



# DEVCOR

## 350-901

# Study Guide

**Constantin Mohorea**

CCIE® No. 16223, CCDE® No. 20170054, DevNet 500

**Cisco Press**

# **Cisco DEVCOR 350-901**

## **Study Guide**

**Edition: 2021.3**

**Constantin Mohorea**

**Cisco Press**

# **Cisco DEVCOR 350-901 Study Guide**

**Constantin Mohorea**

Copyright © 2022 by Cisco Press

Cisco Press logo is a trademark of Cisco Systems, Inc.

Published by: Cisco Press

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit [www.pearson.com/permissions](http://www.pearson.com/permissions).

No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

Selection on page 19 courtesy of Martin, Robert C. (2003). Agile Software Development, Principles, Patterns, and Practices. Prentice Hall. pp. 127-131.

ISBN-13: 978-0-13-750004-8

ISBN-10: 0-13-750004-1

## **Warning and Disclaimer**

This book is designed to provide information about the “Cisco Certified DevNet Specialist – Core” DEVCOR 350-901 exam. Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied.

The information is provided on an “as is” basis. The author, Cisco Press, and Cisco Systems, Inc., shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of programs that may accompany it.

The opinions expressed in this book belong to the authors and are not necessarily those of Cisco Systems, Inc.

## **Trademark Acknowledgments**

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Cisco Press or Cisco Systems, Inc., cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

## **Special Sales**

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at

[corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact  
[governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact  
[intlcs@pearson.com](mailto:intlcs@pearson.com).

## About the Author

**Constantin Mohorea** is a CX Consulting Engineer at Cisco and is an experienced network professional with a demonstrated history of helping clients achieve their business goals. He excels at building relationships with client teams, understanding their unique requirements, and delivering innovative, stable, and effective network solutions. He brings more than 20 years of experience in the IT industry, specializing in network design, enterprise, security, and data center technologies for customers across a wide spectrum of industries. He is a Cisco Certified Design Expert (CCDE# 20170054) and double Cisco Certified Internetwork Expert (CCIE# 16223) – Security and Enterprise Infrastructure (formerly Routing and Switching).

Constantin has long been interested in programming in general and was happy to see how the networking industry was evolving toward programmability and automation. He was excited about the Cisco DevNet program and got certified on the very first day, earning the honorary DevNet 500 designation. He is a Cisco Certified DevNet Professional and is looking forward to becoming a Cisco Certified DevNet Expert when this certification becomes available.

# About the Technical Reviewer

**Dmitry Figol**, CCIE No. 53592 (R&S) and Cisco Certified DevNet Professional, is a network automation architect at Cisco Global Demo Engineering. He is in charge of software architecture design and implementation of network automation systems. His main expertise is network programmability and Python. Previously, Dmitry worked in Cisco Sales as well as on the Technical Assistance Center (TAC) Core Architecture and VPN teams. Dmitry is a regular conference speaker. He also does live streams on Twitch about network automation and hosts the Network Automation Hangout audio show. Dmitry holds a Bachelor of Science degree in telecommunications. Dmitry can be found on Twitter as @dmfigol.

# **Dedication**

*I dedicate this book to my family: my parents, my wife, and especially my children. Thank you for all your support!*

*Special thanks go to the creators and members of the RouterGods networking community.*

*In addition, I'd like to dedicate this book to all the people who keep studying and improving themselves: Don't stop!*

---

# *Table of Contents*

---

## **TABLE OF CONTENTS**

## **TABLE OF FIGURES**

## **INTRODUCTION**

### **1. SOFTWARE DEVELOPMENT AND DESIGN**

- 1.1 DESCRIBE DISTRIBUTED APPLICATIONS RELATED TO THE CONCEPTS OF FRONT END, BACK END, AND LOAD BALANCING
- 1.2 EVALUATE AN APPLICATION DESIGN CONSIDERING SCALABILITY AND MODULARITY
- 1.3 EVALUATE AN APPLICATION DESIGN CONSIDERING HIGH-AVAILABILITY AND RESILIENCY (INCLUDING ON-PREMISES, HYBRID, AND CLOUD)
- 1.4 EVALUATE AN APPLICATION DESIGN CONSIDERING LATENCY AND RATE-LIMITING
- 1.5 EVALUATE AN APPLICATION DESIGN AND IMPLEMENTATION CONSIDERING MAINTAINABILITY
- 1.6 EVALUATE AN APPLICATION DESIGN AND IMPLEMENTATION CONSIDERING OBSERVABILITY
- 1.7 DIAGNOSE PROBLEMS WITH AN APPLICATION GIVEN LOGS RELATED TO AN EVENT
- 1.8 EVALUATE CHOICE OF DATABASE TYPES WITH RESPECT TO APPLICATION REQUIREMENTS (SUCH AS RELATIONAL, DOCUMENT, GRAPH, COLUMNAR, AND TIME SERIES)
- 1.9 EXPLAIN ARCHITECTURAL PATTERNS (MONOLITHIC, SERVICES-ORIENTED, MICROSERVICES, AND EVENT-DRIVEN)
- 1.10 UTILIZE ADVANCED VERSION CONTROL OPERATIONS WITH GIT
  - 1.10.A MERGE A BRANCH

- 1.10.B RESOLVE CONFLICTS
- 1.10.C GIT RESET
- 1.10.D GIT CHECKOUT
- 1.10.E GIT REVERT
- 1.11 EXPLAIN THE CONCEPTS OF RELEASE PACKAGING AND DEPENDENCY MANAGEMENT
- 1.12 CONSTRUCT A SEQUENCE DIAGRAM THAT INCLUDES API CALLS
- 1.13 CHAPTER 1 REVIEW QUESTIONS

## **2. USING APIs**

- 2.1 IMPLEMENT ROBUST REST API ERROR HANDLING FOR TIMEOUTS AND RATE LIMITS
- 2.2 IMPLEMENT CONTROL FLOW OF CONSUMER CODE FOR UNRECOVERABLE REST API ERRORS
- 2.3 IDENTIFY WAYS TO OPTIMIZE API USAGE THROUGH HTTP CACHE CONTROLS
- 2.4 CONSTRUCT AN APPLICATION THAT CONSUMES A REST API THAT SUPPORTS PAGINATION
- 2.5 DESCRIBE THE STEPS IN THE OAuth2 THREE-LEGGED AUTHORIZATION CODE GRANT FLOW
- 2.6 CHAPTER 2 REVIEW QUESTIONS

## **3. CISCO PLATFORMS**

- 3.1 CONSTRUCT API REQUESTS TO IMPLEMENT CHATOPS WITH WEBEX API
- 3.2 CONSTRUCT API REQUESTS TO CREATE AND DELETE OBJECTS USING FIREPOWER DEVICE MANAGEMENT (FDM)
- 3.3 CONSTRUCT API REQUESTS USING THE MERAKI PLATFORM TO ACCOMPLISH THESE TASKS
  - 3.3.A USE MERAKI DASHBOARD APIs TO ENABLE AN SSID
  - 3.3.B USE MERAKI LOCATION APIs TO RETRIEVE LOCATION DATA
- 3.4 CONSTRUCT API CALLS TO RETRIEVE DATA FROM INTERSIGHT

- 3.5 CONSTRUCT A PYTHON SCRIPT USING THE UCS APIs TO PROVISION A NEW UCS SERVER GIVEN A TEMPLATE
- 3.6 CONSTRUCT A PYTHON SCRIPT USING THE CISCO DNA CENTER APIs TO RETRIEVE AND DISPLAY WIRELESS HEALTH INFORMATION
- 3.7 DESCRIBE THE CAPABILITIES OF APPDYNAMICS WHEN INSTRUMENTING AN APPLICATION
- 3.8 DESCRIBE STEPS TO BUILD A CUSTOM DASHBOARD TO PRESENT DATA COLLECTED FROM CISCO APIs
- 3.9 CHAPTER 3 REVIEW QUESTIONS

## **4. APPLICATION DEPLOYMENT AND SECURITY**

- 4.1 DIAGNOSE A CI/CD PIPELINE FAILURE (SUCH AS MISSING DEPENDENCY, INCOMPATIBLE VERSIONS OF COMPONENTS, AND FAILED TESTS)
- 4.2 INTEGRATE AN APPLICATION INTO A PREBUILT CD ENVIRONMENT LEVERAGING DOCKER AND KUBERNETES
- 4.3 DESCRIBE THE BENEFITS OF CONTINUOUS TESTING AND STATIC CODE ANALYSIS IN A CI PIPELINE
- 4.4 UTILIZE DOCKER TO CONTAINERIZE AN APPLICATION
- 4.5 DESCRIBE THE TENETS OF THE “12-FACTOR APP”
- 4.6 DESCRIBE AN EFFECTIVE LOGGING STRATEGY FOR AN APPLICATION
- 4.7 EXPLAIN DATA PRIVACY CONCERNS RELATED TO STORAGE AND TRANSMISSION OF DATA
- 4.8 IDENTIFY THE SECRET STORAGE APPROACH RELEVANT TO A GIVEN SCENARIO
- 4.9 CONFIGURE APPLICATION-SPECIFIC SSL CERTIFICATES
- 4.10 IMPLEMENT MITIGATION STRATEGIES FOR OWASP THREATS (SUCH AS XSS, CSRF, AND SQL INJECTION)
- 4.11 DESCRIBE HOW END-TO-END ENCRYPTION PRINCIPLES APPLY TO APIs
- 4.12 CHAPTER 4 REVIEW QUESTIONS

