PassQueen
a.k.a dumpsqueen.com

Implementing DevOps Solutions and Practices using Cisco Platforms (DEVOPS)

Cisco 300-910

Version Demo

Total Demo Questions: 18

Total Premium Questions: 182

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Understanding Infrastructure and Automation	25
Topic 2, CI/CD Pipeline Implementation	44
Topic 3, Application Deployment	42
Topic 4, Infrastructure as Code	22
Topic 5, Platform Security	14
Topic 6, Observability	35
Total	182

QUESTION NO: 1

An application communicates with multiple third-party services that use API keys When stress testing the system, developers must change the URLs of the third-party services The application must run on port 5000. but the developers must be able to change the port at runtime if required Which configuration management approach simplifies the development workflow and reduces the configuration changes?

A. secrets

B. sane defaults

C. application parameters

D. config maps

ANSWER: C

Explanation:

application parameters is correct because it externalizes deployment-specific settings, such as third-party service endpoints and the listening port, from the application code. The application can define a default port of 5000 while accepting an overridden value at runtime, commonly through environment variables, command-line arguments, or a parameter file supplied by the deployment environment. Similarly, stress-testing environments can provide alternate service URLs without requiring developers to edit source files, rebuild the application, or maintain separate code branches for each environment.

This approach supports a clean separation between code and configuration: the same application artifact can run in development, test, stress-test, and production environments with different parameter values. It therefore reduces manual configuration changes and makes environment-specific behavior explicit, repeatable, and automation-friendly. API keys should likewise be supplied externally rather than embedded in code, with the parameter mechanism integrated with an appropriate secure secret store where necessary. This aligns with the Twelve-Factor App principle that configuration likely to vary between deployments should be stored in the environment: The Twelve-Factor App: Config. Kubernetes also documents the use of environment variables and configuration resources to provide runtime configuration to containers: Define Environment Variables for a Container.

QUESTION NO: 2

A three-tier web application must be moved to containers. A webserver is already in place, and the middleware container can talk to a central database server. The hostname of the database server is known, but the name of the middleware server must be provided to the webserver.

In which file should the name of the middleware server be configured?

A. Docker Service discovery daemon

B. Docker Swarm

C. Docker Compose

D. Dynamic Host Configuration Protocol

ANSWER: C

Explanation:

Docker Compose is correct because the application's container-to-container service names are defined in the Compose file, conventionally named `compose.yaml` or `docker-compose.yml`. The middleware tier is declared as a service in that file, and its service name becomes the DNS name that other containers on the same Compose network can use. The webserver configuration can therefore refer to the middleware service by that declared name, rather than relying on an ephemeral container IP address. This provides stable, portable configuration as containers are recreated or scaled, since Docker's internal networking resolves the service name to the appropriate running container or containers. A Compose definition also groups the web and middleware tiers with their required networks, environment settings, ports, and dependencies in a version-controlled application configuration. Docker documents that Compose creates a default network for an application

and that each service is reachable by its service name on that network. See the Docker Compose networking documentation and the Compose services reference.

QUESTION NO: 3 - (DRAG DROP)

Drag and drop the code from the bottom onto the box where the code is missing to create a Terraform configuration that builds the network environment for a multitier software application. More EPG, Contract, and Filter definitions have been removed from the code.

ANSWER:

Explanation:

The completed configuration correctly builds the hierarchy and object references required for this multitier Cisco ACI environment. The application profile declares its display value with the `name` attribute, which supplies the application profile name shown as `multi_app_prod`. The web endpoint group is then associated with that application profile through `application_profile_dn`. Its value is already provided by the application profile resource ID expression on the following line, so this attribute establishes the intended parent-child relationship between the endpoint group and application profile.

