You can mention: EC2, S3, EKS, CloudWatch, IAM, etc.

5. Experience with Linux and shell scripting.

6. Databases you have worked on?

MySQL / PostgreSQL

Writing queries

Creating joins

Validating application data

7. Experience with REST APIs and messaging services.

GET, POST, PUT, DELETE

Status codes

Response time

Authentication (JWT, OAuth, Basic)

Messaging services:

Kafka / RabbitMQ (based on your experience)

Validate consumed/published messages

8. Experience in Agile and tools like JIRA.

9. Experience with performance testing.

If yes → Mention JMeter or Locust.

If no → "I have basic understanding but haven't done full-scale performance testing. I can learn quickly."

QualiZeal Interview L1

API

General/basics of api

get billto. adress from json
billTo.adress

```
{"name": "John Smith",
  "sku": "20223",
  "price": 23.95,
  "shipTo": {
    "name": "Jane Smith",
    "address": "123 Maple Street",
    "city": "Pretendville",
    "state": "NY",
    "zip": "12345"
  },
  "billTo": { "name": "John Smith",
    "address": "123 Maple Street",
    "city": "Pretendville",
    "state": "NY",
    "zip": "12345"  }}
```

get billto second adress from json
billTo[1].adress

```
{ "name": "John Smith",
  "sku": "20223",
  "price": 23.95,
  "shipTo": {
    "name": "Jane Smith",
    "address": "123 Maple Street",
    "city": "Pretendville",
    "state": "NY",
    "zip": "12345"},
  "billTo": [{
    "name": "John Smith1",
    "address": "123 Maple Street",
    "city": "Pretendville",
    "state": "NY",
    "zip": "12345 },
  { "name": "John Smith2",
    "address": "124 Maple Street",
    "city": "Pretendville",
    "state": "NY",
    "zip": "12345"}]}
```

AUTOMATION

Framework level questions
Excel utils, dependency (apache-poi)
find a respective check of the given country
https://cosmocode.io/automation-practice-webtable/
//tr//td[normalize-space()='Afghanistan']/preceding-sibling::td//input[@type='checkbox']
general/ basics of automation, selenium

JAVA

string mutable, immutable
==, .equals. if s1==s2?
Exceptions
general/basics of java

L2 (client : SCIF-State Compensation Insurance Fund)

JAVA

Colletions and examples, map insertion order

Collection Type	Usage Example in Automation
List (`ArrayList`, `LinkedList`)	Storing multiple test data values, e.g., multiple usernames or browser tabs.
Set (`HashSet`, `LinkedHashSet`, `TreeSet`)	Storing unique data like unique IDs, removing duplicates.
Map (`HashMap`, `LinkedHashMap`, `TreeMap`)	Storing key-value pairs, e.g., test case ID → status, configuration properties.
Queue / Deque (`PriorityQueue`, `ArrayDeque`)	Managing execution order or storing logs temporarily.

Map and Insertion Order

`HashMap` → Does NOT maintain insertion order.
`LinkedHashMap` → Maintains insertion order (order in which elements are added).
`TreeMap` → Sorts entries based on keys (natural order or custom comparator).

For key-value pairs where order matters, always use `LinkedHashMap`.
For unique unordered data, use `HashMap`.
Collections like `ArrayList` and `HashSet` are common for storing test data in automation frameworks.

string mutable, immutable

String / StringBuilder are classes

StringBuilder/Buffer Mutable Object whose state/content can be changed after creation.
String Immutable Object whose state/content cannot be changed after creation.

== vs .equals. if string1==string2?

`==` Operator
Compares memory references, not the content of objects.
Returns `true` if both references point to the same object.
Works for primitives as a value comparison.

```
String a = new String("hello");
String b = new String("hello");
System.out.println(a == b); // false, different objects in memory
int x = 5, y = 5;
System.out.println(x == y); // true, primitive comparison
```

```
.equals() Method
Compares object contents (logical equality).
Defined in Object class, can be overridden by classes like String,
Integer, etc.
Returns true if contents are equal.
```

```
String a = new String("hello");
String b = new String("hello");
System.out.println(a.equals(b)); // true, content is same
```

Exceptions and handling

Type	Checked / Unchecked	Handled By	Examples	Notes
Checked Exception	Checked (compile-time)	Must handle with `try-catch` or `throws`	`IOException`, `SQLException`, `FileNotFoundException`	Compiler enforces handling.
Unchecked Exception	Unchecked (runtime)	Optional handling	`ArithmeticException`, `NullPointerException`, `ArrayIndexOutOfBoundsException`	Runtime exceptions; handling not enforced by compiler.

