**Basic Level Questions**

**1. What is contract testing in software testing?**

Contract testing is a way to **ensure that two services (a consumer and a provider) can communicate correctly**. It verifies that the API responses match what the consumer expects, preventing integration issues.

**Example**: If a front-end app expects a user API to return { "name": "Alice", "age": 25 }, contract testing ensures the API returns this exact format.

**2. How is contract testing different from traditional API testing?**

| **Feature** | **Contract Testing** | **Traditional API Testing** |
| --- | --- | --- |
| Focus | Communication contract between services | API functionality and response correctness |
| Scope | Tests interactions between consumer and provider | Tests provider in isolation |
| Test Dependency | Consumer and provider teams collaborate | Independent testing of the API |
| Purpose | Prevents breaking changes | Ensures API correctness |

**3. What are the key components of a contract in contract testing?**

* **Consumer**: The service that uses the API (e.g., a frontend app).
* **Provider**: The service that provides the API (e.g., a backend service).
* **Contract**: A document (like a JSON file) that specifies the API request and response format.

**4. What is the difference between consumer-driven and provider-driven contract testing?**

* **Consumer-Driven Contract Testing**: The consumer defines the expected API behavior, and the provider must follow it.
* **Provider-Driven Contract Testing**: The provider defines the API structure, and consumers must adapt.
* **Example**: A mobile app (consumer) defines what data it needs from a server (provider), ensuring the server doesn’t accidentally change its API in a way that breaks the app.

**5. How does contract testing help in microservices architecture?**

Microservices communicate using APIs. Contract testing:  
✅ Each service defines a **contract** (expected request and response format).

✅ Providers (services that expose APIs) and consumers (services that call APIs) test against this contract.

✅ This prevents breaking changes when services update.  
✅ Reduces the need for **full end-to-end tests**. Traditional integration tests require all services to be deployed, which can be slow and complex. Contract testing verifies compatibility at the API level without needing full integration.  
✅ Helps services evolve independently **without fear of failure**.

**6. What tools are commonly used for contract testing?**

* **Pact** (Most popular)
* **Spring Cloud Contract** (For Java-based microservices)
* **Contract First** (For OpenAPI and Swagger)
* **Dredd** (For validating OpenAPI specifications)

**7. What is Pact, and how does it work?**

Pact is a **contract testing tool** that:

1. **Consumer records an expected request and response (Pact file).**
2. **Provider verifies that it can satisfy this contract.**
3. **If there’s a mismatch, the contract fails, indicating an issue.**

**8. What is a Pact file, and what does it contain?**

A Pact file is a JSON file that stores:

* **Consumer name** (e.g., "Shopping App")
* **Provider name** (e.g., "Payment Service")
* **Expected API request and response**

**9. What are the benefits of contract testing over end-to-end (E2E) testing?**

✅ **Faster** (No need to spin up multiple services).  
✅ **More reliable** (E2E tests are flaky).  
✅ **Less expensive** (Fewer test dependencies).

**10. Can contract testing be used for GraphQL APIs?**

Yes! Tools like **Pact** and **GraphQL contract testing** frameworks can verify GraphQL responses match expectations.

**Intermediate Level Questions**

**11. How does Pact ensure compatibility between consumer and provider?**

* Consumer **defines expectations** in a Pact file.
* Provider **runs tests against this file** to ensure compatibility.

**12. What is the role of the Pact broker?**

A Pact broker is a **central repository** that stores contracts. It allows:  
🔹 Sharing of contracts between teams.  
🔹 Tracking contract versions.  
🔹 Verifying provider compliance with multiple consumers.

**13. How do you verify a contract on the provider side?**

* Run **Pact provider tests** against the contract.
* Ensure the provider **returns the expected responses**.
* Fix **mismatches** if any.

**14. How does contract testing handle breaking changes in an API?**

* Detects API changes that **break existing consumers**.
* Provides **early feedback** before deployment.
* Requires **consumer and provider collaboration** to fix issues.

**15. Can contract testing replace integration testing? Why or why not?**

❌ No, contract testing can’t replace all integration tests.  
✅ But it **reduces** the need for **full end-to-end testing**.

**16. How do you manage contract versions in a microservices environment?**

Using **versioning strategies** like:

* Semantic versioning (v1, v2).
* Maintaining **multiple contract versions** in the Pact broker.

**17. What challenges have you faced while implementing contract testing?**

Some common issues:  
⚠️ Resistance from teams unfamiliar with contract testing.  
⚠️ Managing **dynamic data** in contract tests.  
⚠️ Handling **complex provider logic**.

**18. Explain how contract testing can be integrated into a CI/CD pipeline.**

* Consumer runs Pact tests 🡆 Generates Pact file 🡆 Publishes to Pact broker.
* Provider verifies contract against latest Pact file before deployment.

**19. How do you handle dynamic data in contract testing?**

Using **matchers** instead of hardcoded values.  
Example: Instead of "id": 123, use "id": like(ANY\_INTEGER).

**20. What is the difference between Pact and Postman contract testing?**

| **Feature** | **Pact** | **Postman** |
| --- | --- | --- |
| Approach | Consumer-driven | Manual or automated |
| Automation | Fully automated | Semi-automated |

**Advanced Level Questions**

**21. How do you handle multiple consumers in contract testing?**

By maintaining **separate contracts** for each consumer.

**22. What are the best practices for maintaining contract test suites?**

* Keep contracts **small and focused**.
* Update contracts **whenever API changes**.
* Run tests **regularly** in CI/CD.

**23. How do you deal with authentication and authorization in contract testing?**

* Use **mock tokens** instead of real authentication.
* Exclude auth headers from contract tests.

**24. Explain bi-directional contract testing and its significance.**

It ensures both **consumer and provider independently verify** their expectations **without a Pact broker**.

**25. How can contract testing be scaled for large enterprises?**

* Use a **Pact broker** to manage contracts.
* Automate contract testing in **CI/CD**.

**26. What are some limitations of Pact, and how do you overcome them?**

🔹 **Doesn’t test databases** → Use integration tests.  
🔹 **Difficult for public APIs** → Use OpenAPI testing.

**27. How do you handle backward compatibility in contract testing?**

* Keep **old versions** running until all consumers upgrade.
* Test against **multiple contract versions**.

**28. Explain consumer-driven contract testing with an example.**

A mobile app **defines an expected API response**, and the backend team ensures the API meets this expectation.

**29. How do you mock dependencies while writing contract tests?**

* Use **mock servers** for provider tests.
* Use **stubs** to simulate responses.

**30. How would you convince a team to adopt contract testing in an existing project?**

* Show how it **prevents breaking changes**.
* Explain how it **reduces testing time**.
* Demonstrate **real-world success stories**.