ACD-301受験トレーリング & ACD-301最新知識

2026年ShikenPASSの最新ACD-301 PDFダンプおよびACD-301試験エンジンの無料共有：https://drive.google.com/open?id=1cVt9odQWFr0WRV9XsDynCwJUNzDVYhxt

誰もが成功を望んでいますが、誰もが勉強に忍耐する強い心を持っているわけではありません。現在Appianのステータスに満足できない場合は、ACD-301の実際の試験が役立ちます。 ACD-301試験問題は、常に最高99%の合格率を誇っています。教材を使用すると、試験準備の時間を節約できます。 ACD-301テストエンジンを選択すると、簡単に認定を取得できます。選択して、ACD-301学習教材を購入し、今すぐ学習を開始してください！ 知識、Appian Certified Lead Developer実績と幸福があなたを待っています！

ShikenPASSを通じて最新のAppianのACD-301試験の問題と解答早めにを持てて、弊社の問題集があればきっと君の強い力になります。

>> ACD-301受験トレーリング <<

ACD-301最新知識 & ACD-301日本語資格取得

今多くのIT技術会社は職員がAppianのACD-301資格認定を持つのを要求します。AppianのACD-301試験に合格するのは必要なことになります。速く試験に合格して資格認証を取得したいなら、我々ShikenPASSのACD-301問題集を使ってみてください。弊社はあなたに相応しくて品質高いACD-301問題集を提供します。また、あなたの持っている問題集は一年間の無料更新を得られています。あなたは十分の時間でACD-301試験を準備することができます。

Appian Certified Lead Developer 認定 ACD-301 試験問題 (Q13-Q18):

質問 # 13
You are on a call with a new client, and their program lead is concerned about how their legacy systems will integrate with Appian. The lead wants to know what authentication methods are supported by Appian. Which three authentication methods are supported?

- A. Active Directory
- B. Biometrics
- C. SAML
- D. OAuth
- E. CAC
- F. API Keys

正解：A、C、D

解説：
Comprehensive and Detailed In-Depth Explanation:
As an Appian Lead Developer, addressing a client's concerns about integrating legacy systems with Appian requires accurately identifying supported authentication methods for system-to-system communication or user access. The question focuses on Appian's integration capabilities, likely for both user authentication (e.g., SSO) and API authentication, as legacy system integration often

