KTest

更に上のクオリティ　更に上のサービス

問題集

http://www.ktest.jp

1年で無料進級することに提供する

Exam : **PLAT-DEV-201**

Title : Salesforce Certified Platform Developer

Version : DEMO

1.A company's engineering department is conducting a month-long test on the scalability of an in-house-developed software that requires a cluster of 100 or more servers.
Which of the following models is the best to use?
A. PaaS
B. SaaS
C. BaaS
D. IaaS
Answer: D
Explanation:
Infrastructure as a Service (IaaS) is the best model for the scenario described because it provides on-demand access to compute, storage, and networking resources that are ideal for a scalable server cluster. IaaS allows the engineering team to rent infrastructure resources without having to invest in physical hardware, making it perfect for temporary or fluctuating workloads, such as scalability testing.
Key Characteristics of IaaS:
Full control over the operating systems and applications running on the servers.
Flexible resource allocation to support high scalability.
Ideal for custom software testing where specific server configurations may be needed.
Why not the other options?
PaaS (Platform as a Service): While PaaS is excellent for application development and deployment, it abstracts the infrastructure layer, which would limit the engineering team's control over the cluster's configuration.
SaaS (Software as a Service): SaaS delivers fully managed applications, not infrastructure or testing environments. It’s irrelevant for this use case.
BaaS (Backend as a Service): BaaS is tailored to mobile or web application backend development, providing APIs and pre-built services, not infrastructure for a server cluster.
Platform Developer
Reference: While this question is broader than Salesforce-specific concepts, understandingIaaS vs. PaaSis relevant when working with Salesforce development.
For example:
Salesforce operates as aPaaS (e.g., Force.com platform), allowing developers to build and deploy applications without managing underlying servers.
Testing scalability and performance at an infrastructure level (as in the question) would fall underIaaSconcepts, which Salesforce developers might encounter when integrating external services or infrastructure like AWS, Azure, or Google Cloud.
This foundational knowledge complements your understanding of cloud services in the Salesforce ecosystem.

2.What are three characteristics of change set deployments? Choose 3 answers Sending a change set between two orgs requires a deployment connection.
A. Change sets can deploy custom settings data.
B. Change sets can only be used between related organizations.
C. Deployment is done in a one-way, single transaction.
D. Sending a change set between two orgs requires a deployment connection.
E. Change sets can be used to transfer records.

Answer: B,C,D

Explanation:

Change sets can only be used between related organizations:

Change sets require a relationship between the source and target Salesforce orgs, such as a production org and its sandboxes. You cannot use change sets between unrelated Salesforce orgs.

Reference: Salesforce Change Set Overview

Deployment is done in a one-way, single transaction:

When a change set is deployed, it is applied to the target org as a single transaction. If there are any errors in the deployment, the entire transaction is rolled back.

Reference: Apex Development Guide

Sending a change set between two orgs requires a deployment connection:

Before sending or deploying a change set, a deployment connection must be established between the source and target orgs. This connection is configured in the setup menu under "Deployment Connections."

Reference: Trailhead: Deployment Connections

Incorrect Options:

A. Change sets can deploy custom settings data:

Change sets cannot deploy data, including custom settings data. They only move metadata.

E. Change sets can be used to transfer records:

Records (data) are not supported by change sets; they only facilitate metadata migration.

3.Universal Containers wants to ensure that all new leads created in the system have a valid email address. They have already created a validation rule to enforce this requirement, but want to add an additional layer of validation using automation.

What would be the best solution for this requirement?

A. Submit a REST API Callout with a JSON payload and validate the fields on a third party system

B. Use a before-save Apex trigger on the Lead object to validate the email address and display an error message if it is invalid

C. Use a custom Lightning Web component to make a callout to validate the fields on a third party system.

D. Use an Approval Process to enforce the completion of a valid email address using an outbound message action.

Answer: B

Explanation:

Before-save Apex Trigger:

This is the best solution because abefore-save triggerruns before the record is saved to the database, providing an opportunity to validate or modify the data.

In this case, the Apex trigger can validate the email address using a regular expression or a third-party API call to ensure the email address is valid. If the email is invalid, an error message can be displayed using addError ().

Why not the other options?

A. Submit a REST API Callout with a JSON payload:

REST callouts are more complex and generally not recommended for simple validation tasks like email format validation. Additionally, callouts cannot be performed directly in a before-save trigger.

C. Use a custom Lightning Web Component (LWC):
LWCs are primarily for UI interactions and should not be used to enforce data validations that are server-side requirements.
D. Use an Approval Process:
Approval Processes are for managing the approval flow of records, not for real-time validations. They cannot enforce email validation directly.
Reference: Apex Triggers Documentation
Trigger Context Variables

4.Which statement should be used to allow some of the records in a list of records to be inserted if others fail to be inserted?
A. Database. insert (records, true)
B. insert records
C. insert (records, false)
D. Database. insert (records, false)
Answer: D
Explanation:
Database.insert (records, false):
The Database.insert () method with the allOrNone parameter set to false allows for partial success.
If some records in the list fail due to validation rules, triggers, or other errors, the method will still attempt to insert the remaining valid records.
The false parameter ensures that records that fail will not roll back the transaction for the others.
Why not the other options?
A. Database.insert (records, true):
The true parameter makes the operation transactional (all or none). If any record fails, all records will roll back.
B. insert records:
The insert DML statement behaves like Database.insert (records, true) by default and rolls back all records if any error occurs.
C. insert (records, false):
This syntax is invalid in Apex.
Reference: Apex DML Operations Documentation
Database Methods

5.A developer identifies the following triggers on the Expense _c object:

```
deleteExpense,
applyDefaultsToExpense,
validateExpenseUpdate;
```

The triggers process before delete, before insert, and before update events respectively.
Which two techniques should the developer implement to ensure trigger best practices are followed?
Choose 2 answers
A. Unity all three triggers In a single trigger on the Expense__c object that Includes all events.
B. Unify the before insert and before update triggers and use Flow for the delete action.
C. Create helper classes to execute the appropriate logic when a record is saved.

D. Maintain all three triggers on the Expense __c object, but move the Apex logic out of the trigger definition.

Answer: A,C

Explanation:

A. Unify all three triggers in a single trigger on the Expense__c object that includes all events:
Salesforce best practices recommend having only one trigger per object to avoid redundancy and conflicts.
By combining all the events (before delete, before insert, and before update) into a single trigger, the developer can manage the logic in an organized and maintainable manner.
This also simplifies debugging and ensures that the trigger logic executes in a predictable order.
C. Create helper classes to execute the appropriate logic when a record is saved:
Using helper classes allows for a clean separation of concerns. The trigger becomes a dispatcher that delegates logic to dedicated classes.
For example, you can create methods like applyDefaultsToExpense (), validateExpenseUpdate (), and deleteExpense () in a helper class and invoke them from the trigger.
This improves reusability, readability, and testability of the code.
Why not the other options?
B. Unify the before insert and before update triggers and use Flow for the delete action:
While Flow is a powerful tool, it is not ideal for deleting records or replacing Apex trigger functionality, especially when triggers already exist for other events.
D. Maintain all three triggers on the Expense__c object but move the Apex logic out of the trigger definition:
Maintaining multiple triggers on the same object can lead to conflicts and execution order issues, even if the logic is moved to helper classes.
Reference: Trigger Best Practices
Trigger Design Patterns