

New ACD301 Latest Test Camp Pass Certify | Professional Learning ACD301 Materials: Appian Lead Developer



P.S. Free 2026 Appian ACD301 dumps are available on Google Drive shared by SurePassExams: <https://drive.google.com/open?id=1nKIT3su-hEu-2b19ZIIh4EEHkWkfPCGo>

SurePassExams help you to find real Appian ACD301 exam preparation process in a real environment. If you are a beginner, and if you want to improve your professional skills, SurePassExams Appian ACD301 exam braindumps will help you to achieve your desire step by step. If you have any questions about the exam, SurePassExams the Appian ACD301 will help you to solve them. Within a year, we provide free updates. Please pay more attention to our website.

Appian ACD301 Exam Syllabus Topics:

Topic	Details
Topic 1	<ul style="list-style-type: none">Platform Management: This section of the exam measures skills of Appian System Administrators and covers the ability to manage platform operations such as deploying applications across environments, troubleshooting platform-level issues, configuring environment settings, and understanding platform architecture. Candidates are also expected to know when to involve Appian Support and how to adjust admin console configurations to maintain stability and performance.
Topic 2	<ul style="list-style-type: none">Proactively Design for Scalability and Performance: This section of the exam measures skills of Application Performance Engineers and covers building scalable applications and optimizing Appian components for performance. It includes planning load testing, diagnosing performance issues at the application level, and designing systems that can grow efficiently without sacrificing reliability.
Topic 3	<ul style="list-style-type: none">Extending Appian: This section of the exam measures skills of Integration Specialists and covers building and troubleshooting advanced integrations using connected systems and APIs. Candidates are expected to work with authentication, evaluate plug-ins, develop custom solutions when needed, and utilize document generation options to extend the platform's capabilities.

Topic 4	<ul style="list-style-type: none"> • Application Design and Development: This section of the exam measures skills of Lead Appian Developers and covers the design and development of applications that meet user needs using Appian functionality. It includes designing for consistency, reusability, and collaboration across teams. Emphasis is placed on applying best practices for building multiple, scalable applications in complex environments.
Topic 5	<ul style="list-style-type: none"> • Data Management: This section of the exam measures skills of Data Architects and covers analyzing, designing, and securing data models. Candidates must demonstrate an understanding of how to use Appian's data fabric and manage data migrations. The focus is on ensuring performance in high-volume data environments, solving data-related issues, and implementing advanced database features effectively.

>> ACD301 Latest Test Camp <<

Learning ACD301 Materials, ACD301 Certification Dump

You may be given the Appian ACD301 practice exam results as soon as they have been saved in the software. SurePassExams modified Appian ACD301 exam dumps allow students to learn effectively about the real Appian ACD301 Certification Exam. Appian ACD301 practice exam software allows students to review and refine skills in a preceding test setting.

Appian Lead Developer Sample Questions (Q31-Q36):

NEW QUESTION # 31

You have an active development team (Team A) building enhancements for an application (App X) and are currently using the TEST environment for User Acceptance Testing (UAT).

A separate operations team (Team B) discovers a critical error in the Production instance of App X that they must remediate. However, Team B does not have a hotfix stream for which to accomplish this. The available environments are DEV, TEST, and PROD.

Which risk mitigation effort should both teams employ to ensure Team A's capital project is only minorly interrupted, and Team B's critical fix can be completed and deployed quickly to end users?

- A. Team B must address the changes directly in PROD. As there is no hotfix stream, and DEV and TEST are being utilized for active development, it is best to avoid a conflict of components. Once Team A has completed their enhancements work, Team B can update DEV and TEST accordingly.
- B. Team B must address changes in the TEST environment. These changes can then be tested and deployed directly to PROD. Once the deployment is complete, Team B can then communicate their changes to Team A to ensure they are incorporated as part of the next release.
- **C. Team B must communicate to Team A which component will be addressed in the hotfix to avoid overlap of changes. If overlap exists, the component must be versioned to its PROD state before being remediated and deployed, and then versioned back to its latest development state. If overlap does not exist, the component may be remediated and deployed without any version changes.**
- D. Team A must analyze their current codebase in DEV to merge the hotfix changes into their latest enhancements. Team B is then required to wait for the hotfix to follow regular deployment protocols from DEV to the PROD environment.