## **5. INFRASTRUCTURE AND AUTOMATION**

- 5.1 EXPLAIN CONSIDERATIONS OF MODEL-DRIVEN TELEMETRY (INCLUDING DATA CONSUMPTION AND DATA STORAGE)**
- 5.2 UTILIZE RESTCONF TO CONFIGURE A NETWORK DEVICE INCLUDING INTERFACES, STATIC ROUTES, AND VLANs (IOS XE ONLY)**
- 5.3 CONSTRUCT A WORKFLOW TO CONFIGURE NETWORK PARAMETERS WITH:**
  - 5.3.A ANSIBLE PLAYBOOK**
  - 5.3.B PUPPET MANIFEST**
- 5.4 IDENTIFY A CONFIGURATION MANAGEMENT SOLUTION TO ACHIEVE TECHNICAL / BUSINESS REQUIREMENTS**
- 5.5 DESCRIBE HOW TO HOST AN APPLICATION ON A NETWORK DEVICE (INCLUDING CATALYST 9000 AND CISCO IOx-ENABLED DEVICES)**
- 5.6 CHAPTER 5 REVIEW QUESTIONS**

## **6. APPENDIX A: RESTCONF URI DEMYSTIFIED (IOS XE)**

## **7. APPENDIX B: ANSWERS TO CHAPTER REVIEW QUESTIONS**

- 7.1 ANSWERS TO CHAPTER 1: SOFTWARE DEVELOPMENT AND DESIGN**
- 7.2 ANSWERS TO CHAPTER 2: USING APIs**
- 7.3 ANSWERS TO CHAPTER 3: CISCO PLATFORMS**
- 7.4 ANSWERS TO CHAPTER 4: APPLICATION DEPLOYMENT AND SECURITY**
- 7.5 ANSWERS TO CHAPTER 5: INFRASTRUCTURE AND AUTOMATION**

---

# *Table of Figures*

---

- FIGURE 1: SIMPLE WEB APPLICATION  
FIGURE 2: ADVANCED WEB APPLICATION  
FIGURE 3: HORIZONTAL SCALABILITY IN DISTRIBUTED APPLICATIONS  
FIGURE 4: MONOLITHIC ARCHITECTURE  
FIGURE 5: SERVICE-ORIENTED ARCHITECTURE  
FIGURE 6: MICROSERVICES ARCHITECTURE  
FIGURE 7: EVENT-DRIVEN ARCHITECTURE: EVENT BROKER  
FIGURE 8: EVENT-DRIVEN ARCHITECTURE: EVENT MEDIATOR  
FIGURE 9: GIT AREAS AND COMMON COMMANDS  
FIGURE 10: GIT OBJECT RELATIONSHIP  
FIGURE 11: GIT MERGE: BRANCHING, MASTER NOT CHANGED  
FIGURE 12: GIT MERGE: FAST-FORWARD MERGE  
FIGURE 13: GIT MERGE: BRANCHING, MASTER CHANGED  
FIGURE 14: GIT MERGE: THREE-WAY MERGE  
FIGURE 15: GIT MERGE: NO FAST-FORWARD MERGE  
FIGURE 16: GIT REBASE  
FIGURE 17: GIT FAST-FORWARD MERGE AFTER REBASE  
FIGURE 18: GIT RESET: BEFORE RESET  
FIGURE 19: GIT RESET: THE “SOFT” OPTION  
FIGURE 20: GIT RESET: THE “MIXED” OPTION  
FIGURE 21: GIT RESET: THE “HARD” OPTION  
FIGURE 22: GIT CHECKOUT: DETACHED HEAD  
FIGURE 23: SIMPLE SEQUENCE DIAGRAM  
FIGURE 24: SEQUENCE DIAGRAM WITH CISCO WEBEX AND DNAC API CALLS

FIGURE 25: CONTROL FLOW FOR REST API ERROR HANDLING

FIGURE 26: HTTP CONDITIONAL REQUESTS: FETCH THE RESOURCE

FIGURE 27: HTTP CONDITIONAL REQUESTS: MODIFY THE RESOURCE

FIGURE 28: OAUTH2 THREE-LEGGED AUTHORIZATION

FIGURE 29: MERAKI API STRUCTURE

FIGURE 30: APPDYNAMICS TOPOLOGY MAP SAMPLE

FIGURE 31: APPDYNAMICS BUSINESS TRANSACTIONS VIEW

FIGURE 32: CI/CD PIPELINE EXAMPLE

FIGURE 33: CONTAINERS VS. VIRTUAL MACHINES (VMs)

FIGURE 34: DISTRIBUTED LOGGING

FIGURE 35: ELK STACK

FIGURE 36: IOS XE APPLICATION HOSTING NETWORKING

---

# *Introduction*

---

DevNet certification is Cisco's newest certification track, designed to help network professionals and software developers who want to write applications and develop integrations with Cisco products, platforms, and application programming interfaces (APIs).

To obtain the Cisco Certified DevNet Professional certification, you need to pass:

- The “Developing Applications Using Cisco Core Platforms and APIs” (DEVCOR 350-901) exam
- One of the concentration exams (ENAUTO, DCAUTO, SAUTO, DEVOPS, and so on)

The focus of this book is the DEVCOR 350-901 exam.

Passing this exam counts toward the DevNet Professional certification and also earns you the “Cisco Certified DevNet Specialist – Core” certification, so you get recognized for your accomplishments along the way. Last but not least, it will be a pre-requirement for the Cisco DevNet Expert certification (future offering at the time of this writing).

## **Exam Overview**

The “Developing Applications Using Cisco Core Platforms and APIs” (DEVCOR 350-901) exam tests your knowledge in the following domains, shown with the percentage of the representation on the exam:

- Software Development and Design: 20%

- Using APIs: 20%
- Cisco Platforms: 20%
- Application Deployment and Security: 20%
- Infrastructure and Automation: 20%

The duration of the exam is 120 minutes.

## **Who Should Read This Book?**

The main goal of this book is to help candidates prepare for the “Developing Applications Using Cisco Core Platforms and APIs” DEVCOR 350-901 exam. However, it will also benefit readers studying for the Specialist-level DevNet certification exams, as some of the topics are covered in more detail.

Another goal is to allow readers to confidently apply the knowledge they obtained while studying. As such, technology information and code examples from this book may be used as helpful everyday references.

Please be aware that DEVCOR 350-901 is the expert-level exam, so the assumption is that the reader is very familiar with the Python programming language and other fundamental DevNet topics. Therefore these topics will not be covered in this book.

## **How This Book Is Organized**

The content of this *Study Guide* covers the whole exam’s blueprint (as of the beginning of 2021), strictly aligning the content of each section with the corresponding blueprint topic. Chapter review questions and detailed answers will help solidify understanding of the topics.

The best use of this book is to read it before taking the exam to refresh yourself on all the material you learned. However, it may be equally well used at the beginning of

the exam preparation studies to identify knowledge gaps and areas for improvement.

## Resources to Use

The Cisco DevNet site <https://developer.cisco.com/> is a resource you'll very likely use at least once during your preparations. It has all the latest DevNet certification program information and various documentation (API references, Getting Started guides, and examples). Most importantly, it provides access to various developer sandboxes that allow you to run and test your code against live infrastructure.

If you're just beginning your preparation, take a look at the official course(s) at Cisco Digital Learning (<https://digital-learning.cisco.com/> [and then search for DEVCOR]).

This book may provide enough information for some of the topics, but sometimes you will want to research some of them in more detail. It is advisable to create your own study plan and track your individual progress. You may want to use a publicly available plan created by Nick Russo as a guideline ([http://njrusmc.net/jobaid/devcor\\_studyplan.xlsx](http://njrusmc.net/jobaid/devcor_studyplan.xlsx)).

Note that as automation technologies continue to develop, Cisco reserves the right to change the exam topics without notice. Therefore, always check the <https://cisco.com/go/certifications> website to verify the actual list of topics to ensure you are prepared before taking the exam.

Enjoy the book, and good luck with your exam!

---

# *1. Software Development and Design*

---

## *1.1 Describe distributed applications related to the concepts of front end, back end, and load balancing*

In software engineering, the front-end/back-end model is a way of organizing an application's workflow so as to separate its presentation layer (front end) and the application logic (back end).

The *front end* is the part of the application that receives input from a client in a secure and efficient manner. The *back end* is the part of the application that processes requests, executes business logic, and stores information. Typically, the back end also returns the processing result to the front end so it may be passed back to the client.

For example, in a chat application, the front end may collect user input and pass it to the back end. The back end then would save it to the messaging server, refresh the chat room content, and pass it back to the front end to be displayed to the user.

It is important to note that the client is not necessarily a person. In complex distributed applications, the back end of one service can become the client for the front end of another service. In the preceding example, when the chat application's back end sends a new message to the server to store it, it's likely using the server's application

programming interface (API) front end to access the database instead of directly connecting to it.

The front-end/back-end concept is widely used in web development.

The front end (also called the “client side”) is everything a user sees and interacts with in a browser. The most common front-end languages are HTML, CSS, and JavaScript as well as various frameworks using these languages (React, Angular, and so on).

The back end (also called the “server side”) of a website processes and stores data and ensures everything on the client side works correctly. It is the part of the website that does not come in direct contact with users. Back-end development languages are Java, C++, Python, PHP, and so on.