The endpoint group's bridge-domain attachment is correctly represented by `aci_bridge_domain.production_bd.id`. This uses the Terraform-generated ID of the bridge-domain resource and assigns the endpoint group to the production bridge domain. In the filter section, `aci_tenant.production_tenant.id` correctly supplies the tenant distinguished-name reference. That makes the filter an object in the production tenant. The filter entry is associated with its parent filter by using `aci_filter.db_traffic.id` for `filter_dn`, ensuring that the user database traffic rule belongs to the database traffic filter.

Finally, `ether_t` is the correct attribute for the value `ip`. Together with the existing TCP protocol and destination port range of 3306, this defines an IP-based database traffic filter entry. These are the expected resource attributes and ID-based references used by the Cisco ACI Terraform provider to connect tenants, application profiles, endpoint groups, bridge domains, filters, and filter entries. The provider documentation for the relevant endpoint-group and filter-entry resources is available at Cisco ACI application EPG resource documentation and Cisco ACI filter entry resource documentation.

QUESTION NO: 4

Configuration changes to the production network devices are performed by a CI/CD pipeline. The code repository and the CI tool are running on separate servers. Some configuration changes are pushed to the code repository, but the pipeline did not start.

Why did the pipeline fail to start?

A. The CI server was not configured as a Git remote for the repository.

B. The webhook call from the code repository did not reach the CI server.

C. Configuration changes must be sent to the pipeline, which then updates the repository.

D. The pipeline must be started manually after the code repository is updated.

ANSWER: B

Explanation:

The webhook call from the code repository did not reach the CI server. In a CI/CD workflow where the repository and CI platform run on separate servers, a push does not inherently cause the CI server to poll for or discover the change immediately. The repository normally sends an HTTP webhook event, such as a push event, to a configured endpoint on the CI server. That event is the trigger that tells the CI system which repository, branch, commit, and event should begin the pipeline.

If the webhook cannot reach the CI server, no event is delivered and therefore no pipeline execution is initiated. Common operational causes include an incorrect webhook URL, DNS or routing failures between servers, firewall or access-control rules blocking the listener port, an unavailable CI service, TLS certificate validation issues, or an authentication secret mismatch. The repository's webhook delivery history and the CI server logs are the appropriate places to verify delivery and diagnose the failure. Jenkins documents that webhooks notify Jenkins of repository changes, and GitHub documents that webhook deliveries are HTTP POST requests to the configured payload URL. See Jenkins GitHub integration documentation and GitHub webhook documentation.

QUESTION NO: 5

Which two practices help make the security of an application a more integral part of the software development lifecycle? (Choose two.)

A. Add a step to the CI/CD pipeline that runs a dynamic code analysis tool during the pipeline execution.

B. Add a step to the CI/CD pipeline that runs a static code analysis tool during the pipeline execution.

C. Use only software modules that are written by the internal team.

D. Add a step to the CI/CD pipeline to modify the release plan so that updated versions of the software are made available more often.

E. Ensure that the code repository server has enabled drive encryption and stores the keys on a Trusted Platform Module or Hardware Security Module.