Answer: C

Explanation:

Comprehensive and Detailed In-Depth Explanation: As an Appian Lead Developer, managing concurrent development and operations (hotfix) activities across limited environments (DEV, TEST, PROD) requires minimizing disruption to Team A's enhancements while ensuring Team B's critical fix reaches PROD quickly. The scenario highlights no hotfix stream, active UAT in TEST, and a critical PROD issue, necessitating a strategic approach. Let's evaluate each option:

* A. Team B must communicate to Team A which component will be addressed in the hotfix to avoid overlap of changes. If overlap exists, the component must be versioned to its PROD state before being remediated and deployed, and then versioned back to its latest development state. If overlap does not exist, the component may be remediated and deployed without any version changes: This is the best approach. It ensures collaboration between teams to prevent conflicts, leveraging Appian's version control (e.g., object versioning in Appian Designer). Team B identifies the critical component, checks for overlap with Team A's work, and uses versioning to isolate changes. If no overlap exists, the hotfix deploys directly; if overlap occurs, versioning preserves Team A's work, allowing the hotfix to deploy and then reverting the component for Team A's continuation. This minimizes interruption to Team A's UAT, enables rapid PROD deployment, and aligns with Appian's change management best practices.

* B. Team A must analyze their current codebase in DEV to merge the hotfix changes into their latest enhancements. Team B is then

required to wait for the hotfix to follow regular deployment protocols from DEV to the PROD environment: This delays Team B's critical fix, as regular deployment (DEV # TEST # PROD) could take weeks, violating the need for "quick deployment to end users." It also risks introducing Team A's untested enhancements into the hotfix, potentially destabilizing PROD. Appian's documentation discourages mixing development and hotfix workflows, favoring isolated changes for urgent fixes, making this inefficient and risky.

* C. Team B must address changes in the TEST environment. These changes can then be tested and deployed directly to PROD. Once the deployment is complete, Team B can then communicate their changes to Team A to ensure they are incorporated as part of the next release: Using TEST for hotfix development disrupts Team A's UAT, as TEST is already in use for their enhancements. Direct deployment from TEST to PROD skips DEV validation, increasing risk, and doesn't address overlap with Team A's work. Appian's deployment guidelines emphasize separate streams (e.g., hotfix streams) to avoid such conflicts, making this disruptive and unsafe.

* D. Team B must address the changes directly in PROD. As there is no hotfix stream, and DEV and TEST are being utilized for active development, it is best to avoid a conflict of components. Once Team A has completed their enhancements work, Team B can update DEV and TEST accordingly: Making changes directly in PROD is highly discouraged in Appian due to lack of testing, version control, and rollback capabilities, risking further instability. This violates Appian's Production governance and security policies, and delays Team B's updates until Team A finishes, contradicting the need for a "quick deployment." Appian's best practices mandate using lower environments for changes, ruling this out.

Conclusion: Team B communicating with Team A, versioning components if needed, and deploying the hotfix (A) is the risk mitigation effort. It ensures minimal interruption to Team A's work, rapid PROD deployment for Team B's fix, and leverages Appian's versioning for safe, controlled changes-aligning with Lead Developer standards for multi-team coordination.

References:

- * Appian Documentation: "Managing Production Hotfixes" (Versioning and Change Management).
- * Appian Lead Developer Certification: Application Management Module (Hotfix Strategies).
- * Appian Best Practices: "Concurrent Development and Operations" (Minimizing Risk in Limited Environments).

NEW QUESTION # 32

You are selling up a new cloud environment. The customer already has a system of record for its employees and doesn't want to re-create them in Appian. So you are going to implement LDAP authentication.

What are the next steps to configure LDAP authentication?

To answer, move the appropriate steps from the Option list to the Answer List area, and arrange them in the correct order. You may or may not use all the steps.

The screenshot shows an exam interface with two columns: 'Options' and 'Answer List'. The 'Options' column contains five steps for configuring LDAP authentication:

- Enter two parameters: the url of the LDAP server and plaintext credentials.
- Test the LDAP integration and save if it succeeds.
- Navigate to the Admin Console > Authentication > LDAP
- Work with the customer LDAP administrator to obtain the LDAP authentication file. Import the xad file in the Admin Console.
- Enable LDAP and enter the appropriate LDAP parameters, such as the URL of the LDAP server and plaintext credentials.