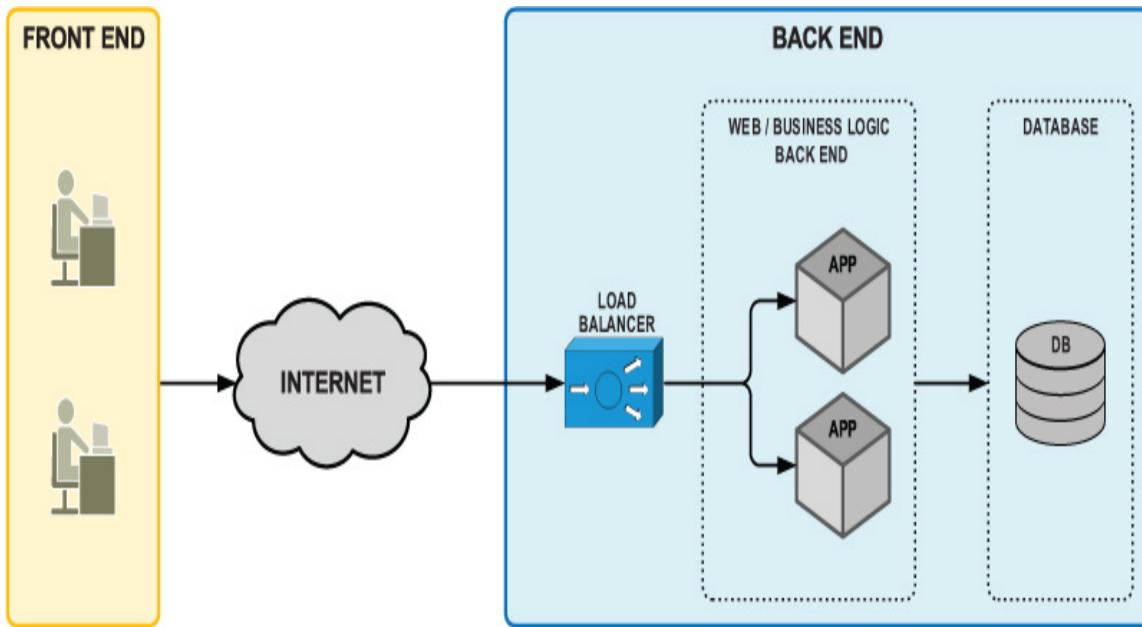
Classic web applications use server-side rendering, where a server receives requests and then creates and sends HTML/CSS responses back to a browser to be displayed.

Changing a view in a browser requires a new page to be provided from a server. The problem is that each request has to travel all the way from the client to the server every time, which introduces latency. Additionally, servers may be busy serving other requests; therefore, overall, user interactions may be sluggish.

One of the ways to increase website performance is to add more servers and use load balancing to efficiently distribute incoming requests across them. Most of the current load balancers qualify as application delivery controllers (ADCs), as load balancing is just one of the techniques they implement to improve the performance of web applications (others are caching, compression, SSL offloading, and so on). There are two kinds of load balancers: hardware based and software based.

In the typical configuration, as shown in [Figure 1](#), the load balancer is placed in front of a group of servers and mediates requests and responses between them and their clients, making the group look like a single virtual server to end users, assigned with its own virtual IP (VIP).

*Figure 1: Simple Web Application*



Load balancing has other benefits besides performance scalability:

- **High availability and reliability:** Requests are only sent to servers that are online and healthy.
- **Reduced downtime:** A single server failure does not bring service down.
- **Redundancy:** Operational servers take over a failed server's load.
- **Flexibility:** Servers may be transparently added or removed from their group, as needed.
- **Efficiency:** The load may be distributed across servers based on their load, response time, number of

connections, and so on.

Global server load balancing (GSLB) is a type of load balancing that addresses the previously mentioned issue of the latency between clients and servers for classic web applications. It is applicable for applications deployed on many servers at multiple geographical locations. GSLB ensures that users connect to a server that is geographically close to them, thus minimizing the travel time.

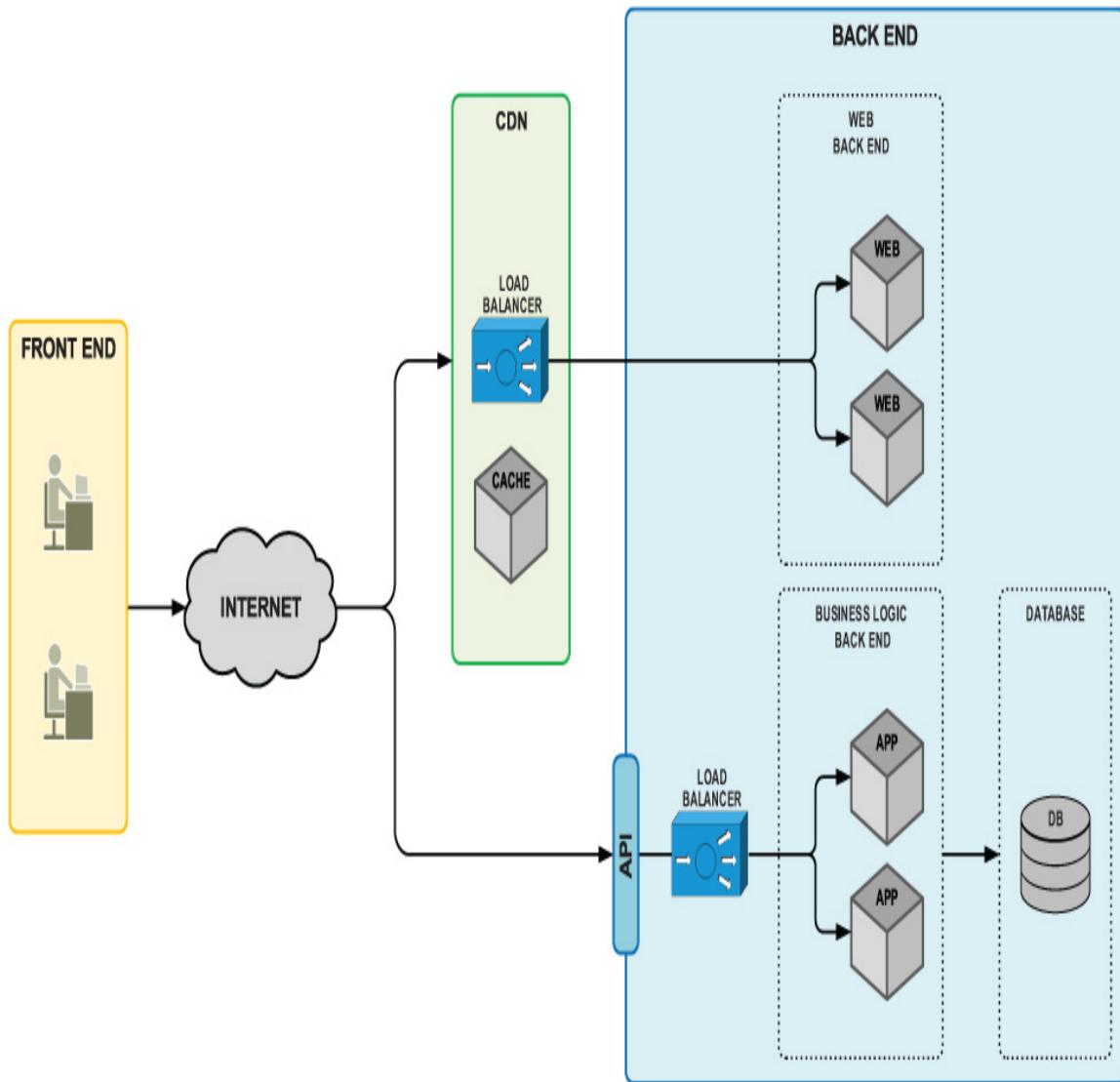
To further reduce latency, modern web architectures use client-side processes and move away from doing everything on a server side. With this model, the server provides a “raw” code that implements some application logic and renders a web page into its final form locally within a browser. It allows the creation of dynamic web pages, where the view changes based on user input and events (for example, hovering a mouse over a thumbnail brings up a full-sized image) without any interaction with a server, resulting in a much better user experience. JavaScript is the language of choice for front-end processes.

With this approach, front-end JavaScript code may use an API, such as REST or GraphQL, to connect to the business logic back end and execute all application logic locally at the client side while the web back end is only left to serve the static content: HTML, CSS, JavaScript source code, images, and videos. Content delivery networks (CDNs) may be used to optimize this static content-distribution process.

CDNs are geographically spread networks of servers that work together to provide faster delivery of Internet content to site visitors. CDNs cache the content from origin servers so it can be served closer to site visitors. Load and utilization of the original servers significantly decrease since end users do not connect directly to them anymore, so the hosting infrastructure may be simplified.

[Figure 2](#) shows an example of the distributed client-side application.

*Figure 2: Advanced Web Application*



## 1.2 Evaluate an application design considering scalability and modularity

*Modularity* is a software design technique that focuses on separating the program functionality into independent and interchangeable modules that contain everything necessary

to execute only one aspect of its functionality. Functions, objects, and modules are all examples of the modularity in the application design.

The main idea of modularity is that internal details of individual modules (called *implementations*) should be hidden behind their public *interfaces*, and all interactions between modules should happen only through these well-defined interfaces. This is an important concept because the code may be organized into separate source modules, but if their internal functions and variables are exposed and used directly from other modules, then that code is not modular.