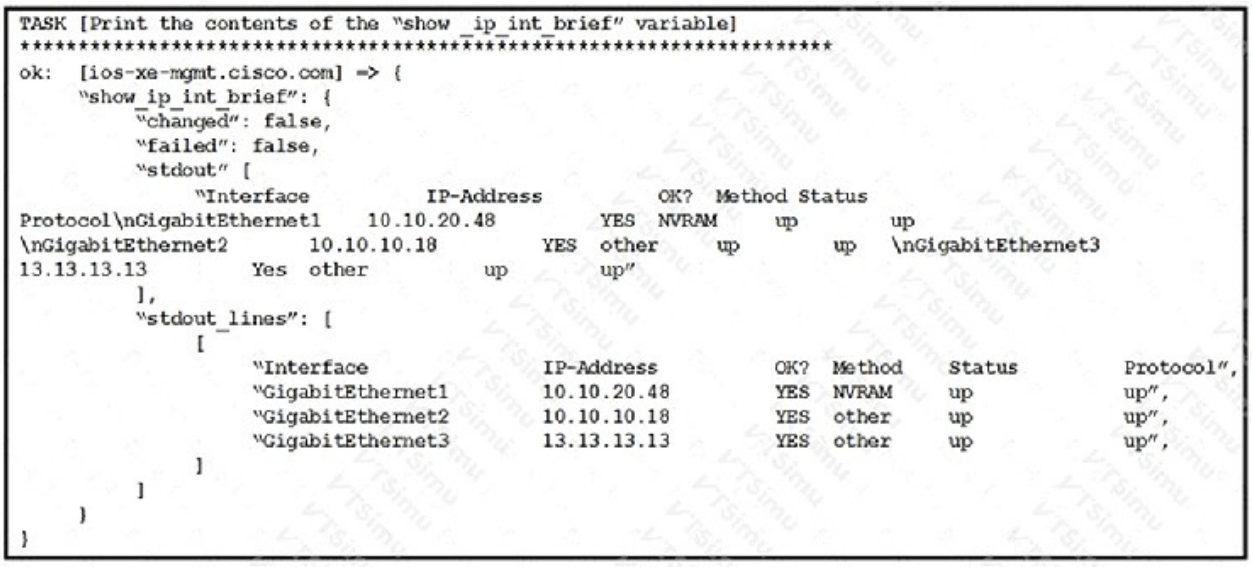
ANSWER: A B

Explanation:

Adding a CI/CD pipeline step that runs a dynamic code analysis tool and adding a CI/CD pipeline step that runs a static code analysis tool both embed repeatable security testing directly into the development and delivery workflow. Static application security testing examines source code, bytecode, or binaries without executing the application. This enables teams to identify insecure coding patterns and vulnerabilities early, while developers can still efficiently remediate them as part of normal build activity. Dynamic application security testing evaluates a running application, typically by interacting with its exposed interfaces and observing its behavior. It therefore provides complementary coverage for runtime issues that may not be evident from source inspection alone. Automating both forms of analysis in the pipeline supports a DevSecOps approach: security checks are performed continuously, results are visible to the delivery team, and defects can be addressed before a release progresses to later stages. OWASP describes the role of static application security testing and dynamic application security testing in application-security verification.

QUESTION NO: 6

Refer to the exhibit.



```
TASK [Print the contents of the "show _ip_int_brief" variable]
*********************************************************************
ok:  [ios-xe-mgmt.cisco.com] => {
     "show_ip_int_brief": {
          "changed": false,
          "failed": false,
          "stdout" [
               "Interface          IP-Address          OK?  Method Status
Protocol\nGigabitEthernet1    10.10.20.48         YES  NVRAM     up        up
\nGigabitEthernet2       10.10.10.18         YES  other     up        up   \nGigabitEthernet3
13.13.13.13         Yes  other          up        up"
          ],
          "stdout_lines": [
               [
                    "Interface              IP-Address          OK?  Method    Status          Protocol",
                    "GigabitEthernet1       10.10.20.48         YES  NVRAM     up              up",
                    "GigabitEthernet2       10.10.10.18         YES  other     up              up",
                    "GigabitEthernet3       13.13.13.13         YES  other     up              up",
               ]
          ]
     }
}
```

The exhibit shows the output of an Ansible task that prints the contents of the

show_ip_int_brief variable that was registered in a different task in the playbook.

Which expression is used to print the output of the command without its header row?