The 'Answer List' column is currently empty. A watermark 'appian' is visible at the bottom of the interface.

Answer:

Explanation:



Explanation:

* Navigate to the Admin console > Authentication > LDAP. This is the first step, as it allows you to access the settings and options for LDAP authentication in Appian.

* Work with the customer LDAP point of contact to obtain the LDAP authentication xsd. Import the xsd file in the Admin console. This is the second step, as it allows you to define the schema and structure of the LDAP data that will be used for authentication in Appian. You will need to work with the customer LDAP point of contact to obtain the xsd file that matches their LDAP server configuration and data model. You will then need to import the xsd file in the Admin console using the Import Schema button.

* Enable LDAP and enter the LDAP parameters, such as the URL of the LDAP server and plaintext credentials. This is the third step, as it allows you to enable and configure the LDAP authentication in Appian. You will need to check the Enable LDAP checkbox and enter the required parameters, such as the URL of the LDAP server, the plaintext credentials for connecting to the LDAP server, and the base DN for searching for users in the LDAP server.

* Test the LDAP integration and see if it succeeds. This is the fourth and final step, as it allows you to verify and validate that the LDAP authentication is working properly in Appian. You will need to use the Test Connection button to test if Appian can connect to the LDAP server successfully.

You will also need to use the Test User Lookup button to test if Appian can find and authenticate a user from the LDAP server using their username and password.

Configuring LDAP authentication in Appian Cloud allows the platform to leverage an existing employee system of record (e.g., Active Directory) for user authentication, avoiding manual user creation. The process involves a series of steps within the Appian Administration Console, guided by Appian's Security and Authentication documentation. The steps must be executed in a logical order to ensure proper setup and validation.

* Navigate to the Admin Console > Authentication > LDAP: The first step is to access the LDAP configuration section in the Appian Administration Console. This is the entry point for enabling and configuring LDAP authentication, where administrators can define the integration settings. Appian requires this initial navigation to begin the setup process.

* Work with the customer LDAP point-of-contact to obtain the LDAP authentication xsd. Import the xsd file in the Admin Console: The next step involves gathering the LDAP schema definition (xsd file) from the customer's LDAP system (e.g., via their point-of-contact). This file defines the structure of the LDAP directory (e.g., user attributes). Importing it into the Admin Console allows Appian to map these attributes to its user model, a critical step before enabling authentication, as outlined in Appian's LDAP Integration Guide.

* Enable LDAP and enter the appropriate LDAP parameters, such as the URL of the LDAP server and plaintext credentials: After importing the schema, enable LDAP and configure the connection details. This includes specifying the LDAP server URL (e.g., ldap://ldap.example.com) and plaintext credentials (or a secure alternative like LDAPS with certificates). These parameters establish the connection to the customer's LDAP system, a prerequisite for testing, as per Appian's security best practices.

* Test the LDAP integration and save if it succeeds: The final step is to test the configuration to ensure Appian can authenticate against the LDAP server. The Admin Console provides a test option to verify connectivity and user synchronization. If successful, saving the configuration applies the settings, completing the setup. Appian recommends this validation step to avoid misconfigurations, aligning with the iterative testing approach in the documentation.

Unused Option:

* Enter two parameters: the URL of the LDAP server and plaintext credentials: This step is redundant and not used. The equivalent action is covered under "Enable LDAP and enter the appropriate LDAP parameters," which is more comprehensive and includes enabling the feature.

Including both would be duplicative, and Appian's interface consolidates parameter entry with enabling.

Ordering Rationale:

* The sequence follows a logical workflow: navigation to the configuration area, schema import for structure, parameter setup for

connectivity, and testing/saving for validation. This aligns with Appian's step-by-step LDAP setup process, ensuring each step builds on the previous one without requiring backtracking.

* The unused option reflects the question's allowance for not using all steps, indicating flexibility in the process.

References: Appian Documentation - Security and Authentication Guide, Appian Administration Console - LDAP Configuration, Appian Lead Developer Training - Integration Setup.

