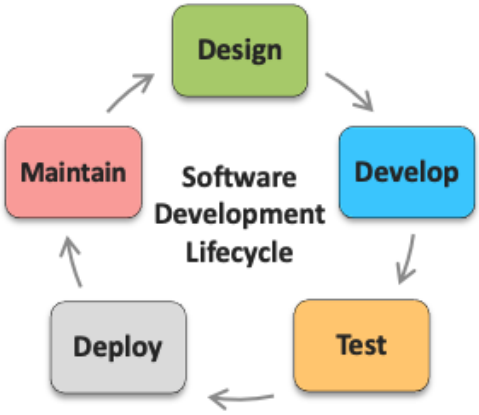
DevNet Expert (v1.1) Exam Topics – Practical Exam

Exam Description:

The Cisco DevNet Expert (v1.1) Practical Exam is an 8-hour, hands-on exam that requires a candidate to plan, design, develop, test, deploy, and maintain software solutions within complex automation-driven network environments.

The following topics are general guidelines for the content that is likely to be included on the exam. Your knowledge, skills, and abilities related to these topics will be tested throughout the entire software development lifecycle, as specified within this document.

The exam is closed book, and no outside reference materials are allowed.



20% 1.0 Software Design, Development, and Deployment

1.1 Design a solution based on an on-premises, hybrid, or public cloud deployment, considering these factors:

- 1.1.a Deployment: maintainability, modularity (e.g., containers, VM, orchestration, automation, components, and infrastructure requirements)
- 1.1.b Reliability: high availability and resiliency
- 1.1.c Performance: scalability, latency, and rate limiting
- 1.1.d Infrastructure: monitoring, observability, and metrics (e.g., instrument placement and instrument deployment)

1.2 Modify an existing network automation solution based on business and technical requirements (includes gap analysis, source of truth)

1.3 Use Git in a CI/CD development workflow

1.4 Troubleshoot issues with a CI/CD pipeline (e.g., code-based failures, pipeline issues, and tool incompatibility)

1.5 Diagnose application performance issues - such as asynchronous request processing, database delays, high memory and CPU utilization, microservice network delays, and asymmetric routing - using network and application tools as well as assurance data.

30% 2.0 Infrastructure as Code

2.1 Build, manage, and operate a Python-based REST API with a web application framework (endpoints, HTTP request and response, OpenAPI specification)

2.2 Build, manage, and operate a Python-based CLI application to use a REST API

2.3 Consume and use a new API, given the documentation
- 2.3.a REST
- 2.3.b GraphQL

2.4 Create a RESTCONF or NETCONF payload based on a given YANG module, and interpret the response

2.5 Create a NETCONF filter by using XPath

2.6 Configure network devices on an existing infrastructure by using NETCONF or RESTCONF, given YANG analysis tools (and driven by a source of truth)

2.7 Create and use a role by utilizing Ansible to manage infrastructure, given support documentation
- 2.7.a Loop control
- 2.7.b Conditionals
- 2.7.c Use of variables and templating
- 2.7.d Use of connection plug-ins such as network CLI, HTTPAPI, and NETCONF

2.8 Use Terraform to statefully manage infrastructure, given support documentation
- 2.8.a Loop control
- 2.8.b Resource graphs
- 2.8.c Use of variables
- 2.8.d Resource retrieval
- 2.8.e Resource provision
- 2.8.f Management of the state of provisioned resources

2.9 Create a basic Cisco NSO service package to meet given business and technical requirements. The service would generate a network configuration on the target device platforms using the "cisco-ios-cli" NED and be of type "python-and-template"
- 2.9.a Create a service template from a provided NSO device configuration
- 2.9.b Create a basic YANG module for the service containers (including lists, leaf lists, data types, leaf references, and single argument "when" and "must" conditions)
- 2.9.c Create basic actions to verify operational status of the service
- 2.9.d Monitor service status by reviewing the NCS Python VM log file

25% **3.0 Network Programmability and Automation**

3.1 Create, modify, and troubleshoot scripts by using Python libraries and SDK documentation to automate against APIs (ACI, AppDynamics, Catalyst Center, FDM, Intersight, IOS XE, Meraki, NSO, Webex)

3.2 Automate the configuration of a Cisco IOS XE network device (based on a provided architecture and configuration), including these components:

- 3.2.a Interfaces
- 3.2.b Static routes
- 3.2.c VLANs
- 3.2.d Access control lists
- 3.2.e BGP peering
- 3.2.f BGP and OSPF routing tables
- 3.2.g BGP and OSPF neighbors

3.3 Modify and troubleshoot an automated test by using pyATS to meet requirements

- 3.3.a Create a testbed file for connecting to Cisco IOS, IOS XE, or NX-OS devices
- 3.3.b Gather current configuration and operational state from devices using the Genie parser and models included with pyATS
- 3.3.c Develop and execute test jobs and scripts using AEtest to verify network health

3.4 Design a model-driven telemetry solution based on given business and technical requirements by using gNMI dial-in, gRPC dial-out, and NETCONF dial-in

3.5 Create YANG model-driven telemetry subscriptions

- 3.5.a Identify model elements and cadence
- 3.5.b On-change or event drive
- 3.5.c Optimize frequency
- 3.5.d Dial-out subscription
- 3.5.e Secure telemetry streams
- 3.5.f Confirm data transmission
- 3.5.g Identify network issues and make changes

10% 4.0 Containers

4.1 Create a Docker image (including Dockerfile)
- 4.1.a From a provided image
- 4.1.b Expose ports
- 4.1.c Add or copy files
- 4.1.d Run commands during image build
- 4.1.e Manipulate entry point and initial commands
- 4.1.f Establish working directories
- 4.1.g Environment variables as part of a definition to control an application
- 4.1.h Docker ignore file
- 4.1.i Volumes

4.2 Package and deploy a solution by using Docker Compose
- 4.2.a Deploy and manage containers
- 4.2.b Define services, networks, volumes, and links

4.3 Package and deploy a solution by using Kubernetes
- 4.3.a Use deployments, secrets, services, ingress, volumes, namespaces, and replicas
- 4.3.b Manage the lifecycle of pods (e.g., scale up, scale down, help status, logs)
- 4.3.c Monitor pods by building health checks)
- 4.3.d Use the kubectl interface

4.4 Create, consume, and troubleshoot a Docker host and bridge-based networks and integrate them with external networks

15% 5.0 Security

5.1 Leverage OWASP secure coding practices into all solutions to meet given requirements
- 5.1.a Input validation
- 5.1.b Authentication and password management
- 5.1.c Access control
- 5.1.d Cryptographic practices
- 5.1.e Error handling and logging
- 5.1.f Communication security

5.2 Create a Certificate Signing Request (CSR) by using OpenSSL; send CSR to a provided Certificate Authority; and use the certificate to secure a web application

5.3 Use OAuth2+ to obtain an authentication token

5.4 Use a secret management system to secure an application

5.5 Use tokens, headers, and secrets to secure a REST API