A. show_ip_int_brief['stdout_lines'][0]

B. show_ip_int_brief['stdout_lines'][1:]

C. show_ip_int_brief['stdout_lines'][0][1:]

D. show_ip_int_brief['stdout_lines']

ANSWER: C

Explanation:

`show_ip_int_brief['stdout_lines'][0][1:]` correctly selects the command output lines while omitting the header. A registered result from a task that executes network CLI commands includes a `stdout_lines` field. This field is structured as a nested list because a task can execute and return output for one or more commands. The first index, `[0]`, selects the list of lines returned for the first command, which includes both the header line and the interface-data lines. The slice `[1:]` then starts at the second item in that line list and continues through the remaining items. Consequently, the header is excluded while every subsequent interface row remains available for rendering or further processing in the playbook. This is standard Python/Jinja list indexing and slicing syntax, which Ansible supports when evaluating templated variables. Ansible documents that command results can expose `stdout_lines` as output split into lines, and its templating system supports accessing list elements and applying Python-style slices. See the Ansible common return values documentation and the Ansible templating guide.

QUESTION NO: 7

A DevOps engineering wants to build an application implementation based on the CI/CD pipeline model. Which service should be used to provide hosted continuous service for open and private projects?

A. Ansible

B. pyATS

C. Genie CLI

D. Travis CI

ANSWER: D

Explanation:

Travis CI is the appropriate service because it is a hosted continuous integration and continuous delivery platform designed to automate software build, test, and deployment workflows. A development team connects its source-code repository to Travis CI and defines the pipeline in a `.travis.yml` configuration file. When developers push commits or open pull requests, Travis CI can automatically provision an execution environment, run the specified validation and test steps, and report the results back to the repository. This makes it well suited to a CI/CD implementation in which changes are consistently verified before release.

Travis CI supports integrations with common source-control providers and is intended for projects whose repositories are publicly available as well as projects requiring private-repository access, subject to the organization's selected plan and configuration. Its hosted model is the key requirement in the question: pipeline infrastructure is provided as a service rather than needing to be built and operated entirely by the engineering team. Cisco DevOps practices commonly emphasize using CI systems to automate repeatable testing and delivery stages. See the Travis CI documentation for configuration and workflow details, and the Cisco DevOps documentation for Cisco's DevOps guidance.

QUESTION NO: 8

Which two statements about Infrastructure as Code are true? (Choose two.)

A. Test-driven development practices make use of Infrastructure as Code.

B. Infrastructure as Code refers to automated testing libraries.

C. DevOps builds upon Infrastructure as Code.

D. Infrastructure as Code is based on practices from software development.

E. Infrastructure as Code must use the same programming language as the application.

ANSWER: C D

Explanation:

DevOps builds upon Infrastructure as Code because IaC provides the repeatable, automated infrastructure provisioning and configuration capabilities needed to support rapid software delivery. Rather than relying on manual device or environment changes, teams define desired infrastructure state in version-controlled files and apply those definitions consistently through automated workflows. This enables the collaboration, repeatability, traceability, and faster feedback loops that are central to DevOps practices. Cisco's DevOps learning material includes infrastructure automation as a core part of implementing DevOps practices across Cisco platforms.

Infrastructure as Code is based on practices from software development because infrastructure definitions are treated as software artifacts. Teams commonly store them in source control, review changes through pull requests, validate and test changes before deployment, and use automated pipelines to deploy them. These practices make infrastructure changes reproducible and auditable while reducing configuration drift. The approach applies software-engineering discipline to networks, compute, cloud resources, and related platform configuration. Cisco describes automation and programmability as key mechanisms for managing infrastructure consistently, while the broader IaC model formalizes that automation through declarative or imperative code definitions. See Cisco DevNet DevOps learning track and HashiCorp: What is Infrastructure as Code?.

QUESTION NO: 9

Which two capabilities of Cisco pyATS and Genie support automated network validation in a CI/CD pipeline? (Choose two.)

A. Parsing device command output into structured data.

B. Comparing learned operational state before and after a change.

C. Replacing source control with device-local configuration archives.

D. Encrypting all device configurations with a Docker logging driver.

E. Automatically converting Python tests into Terraform modules.

ANSWER: A B

Explanation:

Parsing device command output into structured data is correct because Genie parsers transform operational CLI output into structured representations that automation can evaluate programmatically. This lets a pipeline test specific routing, interface, protocol, or platform conditions instead of relying on manual interpretation of text output.