NEW QUESTION # 33

You are asked to design a case management system for a client. In addition to storing some basic metadata about a case, one of the client's requirements is the ability for users to update a case. The client would like any user in their organization of 500 people to be able to make these updates. The users are all based in the company's headquarters, and there will be frequent cases where users are attempting to edit the same case. The client wants to ensure no information is lost when these edits occur and does not want the solution to burden their process administrators with any additional effort. Which data locking approach should you recommend?

- A. Use the database to implement low-level pessimistic locking.
- B. Design a process report and query to determine who opened the edit form first.
- C. Allow edits without locking the case CDI.
- D. Add an `@Version` annotation to the case CDT to manage the locking.

Answer: D

Explanation:

Comprehensive and Detailed In-Depth Explanation:

The requirement involves a case management system where 500 users may simultaneously edit the same case, with a need to prevent data loss and minimize administrative overhead. Appian's data management and concurrency control strategies are critical here, especially when integrating with an underlying database.

Option C (Add an `@Version` annotation to the case CDT to manage the locking):

This is the recommended approach. In Appian, the `@Version` annotation on a Custom Data Type (CDT) enables optimistic locking, a lightweight concurrency control mechanism. When a user updates a case, Appian checks the version number of the CDT instance. If another user has modified it in the meantime, the update fails, prompting the user to refresh and reapply changes. This prevents data loss without requiring manual intervention by process administrators. Appian's Data Design Guide recommends `@Version` for scenarios with high concurrency (e.g., 500 users) and frequent edits, as it leverages the database's native versioning (e.g., in MySQL or PostgreSQL) and integrates seamlessly with Appian's process models. This aligns with the client's no-burden requirement.

Option A (Allow edits without locking the case CDI):

This is risky. Without locking, simultaneous edits could overwrite each other, leading to data loss—a direct violation of the client's requirement. Appian does not recommend this for collaborative environments.

Option B (Use the database to implement low-level pessimistic locking):

Pessimistic locking (e.g., using `SELECT ... FOR UPDATE` in MySQL) locks the record during the edit process, preventing other users from modifying it until the lock is released. While effective, it can lead to deadlocks or performance bottlenecks with 500 users, especially if edits are frequent. Additionally, managing this at the database level requires custom SQL and increases administrative effort (e.g., monitoring locks), which the client wants to avoid. Appian prefers higher-level solutions like `@Version` over low-level database locking.

Option D (Design a process report and query to determine who opened the edit form first):

This is impractical and inefficient. Building a custom report and query to track form opens adds complexity and administrative overhead. It doesn't inherently prevent data loss and relies on manual resolution, conflicting with the client's requirements. The `@Version` annotation provides a robust, Appian-native solution that balances concurrency, data integrity, and ease of maintenance, making it the best fit.

NEW QUESTION # 34

A customer wants to integrate a CSV file once a day into their Appian application, sent every night at 1:00 AM. The file contains hundreds of thousands of items to be used daily by users as soon as their workday starts at 8:00 AM. Considering the high volume of data to manipulate and the nature of the operation, what is the best technical option to process the requirement?

- A. Create a set of stored procedures to handle the volume and the complexity of the expectations, and call it after each integration.
- B. Use an Appian Process Model, initiated after every integration, to loop on each item and update it to the business requirements.
- C. Build a complex and optimized view (relevant indices, efficient joins, etc.), and use it every time a user needs to use the data.

- D. Process what can be completed easily in a process model after each integration, and complete the most complex tasks using a set of stored procedures.

Answer: A

Explanation:

Comprehensive and Detailed In-Depth Explanation: As an Appian Lead Developer, handling a daily CSV integration with hundreds of thousands of items requires a solution that balances performance, scalability, and Appian's architectural strengths. The timing (1:00 AM integration, 8:00 AM availability) and data volume necessitate efficient processing and minimal runtime overhead. Let's evaluate each option based on Appian's official documentation and best practices:

* A. Use an Appian Process Model, initiated after every integration, to loop on each item and update it to the business requirements: This approach involves parsing the CSV in a process model and using a looping mechanism (e.g., a subprocess or script task with `forEach`) to process each item. While Appian process models are excellent for orchestrating workflows, they are not optimized for high-volume data processing. Looping over hundreds of thousands of records would strain the process engine, leading to timeouts, memory issues, or slow execution-potentially missing the 8:00 AM deadline. Appian's documentation warns against using process models for bulk data operations, recommending database-level processing instead. This is not a viable solution.

