NVIDIA SONiC Technical Deep Preparation

1. Python Automation — Deep Dive

A practical guide for designing, implementing, and explaining scalable automation frameworks for SONiC regression testing. The following topics mirror NVIDIA's focus areas for this role: pytest, async programming, test data design, concurrency, and CI/CD integration.

Framework Architecture

Organize tests and utilities into layers — Drivers, API, Tests, and Reporting. Each test imports feature-specific APIs (e.g., VLAN, BGP) implemented using reusable classes. Fixtures handle setup/teardown of DUTs, Redis DBs, and PTF hosts.

pytest Essentials

- Use pytest fixtures for reusable DUT connections. - Parameterize feature tests using YAML or JSON matrices. - Apply pytest-xdist for parallel execution with proper resource locks. - Use pytest markers to isolate L2/L3/QoS categories.

Modular Design & Dependency Injection

All external dependencies (SSH, Redis, TrafficGen) are injected via fixtures instead of hardcoding. Each module maintains clear separation of concerns, and reusable logic is encapsulated in helper classes.

Async Programming & Concurrency

Use asyncio and asyncssh for concurrent operations across hundreds of network ports. Apply semaphores to control concurrency and avoid overloading SONiC's management plane. ThreadPoolExecutors are used for CPU-bound parsing operations.

Performance & Stress Testing

Measure throughput, convergence, and latency under load. Automate ECMP failover tests using Scapy or TRex, monitor container CPU/memory usage, and collect Redis and COUNTERS_DB metrics for trend analysis.

CI/CD Integration

Leverage Jenkins for test orchestration — stages include build, deploy, run, and report. Archive artifacts (logs, JSON, Allure data) and classify failures as Infra/Test/Product automatically. Track KPIs such as test reliability, coverage, and runtime.

2. Networking & SONiC — Deep Dive

Understand SONiC internals, container architecture, Redis DB flow, and how to automate validation across VLANs, routing, and QoS. Demonstrate clear command over L2/L3 protocols and SONiC management tools.

SONiC Architecture Overview

SONiC's modular design runs each service in a container (bgpd, lldpd, syncd, swss). Data flow: CONFIG_DB → APP_DB → ASIC_DB → Hardware → COUNTERS_DB. Validation ensures

consistency across these layers.

L2 Validation

Test VLAN creation, LACP bundling, and LLDP neighbor discovery. Verify VLAN tags and RSTP behavior post link flap. Use pytest fixtures for port-channel checks and Scapy for tagged packet validation.

L3 & Routing

Automate BGP route verification, OSPF adjacency checks, and ECMP hashing tests. Collect Redis entries from APP_DB and ASIC_DB and verify traffic convergence times under link failures.

QoS & Performance Verification

Validate queue scheduling and ECN marking using WRED configurations. Generate controlled congestion and measure ECN-marked packets in COUNTERS_DB. Ensure minimal packet loss and acceptable queue latency.

Tools & Automation

- sonic-mgmt: pytest-based SONiC automation suite. - Ansible: manages testbeds and deploys DUTs. - PTF: sends and verifies packet flow using Scapy. - pytest-splunk: for log correlation and failure analysis.

Troubleshooting Playbooks

Include checklists for packet loss, BGP flaps, LACP instability, and DB inconsistencies. Verify logs in syncd/swss/bgpd containers and monitor ASIC counters during issue reproduction.

Key Takeaway

Master the end-to-end pipeline — from test authoring (pytest) to SONiC internals (Redis flow) to CI/CD dashboards. Be prepared to describe how you’d automate full regression coverage, triage failures, and measure performance across distributed SONiC topologies.