Comparing learned operational state before and after a change is also correct because pyATS and Genie can collect operational state and compare snapshots to identify intended and unintended differences. This supports repeatable precheck and postcheck validation around an automated network deployment. Cisco documents pyATS and Genie capabilities for test automation, parsing, and state learning in the Cisco pyATS documentation.

QUESTION NO: 10 - (DRAG DROP)

Refer to the exhibit. A developer is creating a health check monitoring script that queries information from the Cisco DNA Center platform. The script must trigger an alert if a site health statistic named accessGoodCount drops below 80 and if a network statistic named latestHealthScore is 95 or less.

Drag and drop the code snippets from the bottom onto the blanks in the code to monitor the site and network health on a Cisco DNA Center platform instance. Options may be used more than once. Not all options are used.

url = BASE_URL + SITE_HEALTH	params
url = BASE_URL + NETWORK_HEALTH_URL	'query'
"info"	'timestamp'

ANSWER:

Explanation:

The health-monitoring code correctly uses a timestamp query parameter and then calls the two Cisco DNA Center health endpoints in sequence. The info dictionary must contain the string key `'timestamp'` mapped to the value held by the `timestamp` variable. This creates the request parameter structure expected when the script asks the platform for health information associated with that point in time. Passing this dictionary through `params=info` causes the HTTP client to encode it as a query-string parameter, while `headers=data` continues to provide the Cisco DNA Center authentication token.

For the site check, assigning `url = BASE_URL + SITE_HEALTH` directs the request to `/dna/intent/api/v1/site-health`. The response contains site-health data, including the `accessGoodCount` value the script compares with 80 before invoking the site alert. The script then reassigns the URL with `url = BASE_URL + NETWORK_HEALTH_URL`, which directs the next request to `/dna/intent/api/v1/network-health`. Reusing `params=info` makes the network-health request use the same timestamp input. This provides the network-health response from which the script can evaluate `latestHealthScore` against the required threshold. Cisco documents these health resources and their timestamp query capability in the Cisco DNA Center API documentation. The Python requests library likewise documents that the params argument supplies URL query-string parameters, which is exactly how the timestamp dictionary is used here.

QUESTION NO: 11

A team is implementing Git-based workflow controls for infrastructure-as-code repositories. Which two practices improve the quality and traceability of changes before they are merged into the main branch? (Choose two.)

A. Require pull-request reviews.

B. Run automated validation checks on each pull request.

C. Allow direct commits to the main branch during normal operations.

D. Delete the change history after every successful deployment.

E. Use a separate source repository for every individual change.

ANSWER: A B

Explanation:

Requiring pull-request reviews improves quality and accountability by ensuring that another authorized team member examines a proposed infrastructure change before it is merged. Reviews can identify errors, security concerns, unintended resource changes, and deviations from organizational standards. The pull request also provides a durable record of discussion and approval.

Running automated validation checks on each pull request provides rapid, repeatable feedback before a change is accepted. For infrastructure code, these checks can include syntax validation, formatting, policy evaluation, unit tests, security scanning, and a plan or dry-run operation. The results are associated with the proposed revision and can be enforced as branch-protection requirements.

GitHub documents protected branches and required status checks as mechanisms for controlling merges, while Terraform documents validation and planning as core workflow activities. See GitHub protected branches and the Terraform core workflow.

QUESTION NO: 12

When DevOps practices are integrated into an existing organization, which two characteristics are positive indicators of DevOps maturity? (Choose two.)

A. mean time between success

B. mean time to recover

C. cone testing

D. change lead time

E. age of codebase

ANSWER: B D

Explanation:

Mean time to recover and change lead time are recognized DevOps performance measures. Mean time to recover measures how quickly a team can restore service after an incident or deployment-related failure. A mature DevOps organization aims to reduce this time through observable systems, clear incident processes, automated rollback or remediation, and close collaboration between development and operations. Fast recovery limits customer impact and demonstrates operational resilience.