* B. Build a complex and optimized view (relevant indices, efficient joins, etc.), and use it every time a user needs to use the data: This suggests loading the CSV into a table and creating an optimized database view (e.g., with indices and joins) for user queries via `queryEntity`. While this improves read performance for users at 8:00 AM, it doesn't address the integration process itself. The question focuses on processing the CSV ("manipulate" and "operation"), not just querying. Building a view assumes the data is already loaded and transformed, leaving the heavy lifting of integration unaddressed. This option is incomplete and misaligned with the requirement's focus on processing efficiency.

* C. Create a set of stored procedures to handle the volume and the complexity of the expectations, and call it after each integration: This is the best choice. Stored procedures, executed in the database, are designed for high-volume data manipulation (e.g., parsing CSV, transforming data, and applying business logic). In this scenario, you can configure an Appian process model to trigger at 1:00 AM (using a timer event) after the CSV is received (e.g., via FTP or Appian's File System utilities), then call a stored procedure via the "Execute Stored Procedure" smart service. The stored procedure can efficiently bulk-load the CSV (e.g., using SQL's BULK INSERT or equivalent), process the data, and update tables-all within the database's optimized environment. This ensures completion by 8:00 AM and aligns with Appian's recommendation to offload complex, large-scale data operations to the database layer, maintaining Appian as the orchestration layer.

* D. Process what can be completed easily in a process model after each integration, and complete the most complex tasks using a set of stored procedures: This hybrid approach splits the workload: simple tasks (e.g., validation) in a process model, and complex tasks (e.g., transformations) in stored procedures. While this leverages Appian's strengths (orchestration) and database efficiency, it adds unnecessary complexity. Managing two layers of processing increases maintenance overhead and risks partial failures (e.g., process model timeouts before stored procedures run). Appian's best practices favor a single, cohesive approach for bulk data integration, making this less efficient than a pure stored procedure solution (C).

Conclusion: Creating a set of stored procedures (C) is the best option. It leverages the database's native capabilities to handle the high volume and complexity of the CSV integration, ensuring fast, reliable processing between 1:00 AM and 8:00 AM. Appian orchestrates the trigger and integration (e.g., via a process model), while the stored procedure performs the heavy lifting-aligning with Appian's performance guidelines for large-scale data operations.

References:

- * Appian Documentation: "Execute Stored Procedure Smart Service" (Process Modeling > Smart Services).
- * Appian Lead Developer Certification: Data Integration Module (Handling Large Data Volumes).
- * Appian Best Practices: "Performance Considerations for Data Integration" (Database vs. Process Model Processing).

NEW QUESTION # 35

Your client's customer management application is finally released to Production. After a few weeks of small enhancements and patches, the client is ready to build their next application. The new application will leverage customer information from the first application to allow the client to launch targeted campaigns for select customers in order to increase sales. As part of the first application, your team had built a section to display key customer information such as their name, address, phone number, how long they have been a customer, etc. A similar section will be needed on the campaign record you are building. One of your developers shows you the new object they are working on for the new application and asks you to review it as they are running into a few issues. What feedback should you give?

- A. Ask the developer to convert the original customer section into a shared object so it can be used by the new application.
- B. Create a duplicate version of that section designed for the campaign record.
- C. Provide guidance to the developer on how to address the issues so that they can proceed with their work.
- D. Point the developer to the relevant areas in the documentation or Appian Community where they can find more information on the issues they are running into.

myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt,
myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt,
myportal.utt.edu.tt, www.stes.tyc.edu.tw, www.stes.tyc.edu.tw, jsfury.com, www.stes.tyc.edu.tw, www.stes.tyc.edu.tw,
chelisschoolconsultancy.com, www.stes.tyc.edu.tw, www.intensedebate.com, Disposable vapes

DOWNLOAD the newest SurePassExams ACD301 PDF dumps from Cloud Storage for free: <https://drive.google.com/open?id=1nKIT3su-hEu-2b19ZIIh4EEHkWkfPCGo>