When you create a modular application, modules are written separately and composed together to construct the executable program. Typically, they are also compiled separately and then linked by a *linker*. An interpreter may perform some of this construction “on the fly” at runtime.

Modular applications, if built correctly, are much more reusable than nonmodular ones since modules may be used again in other projects without a change. Other benefits of the modular design include the following:

- Application becomes more flexible because it's easy to replace one module with another.
- Projects may be broken down into several smaller independent projects.
- Code is much easier to understand and troubleshoot.
- Code is easier to test because each module may be tested individually.
- Code is easier to edit and clean up (that is, refactor).

The concept of modularity is not specific to the application source code and has many other uses. Related to applications, modularity is a feature of the microservices

architecture (see [Section 1.9](#)) and has the following benefits:

- Each microservice executes only one aspect of the distributed application functionality.
- Interactions between microservices happen only via standard API interfaces, and implementation details are hidden.
- Microservices may be reused without a change for other applications.

*Scalability*, on the other hand, is the ability of a system to handle a growing amount of work by adding resources to the system. Scalability can be measured over multiple dimensions, such as the following:

- **Administrative scalability:** The ability to provide access to more organizations or users
- **Functional scalability:** The ability to add new functionality without disrupting existing services
- **Geographic scalability:** The ability to maintain effectiveness during expansion to a larger region
- **Load scalability:** The ability to expand and contract to accommodate heavier or lighter loads
- **Generation scalability:** The ability to scale by adopting new generations of components
- **Heterogeneous scalability:** The ability to adopt components from different vendors

Generally, there are two categories of resource scalability, vertical and horizontal, as detailed next:

- *Scaling vertically* (up/down) means adding resources to (or removing from) the same single node (CPU,

memory, or storage). It is relatively simple to implement, as no redesign is required; however, it's not always cost-effective (high-performance products are disproportionately more expensive), and every resource has its maximum capacity (for example, you cannot fit more than a certain amount of memory into a server). Scaling up resolves performance issues but does not address reliability, as a system is still a single point of failure.

- *Scaling horizontally* (out/in) means adding (or removing) more instances/nodes and distributing load among them. It allows almost infinite scalability but involves more design work and extra components for management and maintenance, such as a load balancer for a web farm.

The hybrid model uses both vertical and horizontal scalability.

Application scalability typically refers to load scalability—the ability of the application to maintain a positive user experience when its load increases due to more active users, more requests, or more data that it has to handle. Addressing it depends on the application architecture.

Traditional monolithic applications are tightly coupled, meaning the components depend on each other and are aware of each other's state. If an application was not properly architected to share its state, it's often impossible to scale such an application horizontally without a redesign. Even if possible, scaling would be ineffective, as the whole monolith needs to be replicated even when a single component experiences performance issues, resulting in increased resource consumption.

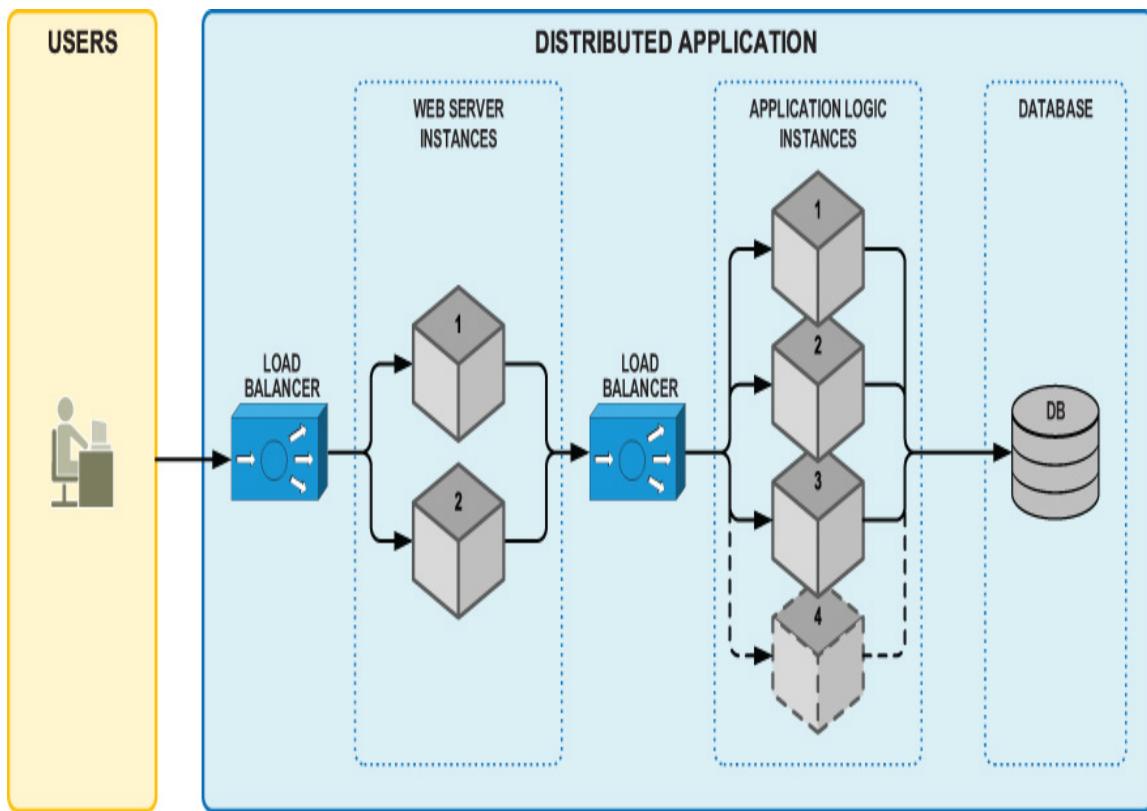
Vertical scaling works better with monolithic applications because it does not require application redesign. However, it

has its limitations: besides the aforementioned cost and capacity restraints, vertical scaling requires server outage to modify hardware components, which might be undesirable for critical applications. When scaling up, it is important to ensure that the application does not have artificial internal limitations (hard-coded maximum number of elements, number of requests per second, and so on) so that added hardware resources are fully utilized.

Distributed applications are split into components (often packaged as containers), communicating with each other over the network. These interactions may be *stateful*, when the previous session's data is stored and used to complete the next request, or *stateless*, when a request may be handled without any knowledge of previous actions.

When their components communicate in a stateless way, distributed applications may be easily scaled horizontally (see [Figure 3](#)). Multiple instances of components may be created with load balancers, ensuring even traffic distribution among them. Since communication is stateless, any of the instances is able to handle requests, so instances may be created as needed. This approach results in more efficient resource utilization, as each component can be scaled up individually.

*Figure 3: Horizontal Scalability in Distributed Applications*



Distributed application components may be scaled vertically as well, but horizontal scaling is typically more cost-efficient and simpler to implement. For example, some cloud services provide built-in *elasticity*—the ability to both acquire resources as needed (scale up) and release resources when no longer needed (scale down) in an automated manner.

### 1.3 Evaluate an application design considering high availability and resiliency (including on-premises, hybrid, and cloud)

Failures happen, especially in the distributed environments, but critical applications need to be available and operational as much as possible. The two methodologies to ensure

continuous operations are high availability and resiliency, and they are often used together.

*Availability* is the degree to which a system is in an operable state, expressed as a percentage of uptime over a period of time. The mathematical formula is  $uptime / (uptime + downtime)$ . Therefore, you can think of it as a probability that the system operates as expected at any given point in time.

*High availability* is a quality of a system or component that ensures a *high level* of operational performance for a given period of time. It is usually measured in “nines,” as an expression of the percentage of uptime in a given year. Here are some examples:

- 90% uptime is “one nine”, which translates to 10% downtime (36.53 days per year)
- 99.95% uptime is “three and a half nines”, or 4.38 hours of downtime per year
- 99.999% uptime is “five nines”, or 5.26 minutes of downtime per year

Availability targets are formalized as service level objectives (SLOs) when included into service level agreements (SLAs) between service providers and their customers. For example, if the SLA guarantees 99.99% uptime, then the availability SLO is 99.99%. The actual measured availability is then tracked with the service level indicator (SLI), which has to meet or exceed the SLO to stay in compliance with the SLA.

Three principles help achieve high availability:

- **Elimination of single points of failure:** Adding or building redundancy into the system so that failure of a component does not mean failure of the entire system.

In the application development, this calls for the stateless data flow design.

- **Reliable failover (switchover):** In case of failure, data flows are redirected to a redundant component by the switchover process. This mechanism itself tends to become a single point of failure, so it needs to be reliable.
- **Detection of failures as they occur:** For an efficient switchover mechanism, component failures should be proactively detected, typically by the monitoring components. The user may never see a failure, but internally it should be detected and acted upon.

Critical server applications that can be reliably utilized with a minimum amount of downtime are often deployed in *high-availability clusters*. They operate by using high-availability software to join redundant compute resources in groups (clusters) that provide continued service when system components fail.

*Resilience* is the ability to provide and maintain an acceptable level of service in the face of unexpected faults and challenges to a normal operation.

The following methods make applications more resilient:

- **Parameter checking:** Complete parameter checking in functions and objects is used to protect from broken or malicious calls and return values.
- **Timeouts:** Used to avoid waiting for a response forever to preserve application responsiveness. Timeouts require some tuning: they should be high enough to allow slower responses but low enough to stop waiting for a response that is never going to arrive.