Change lead time measures the elapsed time from a code change being committed or requested to it running successfully in production. Shorter lead times indicate that the organization can move validated changes through build, test, approval, and deployment workflows efficiently. Continuous integration, automated testing, deployment automation, and small batch sizes help reduce this measure while preserving reliability. Together, these measures show both delivery speed and the ability to restore service quickly, which are central outcomes of mature DevOps practices. Google Cloud's DORA guidance identifies lead time for changes and time to restore service as key software-delivery performance metrics: DORA metrics and measurement model. Cisco's DevOps learning material also emphasizes automation and operational collaboration as foundational DevOps capabilities: Cisco DevOps learning track.

QUESTION NO: 13

Microservices architecture pattern has been applied and the system has been architected as a set of services. Each service is deployed as a set of instances for throughput and availability. In which two ways are these services packaged and deployed? (Choose two.)

A. Service instances must be isolated from one another.

B. Service must be independently deployable and scalable.

C. Service are written using the same languages, frameworks, and framework versions.

D. Service must be dependent, deployable, and scalable.

E. Service instances do not need to be isolated from one another.

ANSWER: A B

Explanation:

In a microservices architecture, each service is a separately managed unit that can have multiple running instances to provide both capacity and resilience. “Service instances must be isolated from one another” is correct because isolation prevents one service’s runtime environment, dependencies, resource consumption, and failures from directly interfering with another service. In practice, this isolation is commonly achieved with containers, virtual machines, or separately managed processes, with containers providing consistent packaging of an application and its dependencies.

“Service must be independently deployable and scalable” is also a defining operational characteristic of microservices. A team should be able to release a service without requiring a coordinated deployment of the entire application, and it should be able to increase or decrease the instance count of that particular service according to its own workload. This autonomy enables faster delivery while allowing capacity to be targeted where it is needed. The microservices guidance in the Microservices.io Microservice Architecture pattern identifies each service as independently deployable and scalable. Docker also describes containers as isolated environments that package an application with its dependencies; see the Docker container overview.

QUESTION NO: 14 - (SIMULATION)

Fill in the blanks to complete the line of Python code that sends a message to a Webex Teams room or person.

```
response = requests.[          ]("https://api.ciscopark.com/v1/[          ]", headers=message_header,
data=message_data)
```

ANSWER: See the explanation for the answer

Explanation:

Fill the blanks with `post` and `messages`:

`POST` is the HTTP method used to create/send a Webex Teams message, and the REST resource for sending messages is `/v1/messages`.

QUESTION NO: 15

Refer to the exhibit.

An organization has issues with code-based failures after implementing a CI/CD pipeline to automate the builds and deployment phases of an application.

Which action must be added to the pipeline, after the application is deployed in the staging environment to minimize failures and to ensure a successful continuous deployment?

A. Restructuring and monitoring tests must be run after it is promoted to production

B. Restructuring and monitoring tests must be run before it is promoted to production

C. Functional and nonfunctional tests must be run after it is promoted to production

D. Functional and nonfunctional tests must be run before it is promoted to production

ANSWER: D

Explanation:

Functional and nonfunctional tests must be run before it is promoted to production. A staging deployment provides a production-like target in which the pipeline can validate the release candidate after deployment but before exposing it to users. Functional testing verifies that the application's expected behaviors, workflows, APIs, and integrations operate correctly in the deployed environment. Nonfunctional testing validates operational qualities that can cause release failures even when features appear to work, such as performance, scalability, reliability, security, and availability. Automating these checks as pipeline gates gives the team rapid, repeatable feedback and prevents an artifact that fails deployment-level validation from being automatically promoted. This supports continuous deployment because promotion remains automated when the defined quality criteria pass, rather than relying on manual, inconsistent validation. It also provides useful test results for diagnosing code-based defects and correcting them early, when the impact and recovery cost are lower. Cisco's DevOps learning materials emphasize CI/CD automation and validation practices across the software delivery lifecycle; see the Cisco DevNet DevOps learning track and the Cisco DevNet DevOps learning modules.