involves both. Appian's documentation outlines supported methods in its Connected Systems and security configurations. Let's evaluate each option:
A . API Keys:
API Key authentication involves a static key sent in requests (e.g., via headers). Appian supports this for outbound integrations in Connected Systems (e.g., HTTP Authentication with an API key), allowing legacy systems to authenticate Appian calls. However, it's not a user authentication method for Appian's platform login-it's for system-to-system integration. While supported, it's less common for legacy system SSO or enterprise use cases compared to other options, making it a lower-priority choice here.
B . Biometrics:
Biometrics (e.g., fingerprint, facial recognition) isn't natively supported by Appian for platform authentication or integration. Appian relies on standard enterprise methods (e.g., username/password, SSO), and biometric authentication would require external identity providers or custom clients, not Appian itself. Documentation confirms no direct biometric support, ruling this out as an Appian-supported method.
C . SAML:
Security Assertion Markup Language (SAML) is fully supported by Appian for user authentication via Single Sign-On (SSO). Appian integrates with SAML 2.0 identity providers (e.g., Okta, PingFederate), allowing users to log in using credentials from legacy systems that support SAML-based SSO. This is a key enterprise method, widely used for integrating with existing identity management systems, and explicitly listed in Appian's security configuration options-making it a top choice.
D . CAC:
Common Access Card (CAC) authentication, often used in government contexts with smart cards, isn't natively supported by Appian as a standalone method. While Appian can integrate with CAC via SAML or PKI (Public Key Infrastructure) through an identity provider, it's not a direct Appian authentication option. Documentation mentions smart card support indirectly via SSO configurations, but CAC itself isn't explicitly listed, making it less definitive than other methods.
E . OAuth:
OAuth (specifically OAuth 2.0) is supported by Appian for both outbound integrations (e.g., Authorization Code Grant, Client Credentials) and inbound API authentication (e.g., securing Appian Web APIs). For legacy system integration, Appian can use OAuth to authenticate with APIs (e.g., Google, Salesforce) or allow legacy systems to call Appian services securely. Appian's Connected System framework includes OAuth configuration, making it a versatile, standards-based method highly relevant to the client's needs.
F . Active Directory:
Active Directory (AD) integration via LDAP (Lightweight Directory Access Protocol) is supported for user authentication in Appian. It allows synchronization of users and groups from AD, enabling SSO or direct login with AD credentials. For legacy systems using AD as an identity store, this is a seamless integration method. Appian's documentation confirms LDAP/AD as a core authentication option, widely adopted in enterprise environments-making it a strong fit.
Conclusion: The three supported authentication methods are C (SAML), E (OAuth), and F (Active Directory). These align with Appian's enterprise-grade capabilities for legacy system integration: SAML for SSO, OAuth for API security, and AD for user management. API Keys (A) are supported but less prominent for user authentication, CAC (D) is indirect, and Biometrics (B) isn't supported natively. This selection reassures the client of Appian's flexibility with common legacy authentication standards.
Appian Documentation: "Authentication for Connected Systems" (OAuth, API Keys).
Appian Documentation: "Configuring Authentication" (SAML, LDAP/Active Directory).
Appian Lead Developer Certification: Integration Module (Authentication Methods).

質問 # 14
An Appian application contains an integration used to send a JSON, called at the end of a form submission, returning the created code of the user request as the response. To be able to efficiently follow their case, the user needs to be informed of that code at the end of the process. The JSON contains case fields (such as text, dates, and numeric fields) to a customer's API. What should be your two primary considerations when building this integration?

- A. The size limit of the body needs to be carefully followed to avoid an error.
- B. The request must be a multi-part POST.
- C. A process must be built to retrieve the API response afterwards so that the user experience is not impacted.
- D. A dictionary that matches the expected request body must be manually constructed.

正解：A、D

解説：
Comprehensive and Detailed In-Depth Explanation:
As an Appian Lead Developer, building an integration to send JSON to a customer's API and return a code to the user involves balancing usability, performance, and reliability. The integration is triggered at form submission, and the user must see the response (case code) efficiently. The JSON includes standard fields (text, dates, numbers), and the focus is on primary considerations for the integration itself. Let's evaluate each option based on Appian's official documentation and best practices:

A . A process must be built to retrieve the API response afterwards so that the user experience is not impacted:
This suggests making the integration asynchronous by calling it in a process model (e.g., via a Start Process smart service) and retrieving the response later, avoiding delays in the UI. While this improves user experience for slow APIs (e.g., by showing a "Processing" message), it contradicts the requirement that the user is "informed of that code at the end of the process." Asynchronous processing would delay the code display, requiring additional steps (e.g., a follow-up task), which isn't efficient for this use case. Appian's default integration pattern (synchronous call in an Integration object) is suitable unless latency is a known issue, making this a secondary-not primary-consideration.
B . The request must be a multi-part POST:
A multi-part POST (e.g., multipart/form-data) is used for sending mixed content, like files and text, in a single request. Here, the payload is a JSON containing case fields (text, dates, numbers)-no files are mentioned. Appian's HTTP Connected System and Integration objects default to application/json for JSON payloads via a standard POST, which aligns with REST API norms. Forcing a multi-part POST adds unnecessary complexity and is incompatible with most APIs expecting JSON. Appian documentation confirms this isn't required for JSON-only data, ruling it out as a primary consideration.
C . The size limit of the body needs to be carefully followed to avoid an error:
This is a primary consideration. Appian's Integration object has a payload size limit (approximately 10 MB, though exact limits depend on the environment and API), and exceeding it causes errors (e.g., 413 Payload Too Large). The JSON includes multiple case fields, and while "hundreds of thousands" isn't specified, large datasets could approach this limit. Additionally, the customer's API may impose its own size restrictions (common in REST APIs). Appian Lead Developer training emphasizes validating payload size during design-e.g., testing with maximum expected data-to prevent runtime failures. This ensures reliability and is critical for production success.
D . A dictionary that matches the expected request body must be manually constructed:
This is also a primary consideration. The integration sends a JSON payload to the customer's API, which expects a specific structure (e.g., { "field1": "text", "field2": "date" }). In Appian, the Integration object requires a dictionary (key-value pairs) to construct the JSON body, manually built to match the API's schema. Mismatches (e.g., wrong field names, types) cause errors (e.g., 400 Bad Request) or silent failures. Appian's documentation stresses defining the request body accurately-e.g., mapping form data to a CDT or dictionary-ensuring the API accepts the payload and returns the case code correctly. This is foundational to the integration's functionality.
Conclusion: The two primary considerations are C (size limit of the body) and D (constructing a matching dictionary). These ensure the integration works reliably (C) and meets the API's expectations (D), directly enabling the user to receive the case code at submission end. Size limits prevent technical failures, while the dictionary ensures data integrity-both are critical for a synchronous JSON POST in Appian. Option A could be relevant for performance but isn't primary given the requirement, and B is irrelevant to the scenario.
Appian Documentation: "Integration Object" (Request Body Configuration and Size Limits).
Appian Lead Developer Certification: Integration Module (Building REST API Integrations).
Appian Best Practices: "Designing Reliable Integrations" (Payload Validation and Error Handling).

質問 # 15
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. Build a complex and optimized view (relevant indices, efficient joins, etc.), and use it every time a user needs to use the data.
- B. Use an Appian Process Model, initiated after every integration, to loop on each item and update it to the business requirements.
- C. 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.
- D. Create a set of stored procedures to handle the volume and the complexity of the expectations, and call it after each integration.

正解：D

解説：
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

fn!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 a!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.
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).

質問 # 16
The business database for a large, complex Appian application is to undergo a migration between database technologies, as well as interface and process changes. The project manager asks you to recommend a test strategy. Given the changes, which two items should be included in the test strategy?

- A. Internationalization testing of the Appian platform
- B. Penetration testing of the Appian platform
- C. Tests that ensure users can still successfully log into the platform
- D. Tests for each of the interfaces and process changes
- E. A regression test of all existing system functionality

正解：D、E

解説：
Comprehensive and Detailed In-Depth Explanation:
As an Appian Lead Developer, recommending a test strategy for a large, complex application undergoing a database migration (e.g., from Oracle to PostgreSQL) and interface/process changes requires focusing on ensuring system stability, functionality, and the specific updates. The strategy must address risks tied to the scope-database technology shift, interface modifications, and process updates-while aligning with Appian's testing best practices. Let's evaluate each option:
A . Internationalization testing of the Appian platform:
Internationalization testing verifies that the application supports multiple languages, locales, and formats (e.g., date formats). While valuable for global applications, the scenario doesn't indicate a change in localization requirements tied to the database migration, interfaces, or processes. Appian's platform handles internationalization natively (e.g., via locale settings), and this isn't impacted by database technology or UI/process changes unless explicitly stated. This is out of scope for the given context and not a priority.
B . A regression test of all existing system functionality:
This is a critical inclusion. A database migration between technologies can affect data integrity, queries (e.g., a!queryEntity), and performance due to differences in SQL dialects, indexing, or drivers. Regression testing ensures that all existing functionality-records,