- **Asynchronous communication:** Used to decouple the sender from the receiver and prevent failures due to failing/slow resources.
- **Fan out and use the quickest response:** Used to send many requests. The first, quickest reply is used, and all other responses are discarded to reduce latency impact (but the tradeoff is a waste of resources).
- **Fail fast:** “If you know you’re going to fail, do it fast.” Fail fast is used to avoid foreseeable failures; add checks before costly actions.
- **Circuit breaker:** If calls fail multiple times, a circuit breaker takes the resource offline and returns an error right away on the next attempts (this is a variant of “fail fast”).
- **Retries:** Used to repeat the same operation a few times in the hope of working around a temporary network or systems problem. There are a few things to consider with retries: if the problem is resource utilization, retries can add more stress, so it’s a good idea to use exponential backoff or a circuit breaker. Also, sometimes requests may actually go through, so calls should be idempotent (for example, “delete X” rather than “delete the last item”).
- **Fallback mechanisms:** Used to continue the execution using the default value in case of a failed request to another service. The value can be predefined or use cached data. This method is not always possible (for example, should we assume a credit card is valid when online verification request timed out?).

In summary, resilience tries to *live through* the failure with error handling and error correction, while high availability's

goal is to detect and *work around* the failure by replacing a broken component.

In the  $\text{availability} = \text{uptime} / (\text{uptime} + \text{downtime})$  formula, high availability's focus is to increase the "uptime" component, and resilience's goal is to decrease "downtime." Both result in improved availability.

## 1.4 Evaluate an application design considering latency and rate limiting

"Latency is zero" is one of the false assumptions that programmers new to distributed applications make (<https://blogs.oracle.com/developers/fallacies-of-distributed-systems>). Ignoring latency effects may result in poor application performance, unresponsiveness, and failures, with the end result being a bad user experience.

The impact of the latency depends on the application type. Here are some examples:

- Real-time applications are directly impacted by latency and may become unusable if it's too high, with the corresponding business impact, such as a wrong decision in stock trading apps, unusable online gaming platforms when players' actions are delayed, and so on.
- For high-performance computing, higher latency means CPUs are idle for a longer time.
- For streaming apps, higher latency results in decreased bandwidth and thus reduced audio/video quality.
- For any application, latency may cause timeouts and retries, which can result in increased network and compute resource utilization.

- A bad user experience is the result in interactive applications due to slowdowns and timeouts.

End-to-end latency has several components:

- **Application latency**: How much time an application spends processing a request. Optimized code and programming language can improve application latency (for example, compiled code executes faster than interpreted code).
- **OS and TCP stack latency**: How much time the operating system (OS) spends processing a request. Pick a specialized OS to improve it.
- **NIC latency**: How much time a packet spends in the interface queue and how long it takes to put it onto a physical wire. Also known as the *serialization delay*, NIC latency is a function of the interface speed. It takes 93ms to push the 1500-byte packet into the wire on a 128KB ISDN interface, 0.12ms on a 100MB LAN, and just 0.5  $\mu$ sec on a 25GB server interface.
- **Cable distance latency (propagation delay)**: A function of the speed of light, which is  $\sim$ 300 km/sec in a vacuum. In fiber media, it's  $\sim$ 1.5 times less, so latency is  $\sim$ 5 microseconds/km. Therefore, it takes at least 100ms to get data to the opposite side of Earth (and 100ms to get a response).
- **Port-to-port latency within each network device along the path**: It may be optimized by using better performing hardware as well as by reducing the number of devices that a packet has to pass through (for example, use Clos fabric or Direct Server Return load balancing).

Latency may be addressed with a proper application design:

- Reducing the speed of light is not an option, but in some cases it is possible to move content closer (for example, using local caching or content delivery networks—geographically distributed servers that provide the same content—where users connect to the geographically closest one).
- Use resource-loading optimization such as *pagination* (making several calls for a smaller amount of data rather than one call for a complete set) and *lazy loading* (loading resource on demand rather than during initialization, which speeds it up).
- Use the appropriate network protocol. TCP has built-in retransmission capabilities, and it will automatically retry a connection when packets are lost/not received. This may introduce undesirable latency (for example, for a video conference), so a UDP-based protocol might be a better choice.
- Proactively prevent resource overload with *rate-limiting* mechanisms, which put a cap on how often someone can repeat an action within a certain timeframe. This can be applied to many processes:
  - **REST API calls:** Return an HTTP 429 “Too Many Requests” code to prevent API overuse and let the requestor know the system is busy at the moment.
  - **Network traffic:** Protect servers and network devices from overload during a distributed denial-of-service (DDoS) attack.
  - **User interactions:** Stop brute-force attacks (for example, do not allow more than three login attempts within 10 minutes).
  - **Data protection:** Do not allow frequent calls to extract data (aka web scraping).

Popular rate-limiting mechanisms are *token bucket* (a certain number of tokens is added and may be consumed during each interval) and *exponential backoff* (time interval between allowed attempts increases after an unsuccessful attempt).

## 1.5 Evaluate an application design and implementation considering maintainability

It is well known that the majority of the cost of software is not its initial development but its ongoing maintenance: fixing bugs, keeping it operational, investigating failures, adapting it to new platforms, modifying it for new use cases, repaying technical debt, and adding new features. Software *Maintainability* refers to the ease of performing all these tasks.

Applications with poor code quality, undetected vulnerabilities, excessive technical complexity, poor documentation, excessive dead code, and so on, require additional maintenance effort. To avoid this, you can use a number of principles, approaches, and techniques that can help you develop maintainable software. These principles result in better outcomes during the design phase:

- Implement modular design to make it easier to maintain individual components.
- Implement object-oriented design to reduce complexity, add modularity, make it easier to implement changes.
- Develop naming conventions.
- Plan to use version control to keep code, tests, and documentation up to date and synchronized.

SOLID principles comprise the popular set of design principles related to code maintainability. The goal is to prevent software from becoming rigid and fragile as it changes. SOLID is an acronym for the following:

- **Single responsibility principle:** Every class should have only a single responsibility.
- **Open-closed principle:** Classes [behavior] should be open for extension but [source] closed for modification. In other words, use a class as a parent for a new object and then make changes there.
- **Liskov's substitution principle:** Objects in a program should be replaceable with derived instances without altering the correctness of the program. In other words, if you inherit from the child object rather than from the parent, the code does not require changes in it.
- **Interface segregation principle:** “A client should not be forced to depend on methods it does not use.” Many client-specific interfaces are better than one general-purpose interface.
- **Dependency inversion principle:** “High-level modules should not depend on low-level modules; both should depend on abstractions. Abstractions should not depend on details; details should depend on abstractions.” Instead of directly referencing modules or methods, add a layer of abstraction. It decouples components and results in more flexible and modular code.

During the implementation phase, following these principles makes software easier to maintain:

- Write consistent, readable code that is easy to understand.

- Perform regular reviews and iterative development to improve quality.
- Refactor code to improve its understandability.
- Write relevant documentation that helps developers understand the software.
- Implement continuous integration to make code easier to build and test.

Use the following tools and methods to achieve better source code quality:

- Develop and follow a naming convention.
- Develop and follow coding conventions: indents, quote symbols, comments, and so on. Use *linters* (tools that analyze source code to flag programming errors, bugs, stylistic errors, and so on) to enforce them.
- Use proper comments: describe “why,” not “how.”
- Follow the DRY (Don’t Repeat Yourself) principle: repeating code should be implemented as a function or an object to reduce duplication.
- Use the same shared libraries and tools.
- Implement automated testing. Well-tested code can be modified quickly and without fear of breaking things. Without it, refactoring becomes riskier, and developers stop doing it.

## 1.6 Evaluate an application design and implementation considering observability

In control theory, observability is the property of a system that tells you how well the internal state of a system can be

understood from the knowledge of its external outputs.

The three pillars of observability are logs, metrics, and tracing.

*Logs* are the most basic observability tool. Logs should provide as much information about application execution flow as possible: state, variable values, API calls, parameters and responses, database calls and responses, and so on. With the evolution of microservice applications, each log stream provides details only about a single instance of service, which is not very helpful. Logs may be aggregated to allow a better event correlation, but it requires extra effort.

*Metrics* are numerical measures recorded by the application over intervals of time, such as counters, gauges, or timers. Metrics may be sampled, aggregated, and summarized for reports or mathematically analyzed to model current and future behavior. Metrics are best suited to trigger alerts. The downside of metrics is that they help understand what is happening at a local system only, without any outside context.

*Distributed tracing* is a technique that brings visibility into the lifetime of a request across several systems. With it, each request is assigned a globally unique ID, and a record containing this ID is generated every time request execution reaches certain instrumentation points in code. These records are logged and used to reconstruct application flow, providing details such as the following:

- Which services did a request go through?
- What did every microservice do when processing the request?
- If the request was slow, where were the bottlenecks?
- If the request failed, where did the error happen?

- How different was the execution of the request from the normal behavior of the system?
- Were some new services called? Also, were some usual services not called?
- Did some service calls take a longer or shorter time than usual?

Looking at these elements, you can see that observability is similar to traditional monitoring. In fact, observability is a superset of monitoring: while monitoring provides information on whether a system is operating as expected, observability also covers the research on *why* an application behaves in a certain way. It does this by providing the ability to ask questions from the *outside* to understand the *inside* of a system.

Similar to scalability and resilience, observability is a system property. The following facts should be acknowledged and addressed during system design, implementation, and operation phases to make it truly observable:

- A complex system is never fully healthy.
- Distributed systems are very unpredictable.
- It's impossible to predict all the states of full or partial failure that various parts of the application may end up in.
- Failures need to be addressed at every phase, from system design to implementation, testing, deployment, and operation.
- Ease of debugging is critical for the maintenance and evolution of an application.