QUESTION NO: 16

Which Dockerfile yields the most predictable builds?

A)

```
FROM python:3.8.0b2-buster

RUN apt update -y
RUN apt upgrade -y
RUN apt install git -y

RUN pip install requests==2.22.0
```

B)

```
FROM python

RUN apt update -y
RUN apt upgrade -y
RUN apt install git -y

RUN pip install requests
```

C)

```
FROM python:3.8.0b2-buster

RUN apt update -y
RUN apt install git -y

RUN pip install requests==2.22.0
```

D)

```
FROM python:latest

RUN apt update -y
RUN apt install git -y

RUN pip install requests==2.22.0
```

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: C

Explanation:

Option C yields the most predictable builds because reproducibility depends on defining every build input as precisely as possible. A predictable Dockerfile uses immutable or tightly pinned dependencies, including a specific base-image version or digest and exact package or application dependency versions. This ensures that a build performed today uses the same inputs as a build performed later, reducing the risk that a newly published image tag, operating-system package, or language-library release silently changes the resulting container.

Docker recommends pinning base-image versions and, when strict reproducibility is required, pinning them by digest. Tags such as `latest` are mutable and can point to different image contents over time. The same principle applies to packages installed during the image build: exact versions make the artifact traceable, testable, and repeatable across developer systems and CI/CD runners. This approach supports reliable promotion through pipeline stages because the image behavior is based on known, controlled inputs rather than whatever happens to be current in an upstream repository. See Docker's guidance on pinning base image versions and its explanation of version-pinning packages during builds.

QUESTION NO: 17

A team is developing an application for end users. The application will use microservices. For user access, dual-factor authentication will be used. Which type of test must be performed by the CI/CD tool to replicate user behavior and to verity that various user actions work as expected?

A. Unit

B. End-to-end

C. A/B

D. sanity

ANSWER: B

Explanation:

End-to-end testing is the appropriate test type because it validates a complete user journey through the application, rather than validating an isolated component or service. A CI/CD pipeline can automate a representative workflow such as opening the application, authenticating through the configured dual-factor authentication process or a controlled test equivalent, navigating the user interface, invoking the required microservices, submitting data, and verifying the final outcome visible to the user. This confirms that the integrated system behaves correctly across its client interface, authentication layer, APIs, service-to-service interactions, data stores, and any required external dependencies. It is particularly important for a microservices application because a feature experienced by an end user often spans several independently deployed services. Automated end-to-end tests therefore provide confidence that real user actions continue to work after a build, configuration change, or deployment. The testing approach and its role in validating complete workflows are described in the Cypress testing documentation and in the Microservice Testing guidance by Martin Fowler.

QUESTION NO: 18

Which two practices help secure a CI/CD pipeline that deploys infrastructure changes? (Choose two.)

A. Use short-lived credentials with least-privilege access.

B. Require approval before production deployment.

C. Share one administrative account among all pipeline stages.

D. Store production credentials in the pipeline YAML file.

E. Allow failed security checks to continue automatically.

ANSWER: A B

Explanation:

Using short-lived credentials with least-privilege access reduces the impact of a compromised pipeline job or credential. The identity used by a pipeline should receive only the permissions required for its specific stage and target environment. Short-lived tokens also reduce exposure compared with static credentials that remain valid indefinitely.

Requiring approval before production deployment provides a controlled gate before a change reaches a sensitive environment. An approval can verify that required tests, security checks, change controls, and deployment conditions have been satisfied. This is especially important when the pipeline can change production infrastructure or network services.

OWASP identifies excessive permissions and inadequate access controls as CI/CD security risks. See the OWASP Top 10 CI/CD Security Risks and the OWASP CI/CD Security Cheat Sheet.