PassQueen
a.k.a dumpsqueen.com

Appian Certified Lead Developer

Appian ACD300

Version Demo

Total Demo Questions: 5

Total Premium Questions: 58

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Application Design	5
Topic 2, Data Modeling	2
Topic 3, User Interface Design	4
Topic 4, Process Model Design	10
Topic 5, Integrations	9
Topic 6, Security	2
Topic 7, Deployment	8
Topic 8, Performance and Scalability	16
Topic 9, Mix Questions	2
Total	58

QUESTION NO: 1

You are planning data-volume testing for a case-management application. Production is expected to contain several years of cases, with substantial variation in the number of attachments, related tasks, and comments per case. Usage peaks occur when many users search and update cases shortly after the start of the business day.

Which three considerations should be included in the testing strategy?

Select three.

A. Populate representative data volumes and distributions for cases, related records, and attachments.

B. Execute concurrent search and update scenarios that reflect the expected peak workload.

C. Test projected future data growth in addition to the expected go-live volume.

D. Use a uniformly small set of cases so that functional test failures are easier to diagnose.

E. Run all performance scenarios with one user to isolate database response times.

F. Copy production data without masking it because it is the most realistic test data.

ANSWER: A B C

Explanation:

The test data should represent the expected production data volume and distribution, including realistic variation in related records and attachments. Testing should also include the expected concurrent search and update workload during peak periods. Finally, the team should test growth beyond the initial go-live volume so that database access patterns, reports, and record queries are evaluated against the anticipated operational lifecycle.

A uniformly small test data set does not expose query selectivity, paging, indexing, or relationship-volume issues that appear in production. Testing only one user at a time omits the concurrency characteristic described in the requirement. Using unmasked production data is not necessary and can create unnecessary security and privacy risk.

QUESTION NO: 2

You are required to configure a connection so that Jira can inform Appian when specific tickets change (using webhook).

Which three required steps will allow you to connect both systems?

A. Create a Web API object and set up the correct security.

B. Configure the connection in Jira specifying the URL and credentials.

C. Create a new API Key and associate a service account.

D. Give the service account system administrator privileges

E. Create an integration object from Appian to Jira to periodically check the ticket status.

ANSWER: A B C

Explanation:

Create a Web API object and set up the correct security is required because a Jira webhook needs an Appian HTTP endpoint that can receive its event payload. A Web API provides that endpoint and defines the HTTP method, request handling, and execution context needed to process the ticket-change notification. Its security must be deliberately configured so that the webhook caller can invoke the endpoint under an appropriately authorized identity.

Configure the connection in Jira specifying the URL and credentials is also required. The Jira webhook configuration identifies the Appian Web API endpoint as its callback URL and determines which Jira events, such as issue updates or

transitions, trigger delivery. Where the Appian endpoint requires authentication, Jira must send the corresponding credentials with the request.

Create a new API Key and associate a service account establishes a secure, non-personal authentication mechanism for the inbound request. The API key is presented to Appian by the calling system, while the associated service account supplies the identity and permissions under which the Web API executes. This supports least-privilege access to the Appian data and processes involved in handling Jira ticket events. See Appian's Web APIs documentation and API Keys documentation.

QUESTION NO: 3

You are designing a process that is anticipated to be executed multiple times a day. This process retrieves data from an external system and then calls various utility processes as needed. The mam process will not use the results of the utility processes, and there are no user forms anywhere.

Which design choice should be used to start the utility processes and minimize the load on the execution engines?

A. Use the Start Process Smart Service to start the utility processes.

B. Start the utility processes via a subprocess synchronously.

C. Use Process Messaging to start the utility process.

D. Start the utility processes via a subprocess asynchronously

ANSWER: C

Explanation:

Use Process Messaging to start the utility processes. This design is appropriate because the parent process does not need a return value from the utility work and has no need to wait for that work to complete. A process message decouples the initiating process from the utility process: the parent can publish the message and continue or finish, while a message event starts the appropriate utility process independently. This avoids maintaining a subprocess relationship solely to launch work whose outcome is not needed by the caller.

Process messaging is particularly well suited to high-volume, event-driven process designs. It supports asynchronous handling of work and helps keep the main process model focused on retrieving data and determining which downstream actions are required. The utility process can be designed to receive the relevant message data and perform its work without retaining the parent process on an execution path while the utility activity runs. This improves scalability and reduces unnecessary execution-engine overhead compared with coupling the processes through a subprocess call. Appian documents process messages as a mechanism for communication between process instances and for initiating work through message events. See the Appian Process Messaging documentation and Appian process model recipes.

QUESTION NO: 4

A purchase request process receives a list of line items. Each line item must be reviewed by the appropriate category buyer. All reviews can occur concurrently, but the purchase request must not move to final approval until every line-item review is complete.

Which process model design is most appropriate?

A. Configure the review activity as a Multiple Node Instance using the line-item list.

B. Assign one review task to the request initiator and require that user to collect all buyer decisions.

C. Start one independent process through Process Messaging for each line item.

D. Loop through the line items sequentially in a single review task.

ANSWER: A

Explanation:

Configure the review activity as a Multiple Node Instance using the list of line items as its input. Appian creates one instance of the review activity for each line item, allowing the category buyers to work concurrently. The process continues only after the configured multiple node instances are complete, which satisfies the final-approval dependency.

A single task assigned to one buyer would not distribute work by line item. Starting independent processes through messaging would decouple the reviews, but it would require additional coordination to determine when every review has completed. A sequential loop would unnecessarily delay independent reviews.

QUESTION NO: 5 - (SIMULATION)

You are reviewing log files that can be accessed in Appian to monitor and troubleshoot platform-based issues.

For each type of log file, match the corresponding Information that it provides. Each description will either be used once, or not at all.

Note: To change your responses, you may deselect your response by clicking the blank space at the top of the selection list.

ANSWER: See the explanation for the answer

Explanation:

Match the logs to the component/activity they report:

Application Server log — Application-server activity, including application/UI requests, server-side errors, and general platform messages.
Data Service log — Data Service activity, particularly record-data synchronization, data-type/data-source processing, and related errors.
Process Execution Engine log — Process-instance execution activity, including node execution, process-engine errors, and asynchronous process behavior.
Search Server log — Search and indexing activity, including indexing failures or search-server errors.

Therefore, select descriptions referring respectively to *application server*, *data synchronization/data service*, *process execution*, and *search indexing*. Do not select descriptions that concern unrelated areas, such as design-object auditing or web-access traffic, unless that specific log type is displayed in the simulation.