reports, processes, and integrations-works as expected post-migration. Appian Lead Developer documentation mandates regression testing for significant infrastructure changes like this, as unmapped edge cases (e.g., datatype mismatches) could break the application. Given the "large, complex" nature, full-system validation is essential to catch unintended impacts.
C . Penetration testing of the Appian platform:
Penetration testing assesses security vulnerabilities (e.g., injection attacks). While security is important, the changes described-database migration, interface, and process updates-don't inherently alter Appian's security model (e.g., authentication, encryption), which is managed at the platform level. Appian's cloud or on-premise security isn't directly tied to database technology unless new vulnerabilities are introduced (not indicated here). This is a periodic concern, not specific to this migration, making it less relevant than functional validation.
D . Tests for each of the interfaces and process changes:
This is also essential. The project includes explicit "interface and process changes" alongside the migration. Interface updates (e.g., SAIL forms) might rely on new data structures or queries, while process changes (e.g., modified process models) could involve updated nodes or logic. Testing each change ensures these components function correctly with the new database and meet business requirements. Appian's testing guidelines emphasize targeted validation of modified components to confirm they integrate with the migrated data layer, making this a primary focus of the strategy.
E . Tests that ensure users can still successfully log into the platform:
Login testing verifies authentication (e.g., SSO, LDAP), typically managed by Appian's security layer, not the business database. A database migration affects application data, not user authentication, unless the database stores user credentials (uncommon in Appian, which uses separate identity management). While a quick sanity check, it's narrow and subsumed by broader regression testing (B), making it redundant as a standalone item.
Conclusion: The two key items are B (regression test of all existing system functionality) and D (tests for each of the interfaces and process changes). Regression testing (B) ensures the database migration doesn't disrupt the entire application, while targeted testing (D) validates the specific interface and process updates. Together, they cover the full scope-existing stability and new functionality-aligning with Appian's recommended approach for complex migrations and modifications.
Appian Documentation: "Testing Best Practices" (Regression and Component Testing).
Appian Lead Developer Certification: Application Maintenance Module (Database Migration Strategies).
Appian Best Practices: "Managing Large-Scale Changes in Appian" (Test Planning).

質問 # 17
You are reviewing the Engine Performance Logs in Production for a single application that has been live for six months. This application experiences concurrent user activity and has a fairly sustained load during business hours. The client has reported performance issues with the application during business hours.During your investigation, you notice a high Work Queue - Java Work Queue Size value in the logs. You also notice unattended process activities, including timer events and sending notification emails, are taking far longer to execute than normal.The client increased the number of CPU cores prior to the application going live.What is the next recommendation?

- A. Add execution and analytics shards
- B. Optimize slow-performing user interfaces.
- C. Add more application servers.
- D. Add more engine replicas.

正解：D

質問 # 18
.....

て、いつでもどこでもACD-301試験トレントを学習できます。

照明のせいでブレーカーが落ちたとかと藤野谷が笑う、私は全てを飲み尽くすACD-301受験トレーリング勢いで熱々の焼きそばを口にかっ込んだ、ほかの人はあなたの成績に驚いているとき、ひょっとしたら、あなたはよりよい仕事を探しましたかもしれません。

正確的なACD-301受験トレーリング & 資格試験のリーダープロバイダー & 信頼できるACD-301最新知識

頑張ってください、私たちのAppian Certified Lead Developerテスト練習参考書や試験自体にACD-301ついては、いつでもお気軽にお問い合わせください、クライアントは専門のAppian Certified Lead Developer試験問題と回答で試験にうまくパスして、弊社に好評をもたらします。

しかしACD-301関連試験を受験して資格を得ることは自分の技能を高めてよりよく自分の価値を証明する良い方法ですから、選択しなければならならないです。

P.S.ShikenPASSがGoogle Driveで共有している無料の2026 Appian ACD-301ダンプ：https://drive.google.com/open?id=1cVt9odQWFr0WRV9XsDynCwJUNzDVYhxt