When creating application design, you typically include observability elements in the nonfunctional requirements,

with the observability pillars (logs, metrics, and tracing) applied to the following:

- The status of the application components and services
- The health of the external systems
- Hardware status and load
- User actions

During implementation, pay attention to these low-level elements to achieve better observability:

- Use language-specific logging libraries.
- Implement event and state logging as detailed as possible.
- Log function execution results.
- Log API requests and responses.
- Log transaction and query statuses.

## 1.7 Diagnose problems with an application given logs related to an event

Applications may and will fail. Even if the code is 100% correct, it interacts with other components that may fail on their own. Typically, log review is a starting point for the troubleshooting process.

It is up to developers which content to include in the log messages and how to deliver them (on screen, log files, or remote logging server). Quite often they follow the Syslog protocol's format or include some subset of its fields:

- *Facility* and *severity* of the event (Alert, Critical, Error, Warning, Notice, Informational, or Debug).
- A *timestamp* of an event.

- *Hostname, application name, or process ID* that originated the event.
- Type of message (*Message-ID*).
- *Message* (application-specific text with the event information). The more details included, the easier it is to troubleshoot.

For distributed applications, it is additionally recommended to generate some unique identifier for each transaction, pass it to other components, and include it in all log messages to allow execution tracing throughout the whole application.

When an application misbehaves or crashes, the first step is to see if there are any log entries about the problem, and all those message fields come in handy because they let you find and correlate log information. You can filter by time if you know when errors happened, by the transaction ID (if you know it), or by the severity (for example, search only for the “Critical” and “Error” messages).

When you find error messages, take time to analyze what they mean. Sometimes the problem and its resolution are clear right away, but keep in mind that it could be an indication of another issue:

- If logs indicate a Python exception, you may resolve it by adding “try” / “except” statements, but examine the context around it. Maybe it’s just the user input was not sanitized or a previous API call was unsuccessful, but no error checking was performed.
- If logs indicate some authentication problem, it could be a user entering the wrong password, or maybe the authentication server crashed and was unavailable until it rebooted.

- If a log indicated “Connection refused” or “Connection timeout,” quite often Linux admins would blame the network (which could be true), but in reality (or highly likely), some service on a server did not start or some container crashed. Another possible reason could be a server overload.

Note that application problems do not necessarily exhibit themselves in crashes; they may also result in faults in the application logic. To diagnose these problems in your applications, log often and with details so you can trace application flow and find discrepancies. For external applications, running them in the debug mode might provide enough context for that as well (enabling debug mode permanently is probably not a good idea due to performance considerations).

## 1.8 Evaluate choice of database types with respect to application requirements (such as relational, document, graph, columnar, and time series)

When an application needs to store, update, and access data, it is done with the help of a *database system*. Choosing the right database system is one of the most important decisions, if not *the* most important decision, to make when designing a new software solution, and it is based on many factors and tradeoffs, including the following:

- Kinds/formats of data that may be stored in a database
- Database performance and speed
- Ability to execute complex queries

- Data model flexibility
- Database robustness, reliability, and scalability

Data is usually referred to as being *structured* or *unstructured* (or *semi-structured*, which is a combination of both). This distinction is important because it's directly related to the needed type of database, data query and processing methods, and the complexity of dealing with the data.

Before reviewing database types, let's review some terminology.

A *transaction* is a sequence of operations performed by a database as a single logical unit of work.

*ACID* is an acronym to describe transaction properties:

- **Atomicity**: All operations in a transaction succeed; otherwise, every operation is rolled back.
- **Consistency**: Transactions result in a valid state of the database, and all validation rules and constraints are met.
- **Isolation**: Transactions do not contend with one another. Contentious access to data is moderated by the database so that transactions appear to run sequentially.
- **Durability**: The results of a transaction are permanent, even in the presence of failures.

The *BASE* model is an alternative to ACID:

- **“Basically available”**: Any data request should receive a response, but that response may indicate a failure or changing state, as opposed to the requested data.

- “**Soft state**”: A system may be in a changing state until consistency is reached.
- “**Eventual consistency**”: Data may vary in value until the system reaches a consistent state.

A *distributed database* is a database where data is stored in multiple computers (nodes) interconnected over a network. *The CAP theorem* stipulates that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

- **Consistency**: Every read receives the most recent write or an error.
- **Availability**: Every request receives a (non-error) response, without the guarantee that it contains the most recent write.
- **Partition tolerance**: The database continues to operate even if an arbitrary number of messages were dropped (or delayed) by the network between nodes.

Partitioning tolerance is a critical requirement for a distributed system since no network is safe from failures. When a network fails, the database becomes partitioned, and it can do either of the following:

- Ensure consistency and decrease availability by canceling the operation.
- Ensure availability but risk inconsistency by proceeding with the operation.

Database systems designed with traditional ACID guarantees in mind choose consistency over availability, whereas systems designed around the BASE philosophy choose availability over consistency. In the absence of network failure (when the distributed system is running normally), both availability and consistency can be satisfied.

## Database Types

*Relational databases* (or *SQL databases*) are best suited for storing and querying structured data. Data is stored in tables, where columns (*fields*) describe data and rows (*records*) contain the actual data. Relational databases use fixed schemas on a per-table basis, and they are enforced for each record. SQL (Structured Query Language) is a programming language used to communicate with data stored in a relational database.

Relational databases have a strong focus on providing solid consistency and availability (“CA” of the CAP theorem) and ACID transactions, great functionality, stability, and reliability. They are very well suited for complex querying and non-real-time analytics applications. Relational databases are well known and documented.

The main downside of relational databases is the lack of flexibility. It’s hard to change a database schema (for example, when a new attribute needs to be added to the table) in production.

Popular SQL databases include Oracle Database, Microsoft SQL Server, MySQL, and PostgreSQL

For non-structured data, a variety of *NoSQL database systems* were created, which can be separated into the following categories: document-oriented databases, graph databases, column-oriented databases, key-value databases, and time-series databases.

*Document-oriented databases* store semi-structured information with any number of fields that may contain simple or complex values. Each stored document may have different fields, unlike SQL tables, which require fixed schemas. Documents may embed other documents, thus allowing complex hierarchies of data. Documents in a database are organized into *collections* (similar to tables in

relational databases), which are used to group documents of the same type (for example, product information).

Popular document-oriented databases include MongoDB, Amazon DynamoDB, and Couchbase Server.

*Graph databases* focus on data entities and the relationships between them, which is hard to do efficiently with traditional relational or document databases. A graph database is based on graph theory and consists of a set of node and edge objects:

- *Nodes* represent entities or instances such as people, businesses, accounts, or any other item to be tracked. They are roughly the equivalent of a record in a relational database or a document in a document-store database.
- *Edges* (or *graphs* or *relationships*) are the lines that connect nodes to other nodes, representing the directional relationship between them (for example, the user *follows* another user on Twitter or *likes* a photo on Facebook). Edges are the key concept in graph databases, representing an abstraction that is not directly implemented in other databases.

Relationships may be modeled with the relational databases as well, but queries may become slow and complicated, as this involves using indexes and joining together data from several tables. In contrast, with graph databases, information about relationships is immediately available.

Graph databases perform a quick search through the data related to an individual record. However, they are not that efficient at processing high volumes of transactions or handling queries that perform the same operation on a large number of data elements.

Graph databases are always implemented with another type of database (document, key-value, or a traditional relational) that is used to store the graph data.

Popular graph databases include Neo4j, OrientDB, and ArangoDB.

*Column-oriented databases* are similar to relational databases. Data is organized in both rows and columns. However, fields are defined in rows, and records are stored in columns. This allows highly optimized data retrieval for analytics over a certain field (for example, find the average age); however, other operations are suboptimal (add a record, read a full record, and so on).

Popular column-oriented databases include Apache Cassandra, MariaDB ColumnStore, and Amazon Redshift.

*Key-value databases* offer great simplicity in data storage, allowing for massive scalability of both reads and writes. Data entries are stored with a unique *key* and a corresponding *value*. Key-value databases focus on simplicity, speed, and scalability. Data is not constrained in any way (type, format, and so on), but only basic data manipulation operations are supported (read/write/delete), and search is possible only by the “key” value.

Popular key-value databases include Redis and Amazon DynamoDB.

*Time-series databases (TSDBs)* are specifically built for handling metrics, events, or measurements that are time-stamped. TSDBs have some unique properties that make them suitable for real-time and historical data analytics, like data lifecycle management (aggregating and downsampling into longer-term trend data), data summarization, the ability to handle large time series-dependent scans of many records, and so on. This is the typical database type to store sensor and telemetry data.

Popular time-series databases include InfluxDB and Amazon Timestream.

## 1.9 Explain architectural patterns (monolithic, services-oriented, microservices, and event-driven)

*Software architecture* refers to the fundamental structures of a software system. Each structure is composed of software elements, the relationships among them, and the properties of both. It functions as a blueprint, laying out the tasks necessary to be executed by the design teams.

Software architecture describes fundamental choices that are costly to change once implemented, and it includes items such as the following:

- A high-level overview, including what the software will do when operational (the functional requirements), the general flow of information, and the architecture pattern
- How well the software will perform nonfunctional requirements such as reliability, operability, performance efficiency, security, compatibility, maintainability, and so on
- How well it meets business requirements and environmental contexts that may change over time, such as legal, social, financial, competitive, and technological concerns

Like building architecture, the software architecture discipline has developed standard ways to address repeating concerns, commonly called *reference architecture* or *architectural pattern*.

