remains operational. Raft is designed to be simpler to deploy and operate than the legacy external ordering approach, while allowing organizations to run multiple orderers across separate infrastructure locations. Fabric documentation describes Raft as the recommended choice for production use and explains that its ordering service is based on the etcd Raft protocol. Production deployments should size the ordering cluster with an odd number of nodes, commonly three or five, so it can maintain a majority quorum during failures. See the Hyperledger Fabric ordering service documentation and the Raft configuration guide.

QUESTION NO: 14

The fastest way to test your chaincode is to use the REST interface on your peers. There are several REST endpoints you can test and interact with chaincode. (Select two.)

A. /registrar

B. /register

C. /chaincode

D. /blockchain

E. /BaaS

ANSWER: A C

Explanation:

/registrar and **/chaincode** are the applicable REST resources in the legacy Hyperledger Fabric peer REST API described by this question. The **/registrar** resource supports enrollment and registration-related operations, allowing a test client to establish the user identity context needed before submitting transactions. The **/chaincode** resource is the peer-facing endpoint used to deploy, invoke, and query chaincode. Together, these endpoints provide the essential flow for quickly exercising a chaincode application through HTTP: obtain or use an enrolled identity, then send a chaincode request to the peer for execution or query processing. This REST interface belongs to early Fabric releases and is not the architecture used by modern Fabric applications, which normally use Fabric Gateway client APIs and connection profiles. Nevertheless, for the legacy REST API context explicitly stated in the question, these are the documented resources for registrar functions and chaincode interaction. See the historical Fabric REST API documentation at Hyperledger Fabric REST API and the current application-development guidance at Fabric Gateway documentation.

QUESTION NO: 15

In Hyperledger not all Nodes are created equal. What are the three distinct types of nodes? (Select three.)

A. MSP Nodes

B. Ordering Nodes

C. Channel Node

D. Client Nodes

E. Peer Nodes

F. Endorser Node

ANSWER: B D E

Explanation:

Hyperledger Fabric's network architecture centers on client applications, peer nodes, and ordering nodes. Client applications, represented here by "Client Nodes," use the Fabric SDK or Gateway APIs to submit transaction proposals, collect required endorsements, and submit endorsed transactions to the ordering service. They act on behalf of users and identities, rather than maintaining the ledger themselves. Peer nodes host ledgers for their channels, execute smart contracts when required for endorsement, validate ordered transactions against endorsement policies and versioning rules, and commit valid blocks to the ledger. "Ordering Nodes" are the nodes of the ordering service: they establish a deterministic transaction order, package transactions into blocks, and distribute blocks to channel peers. This separation of responsibilities is fundamental to Fabric's execute-order-validate model. A peer may be assigned an endorsing role for a chaincode, but endorsement is a peer capability rather than a separate, distinct network-node category. Fabric's architecture documentation describes peers, ordering service nodes, and client applications as the principal participating components. See the Hyperledger Fabric network documentation and the peer-node documentation.

QUESTION NO: 16

Reviewing the code below what are the two possible outcomes of the functions? (Select two.)

```
// Invoke is called per transaction on the chaincode. Each transaction is
// either a 'get' or a 'set' on the asset created by Init function. The Set
// method may create a new asset by specifying a new key-value pair.
func (t *SimpleAsset) Invoke(stub shim.ChaincodeStubInterface) peer.Response {
    // Extract the function and args from the transaction proposal
    fn, args := stub.GetFunctionAndParameters()

    var result string
    var err error
    if fn == "set" {
            result, err = set(stub, args)
    } else {
            result, err = get(stub, args)
    }
    if err != nil {
            return shim.Error(err.Error())
    }

    // Return the result as success payload
    return shim.Success([]byte(result))
}
```

A. Success

B. Shim.Error

C. Shim.Success

D. Error

ANSWER: B C

Explanation:

Shim.Error and **Shim.Success** are the two possible outcomes because Hyperledger Fabric Go chaincode functions return a `pb.Response` object through the Chaincode Shim API. A successful execution path returns `shim.Success(payload)`, which constructs a response with an HTTP-style success status and can include a payload for the transaction proposal response. A failure or validation path returns `shim.Error(message)`, which constructs an error response containing a descriptive message. These response values allow the peer and client application to distinguish a completed chaincode invocation from one that was rejected by the chaincode's business logic or encountered an application-level problem. In normal Go chaincode implementations, the `Init` and `Invoke` methods conventionally use these helpers as their explicit return values, including when dispatching to transaction functions. The Fabric Chaincode Shim API documents both helpers and their response semantics: Fabric Go Chaincode Shim package. Fabric's chaincode documentation also describes returning success or error responses from chaincode methods: Hyperledger Fabric chaincode for developers.

QUESTION NO: 17

The CA (Certificate Authority) in Hyperledger Fabric issues the certificates. These certificates are used for identity validation and for transmission of encrypted data that only the owner (person, organization or software) of a specific certificate is able to decrypt and read.

What types of certificates are issued by the CA?

A. tcert

B. ecert

C. rootcert

ANSWER: A B C

Explanation:

In the terminology used by this question, **ecert**, **tcert**, and **rootcert** are certificate types associated with the Fabric certificate-authority and membership-service model. An ecert, or enrollment certificate, is the long-lived X.509 identity certificate obtained during enrollment. It binds a public key to a Fabric identity and is used to establish the member's authenticated identity and organizational affiliation. A tcert, or transaction certificate, is a transaction-specific certificate concept used in the earlier Fabric membership-service design to provide distinct credentials for transaction activity. A rootcert is the root CA certificate that establishes the trust anchor for a certificate chain: participants use it to validate certificates issued under that CA. Certificate authorities create and distribute their CA certificate as part of establishing trust, while enrollment and other operational certificates are issued under that trusted authority. In current Fabric deployments, Fabric CA documentation focuses primarily on enrollment certificates and TLS certificates; nevertheless, the listed terms reflect the certificate categories intended by this exam question. See the Fabric CA User's Guide and the Hyperledger Fabric identity documentation.

QUESTION NO: 18

Which tool would you select to that would allow users to measure performance of a specific implementation with predefined use cases?

A. Hyperledger Caliper

B. Hyperledger Explorer

C. Hyperledger Cello

D. Hyperledger Quilt

ANSWER: A

Explanation:

Hyperledger Caliper is the Hyperledger project designed specifically for benchmarking blockchain implementations. It enables users to measure performance against predefined workloads or use cases, called benchmark configurations. A benchmark can define the transactions to submit, transaction rates, duration, network configuration, and the adapters needed to connect to a supported blockchain platform. Caliper then collects and reports performance indicators such as transaction throughput, transaction latency, and resource consumption, allowing implementation results to be evaluated in a repeatable and comparable way.

This focus on configurable workloads and quantitative performance reporting directly matches the need to measure a specific implementation using predefined use cases. Caliper supports multiple distributed-ledger technologies through platform-specific adapters and provides a benchmark framework rather than requiring a user to build performance tests entirely from scratch. Its documentation describes it as a blockchain benchmark tool and explains how workloads and benchmark rounds are used to generate measurements. See the Hyperledger Caliper documentation and the project's official GitHub repository for benchmark configuration and supported-platform details.

QUESTION NO: 19

In Hyperledger not all Nodes are created equal. What are the three distinct types of nodes? (Select three.)

A. MSP Nodes

B. Orderer Nodes

C. Channel Node

D. Client Nodes

E. Peer Nodes

F. Endorser Node

ANSWER: B D E

Explanation:

In Hyperledger Fabric, client nodes, peer nodes, and orderer nodes represent the main functional node categories involved in processing a transaction. Client nodes, typically applications using a Fabric SDK or Gateway, submit transaction proposals and later submit endorsed transactions for ordering. They interact with the network on behalf of users but do not maintain the ledger themselves. Peer nodes host ledgers and smart contracts, execute transaction proposals when configured for endorsement, validate ordered transactions against endorsement and policy requirements, and commit valid blocks to their ledger copies. Orderer nodes form the ordering service: they establish a deterministic transaction order, package transactions into blocks, and distribute those blocks to channel peers. This separation allows Fabric to divide application interaction, transaction execution and validation, and transaction ordering into distinct responsibilities. A peer may have an endorsing role, but endorsement is a role performed by a peer rather than a separate, fourth fundamental node category. Hyperledger Fabric's architecture documentation describes clients, peers, and ordering services as the key components participating in transaction flow. See the Fabric peer documentation and the Fabric ordering service documentation.