Error	Not applicable	Rarely handled	`OutOfMemoryError`, `StackOverflowError`	Severe problems; generally outside program control.

try catch finally, finally can be skipped?
"`try` holds risky code, `catch` handles exceptions, and `finally` executes cleanup code; finally executes always unless the JVM exits or the thread is killed."

abstraction - at what level can achieve this?
Abstraction hides complexity and exposes only relevant functionality.
Achieved via abstract classes and interfaces.
Abstract class → partial abstraction, can have concrete methods.
Interface → full abstraction (hides implementation completely).
Method-level abstraction → hides implementation of specific methods.

Level	How Achieved	Example
Class-Level	Abstract class / Interface	`abstract class Vehicle` / `interface Payment`
Method-Level	Abstract methods / Interface methods	`abstract void start()`

```
abstract class Vehicle {
    abstract void start();  // Method-level abstraction
    void stop() {           // Concrete method
        System.out.println("Vehicle stopped");
    }
}

class Car extends Vehicle {
    @Override
    void start() {
        System.out.println("Car started");
    }
}

public class TestAbstraction {
    public static void main(String[] args) {
        Vehicle myCar = new Car();
        myCar.start();  // Implementation hidden
```

```
        myCar.stop();
    }
}

interface Payment {
    void pay(double amount);  // Method-level abstraction
}

class CreditCardPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Paid " + amount + " using credit card");
    }
}

public class TestPayment {
    public static void main(String[] args) {
        Payment payment = new CreditCardPayment();
        payment.pay(1000); // Implementation hidden
    }
}
```

AUTOMATION

cucumber structure | runner and glue plugins and dependencies

```
src
 └─ test
    ├─ java
    │   └─ stepdefinitions   <-- Step definition classes (glue code)
    │   └─ runners           <-- Test runner classes
    │   └─ hooks             <-- Before/After hooks
    └─ resources
        └─ features          <-- Feature files (Gherkin scenarios)
```

```
import io.cucumber.testng.AbstractTestNGCucumberTests;
import io.cucumber.testng.CucumberOptions;

@CucumberOptions(
    features = "src/test/resources/features",
    glue = {"stepdefinitions", "hooks"},
    plugin = {
        "pretty",
        "html:target/cucumber-report.html",
        "json:target/cucumber.json",
        "rerun:target/rerun.txt"
    }
)
```

```
public class TestRunner extends AbstractTestNGCucumberTests {
}
```

Runner starts execution - Usually a TestNG or JUnit runner.
Feature files are read - Cucumber parses .feature files.
Glue matches steps - Each Gherkin step is mapped to a method in step definitions.
Step execution - Selenium/WebDriver code executes actions and validations.
Hooks run before/after scenarios - For setup/cleanup like browser launch/close.
Reporting - Plugins generate HTML, JSON, or rerun reports.

DB

db connections - by coding wise to establish connection, add dependency?

“In Java Selenium, I establish a database connection using JDBC. I load the DB driver, create a connection using `DriverManager` with URL, username, and password, then execute SQL queries using `Statement` or `PreparedStatement` and validate results against the UI.”

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class DBConnectionTest {

    public static void main(String[] args) throws Exception {

        String dbUrl = "jdbc:postgresql://hostname:5432/dbname"; // Redshift uses similar URL
        String username = "db_user";
        String password = "db_password";

        // Establish connection
        Connection conn = DriverManager.getConnection(dbUrl, username, password);

        // Create statement
        Statement stmt = conn.createStatement();

        // Execute query
        ResultSet rs = stmt.executeQuery("SELECT username FROM users WHERE id=1");

        // Read data
        while (rs.next()) {
            System.out.println(rs.getString("username"));
        }

        // Close connection
```

```
        rs.close();
        stmt.close();
        conn.close();
    }
}
```

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.7.3</version>
</dependency>
```

TESTING TOOLS

Test management tool - what you do as a lead?

“As a Test Lead, I use the test management tool to plan and track testing activities, create and review test cases, and ensure requirement traceability.
I assign tasks, monitor execution progress, and manage defects.
I analyze metrics and generate reports to communicate status and risks.”