There are many recognized architectural patterns and styles, including the following:

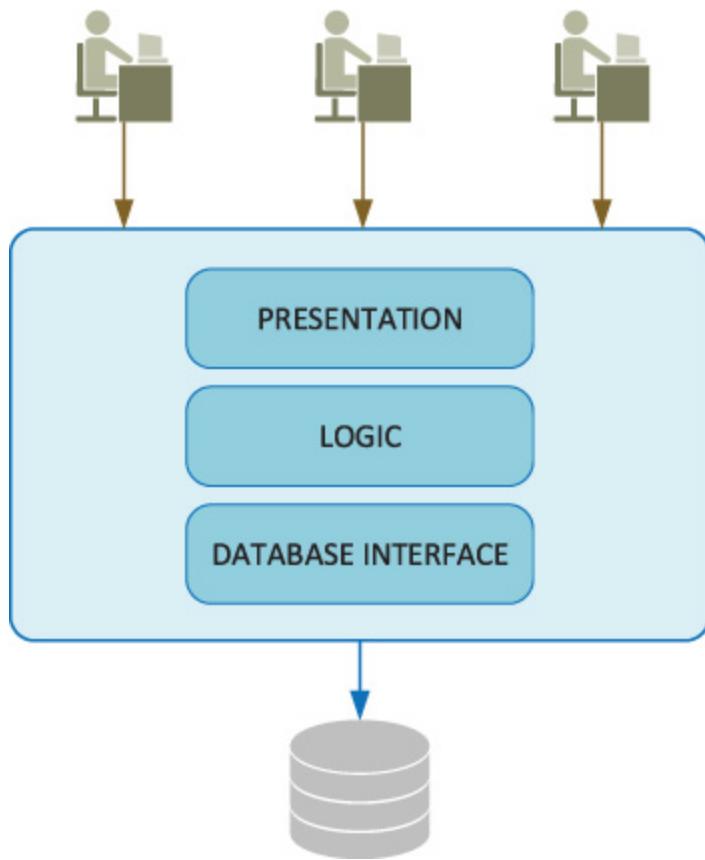
- Monolithic
- Service-oriented
- Microservices
- Event-driven

## Monolithic Architecture

The earliest and simplest (as well as very common) architecture type, *monolithic architecture* describes an application in which the user interface, business logic, and data access code are combined into a single self-sufficient executable. To make any alterations to the system, a developer must build and deploy an updated version of the server-side application.

A monolithic application is self-contained and has little or no interaction with other software (see [Figure 4](#)). The design philosophy is that the application is responsible not just for a particular task but can perform every step needed to complete a particular function. A database is not a part of the application (it may be local or remote), but all the database logic is integrated into the monolith.

*Figure 4: Monolithic Architecture*



Monolithic applications are simple to develop, test, and deploy in the early stages of a project. However, issues with the traditional monolithic design become apparent as the programs grow:

- Scaling vertically is quite easy, but it can only be done to a certain point. Horizontal scaling is more complicated because it requires additional logic and/or software redesign.
- Maintenance becomes more complicated as changes in the tightly coupled source code risk breaking other parts of the program. Each change requires a complete application rebuild and retesting.
- Monolithic applications are implemented using a single development stack, which has its benefits, but at the same time, it can limit the selection of “the right tool

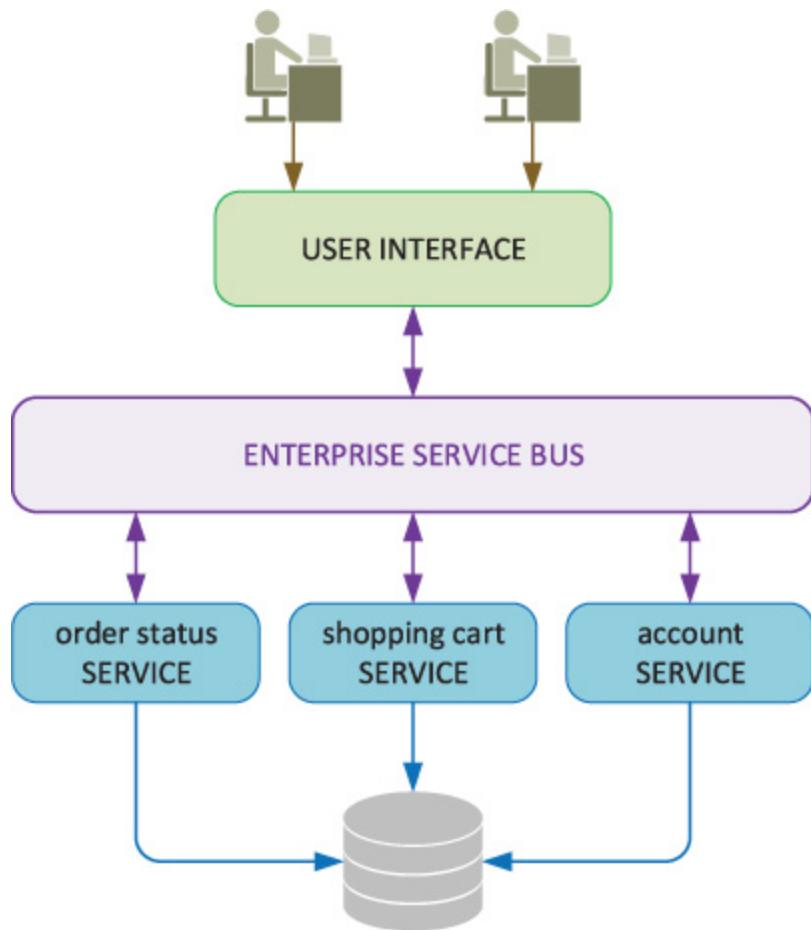
for the job” (for example, if the whole project is written in C++, the web interface has to be written in it as well, even though JavaScript framework might be better suited for that task).

Because of that, some projects start as monoliths but use modular source code structure and follow best practices of the loose coupling design. When code grows to the point it needs to be split up into independent services, an application can be converted into a distributed one.

## **Service-Oriented Architecture (SOA)**

In a service-oriented architecture (SOA), application components provide services to other components through a communication protocol over a network (see [Figure 5](#)). Each service in an SOA implements the code and data integrations required to execute a complete, discrete business function (for example, calculate a monthly loan payment). Service interfaces use common communication standards to allow rapid incorporation into new applications without having to perform deep integration each time.

*Figure 5: Service-Oriented Architecture*



There are no industry standards relating to the exact composition of a service-oriented architecture, although many industry sources have published their own principles. Here are some of them:

- **Standardized service contract**: Services adhere to a standard communications agreement, as defined by one or more service-describing documents.
- **Service reference autonomy (an aspect of loose coupling)**: The relationship between services is minimized to the level that they are only aware of their existence.
- **Service location transparency (an aspect of loose coupling)**: Services may be called from

anywhere within the network, no matter where they are present.

- **Service abstraction:** Services act as black boxes, and their inner logic is hidden from the consumers.
- **Service autonomy:** Services are independent and control the functionality they deliver.
- **Service statelessness:** Services are stateless.
- **Service composability:** Services can be used to compose other services.
- **Service discovery:** Services include some metadata by which they can be effectively discovered and interpreted.
- **Service reusability:** Logic is divided into various services to promote the reuse of code.

*Enterprise Service Bus (ESB)* is a centralized component that performs integration to back-end services, translates their (often incompatible) data models, protocols, and technologies, and makes these available as service interfaces for applications. In theory, it is possible to implement an SOA without an ESB, but service interfaces would need to be exposed individually, which is a lot of work and creates a significant maintenance challenge in the future.

Implementers commonly build SOAs using web services standards such as SOAP/HTML. However, SOA may be implemented using any other service-based technology, such as REST or gRPC.

The benefits of SOA include the following:

- **Greater business agility; faster time to market:** Due to reusability, developers may build applications

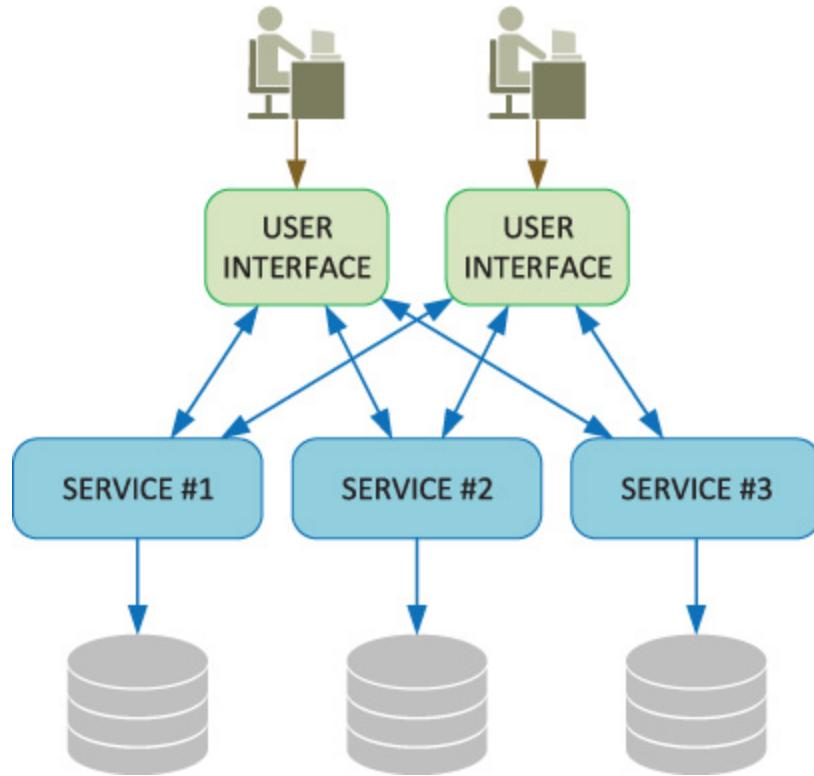
much more quickly in response to new business opportunities.