QUESTION NO: 20

Transaction Log Data is immutable i.e , once its written to the log it cannot be changed or deleted.

A. TRUE

B. FALSE

ANSWER: A

Explanation:

TRUE is correct. In Hyperledger Fabric, the blockchain portion of a channel ledger is an immutable, append-only transaction log. Transactions are assembled into blocks, and every block is cryptographically linked to the preceding block by including the preceding block's hash. This chaining makes an alteration to historical transaction data evident, because changing a prior block would change its hash and invalidate the links and validation assumptions for subsequent blocks. Peers validate and commit blocks in a defined order, preserving a verifiable record of the transactions accepted by the channel.

Although the current value of an asset may be updated by later valid transactions, those updates do not overwrite or delete the earlier committed transactions in the blockchain log. Instead, Fabric maintains the latest world-state values separately while retaining the complete ledger history. This separation supports auditability and allows authorized participants to query historical transactions and reconstruct how ledger state evolved. Fabric's documentation describes the ledger as containing both the immutable blockchain transaction log and the current world state. See the Hyperledger Fabric ledger documentation and the Hyperledger Fabric blockchain documentation.

QUESTION NO: 21

When reviewing chaincode you see a function called "ChaincodeStubInterface" in the program.

What does this function do?

A. It is used to access the ledger.

B. It is used to access the chaincode interface.

C. It is used to access the ledger and modify the ledger.

D. It is used to stop the chaincode interface.

ANSWER: C

Explanation:

`ChaincodeStubInterface` is the Fabric chaincode stub interface supplied to transaction code, rather than a function itself. It is the chaincode's primary API for interacting with the ledger during a proposal's execution. Through this interface, a transaction can retrieve world-state data, including individual keys and query results, and it can propose ledger updates by writing or deleting key-value state. These writes are captured in the transaction's read-write set and are committed to the ledger only after the transaction is validated and ordered. The interface also provides transaction context such as the transaction ID, creator identity, transient data, and access to the channel context, all of which support correct chaincode behavior. Therefore, "It is used to access the ledger and modify the ledger." is the best answer because it includes both ledger reads and the state-changing operations exposed through the stub. Fabric's chaincode documentation describes the stub as the API used by smart contracts to access ledger state, while the Go API documents methods such as `GetState`, `PutState`, and `DelState`. See the Hyperledger Fabric chaincode documentation and the ChaincodeStubInterface API reference.

QUESTION NO: 22

State data by default is implemented in LevelDB but it can be replaced by way of configuration with CouchDB.

A. FALSE

B. TRUE

ANSWER: B

Explanation:

TRUE is correct. In Hyperledger Fabric, each peer maintains a world state database containing the latest values for ledger keys. LevelDB is the default embedded state database: it runs within the peer's filesystem and requires no separately deployed database service. Fabric also supports CouchDB as an alternative state database. A peer administrator selects CouchDB through the peer configuration, specifically the state-database settings, and provides the CouchDB connection details. Once configured, the peer uses CouchDB to store and retrieve world-state data while the blockchain ledger itself remains managed by the peer. CouchDB is particularly useful when an application needs rich JSON document queries, including queries over fields and indexes, because it provides capabilities beyond LevelDB's key-based access pattern. This choice is made at the peer level, so organizations can configure their peers according to their operational and application-query requirements. Hyperledger Fabric's documentation explicitly identifies LevelDB as the default state database and CouchDB as the supported external alternative. See the Hyperledger Fabric ledger documentation and the CouchDB as the state database guide.