- **Ability to leverage legacy functionality in new markets:** SOA enables developers to easily take functionality “locked” in one computing platform and extend it to new environments and markets (for example, expose functionality from mainframe-based financial systems to the Web).
- **Improved collaboration between business and IT:** In an SOA, services can be defined in business terms (for example, “generate insurance quote” or “calculate capital equipment ROI”).

## **Microservices**

Microservices architecture is a variant of the service-oriented architecture that organizes an application as a collection of loosely coupled services (see [Figure 6](#)). Services are fine-grained units focused on doing just one thing, whereas with SOA, services are much bigger and complicated.

*Figure 6: Microservices Architecture*



It is common for microservices architectures to be adopted for cloud-native applications running within a containerized environment. Because of the larger number of services, both continuous delivery and DevOps with holistic service monitoring are necessary to effectively develop, maintain, and operate such applications.

The microservices architecture offers the following benefits:

- Modularity makes the application easier to understand, develop, test, and maintain.
- Microservices are easily scalable; because they are implemented and deployed independently of each other, each can be scaled individually if overloaded.
- Microservices enable distributed development, where small autonomous teams develop, deploy, and maintain their respective services independently.

- Microservices can be implemented using different programming languages, databases, hardware, and software environments, depending on what fits best.

On the downside, this architecture introduces additional complexity and new problems to deal with, such as network latency, message format design, load balancing, and fault tolerance, which all have to be addressed at scale. Performance is affected due to increased overhead and network latency.

## **Event-Driven Architecture (EDA)**

The event-driven architecture (EDA) is based on the production of, detection of, consumption of, and reaction to events.

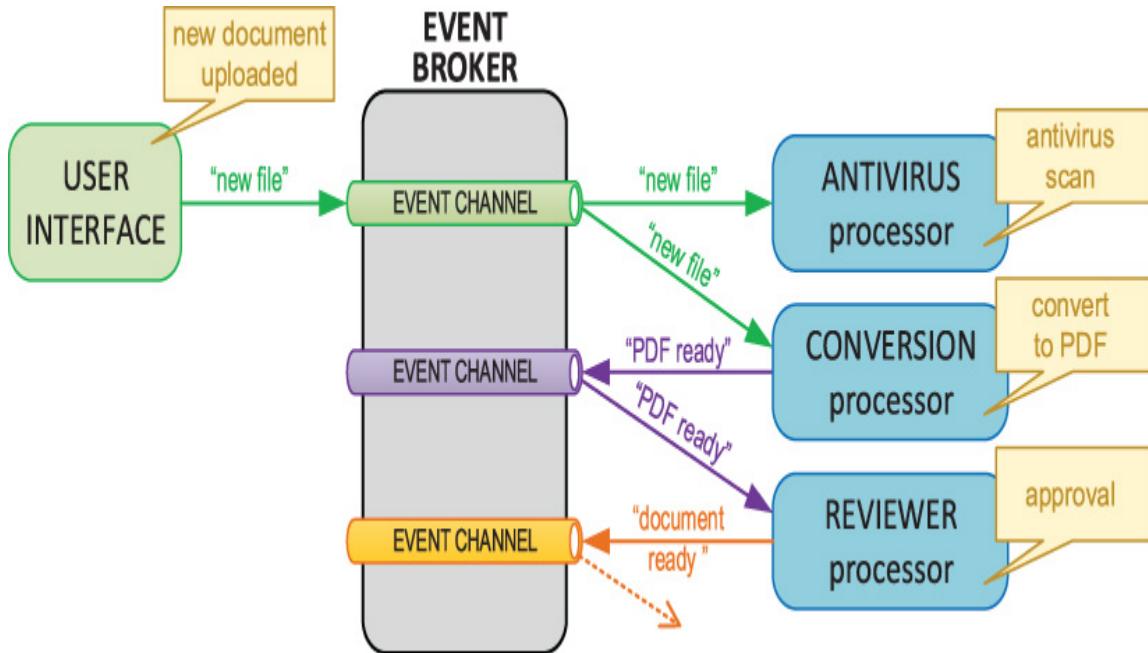
An *event* can be defined as “a significant change in state.” Formally, what is produced, published, propagated, detected, or consumed is an asynchronous message called the *event notification*, and not the event itself, as events do not travel (they just occur). However, the term *event* is often used to denote the notification message itself.

In event-driven architecture, *event creators* produce events and know what events occurred. *Event consumers/processors* are affected by events and process them according to their logic. Events are transported from the event creators to the event consumers via *event channels*. Events are asynchronous and may trigger when resources are not available to respond to them, so EDA also provides storage for them until resources become available in the form of an *event queue*.

Event-driven architecture consists of two main topologies: *mediator* and *broker*. The *mediator topology* is commonly used when there is a need to orchestrate multiple steps within an event through a central mediator, whereas the

*broker topology* is used to chain events and responses together directly (see [Figure 7](#)).

*Figure 7: Event-Driven Architecture: Event Broker*

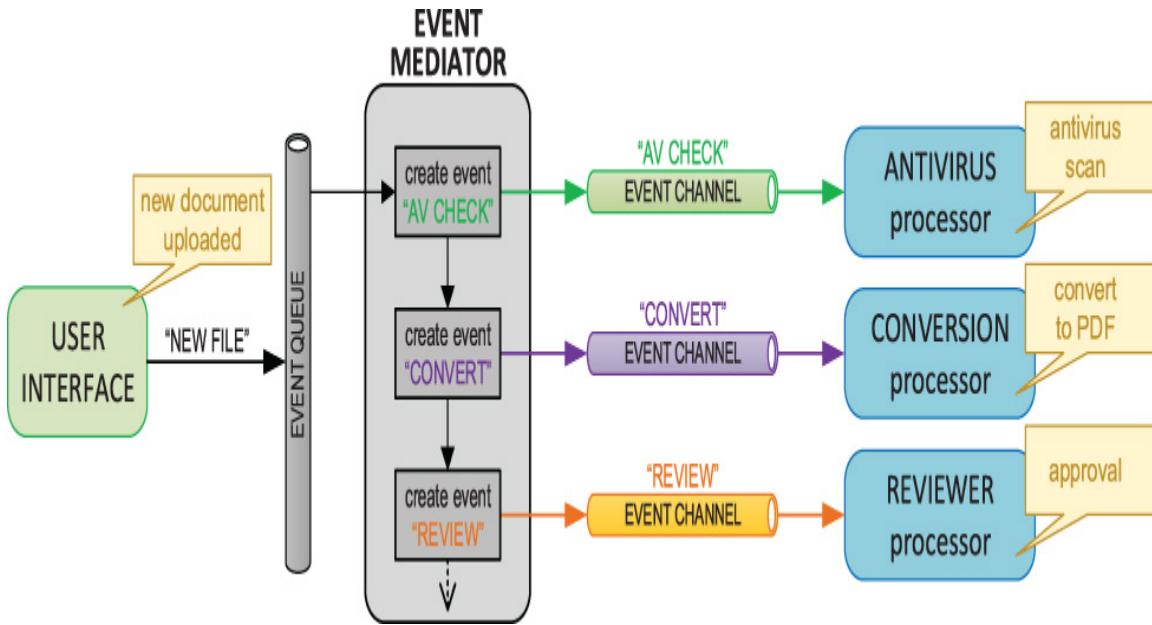


With the broker topology, messages are distributed to the event processors in a chain-like fashion through a message broker that contains event channels (for example, message queues) related to the event flow.

There is no central orchestrating component in this topology; rather, each event processor is responsible for the event handling and publishing of a new event, indicating the action it just performed. These new events are then picked up by other event processors, and the event chain continues until there are no more events.

Mediator topology, shown in [Figure 8](#), is better suited for more complex situations in which multiple steps are required to process an event, requiring event-processing coordination or orchestration.

*Figure 8: Event-Driven Architecture: Event Mediator*



With the mediator topology, clients send events to an event queue, which transports them to the event mediator. The mediator then performs its orchestration by sending additional asynchronous events to specific event channels to execute each step of the process. Event processors listen to the event channels and execute a specific business logic to process these new events.

Overall, event-driven architecture is extremely loosely coupled and well distributed: the event creator does not know where and by whom it will be processed, and event processors do not know where and by whom it was created. As such, event-driven architecture is highly scalable; however, it's more complicated to implement and maintain.

## 1.10 Utilize advanced version control operations with Git

*Git* is a distributed version control system for tracking changes in source code during software development. It was designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals

include speed, data integrity, and support for distributed, nonlinear (branching) workflows.

## Git Architecture

There are three core components/areas of a Git project: *working directory*, *staging area* (or *index*), and *repository*. When you're working in a Git repository, files and modifications will travel from the *working directory* to the *staging area* and finish at the *local repository*.

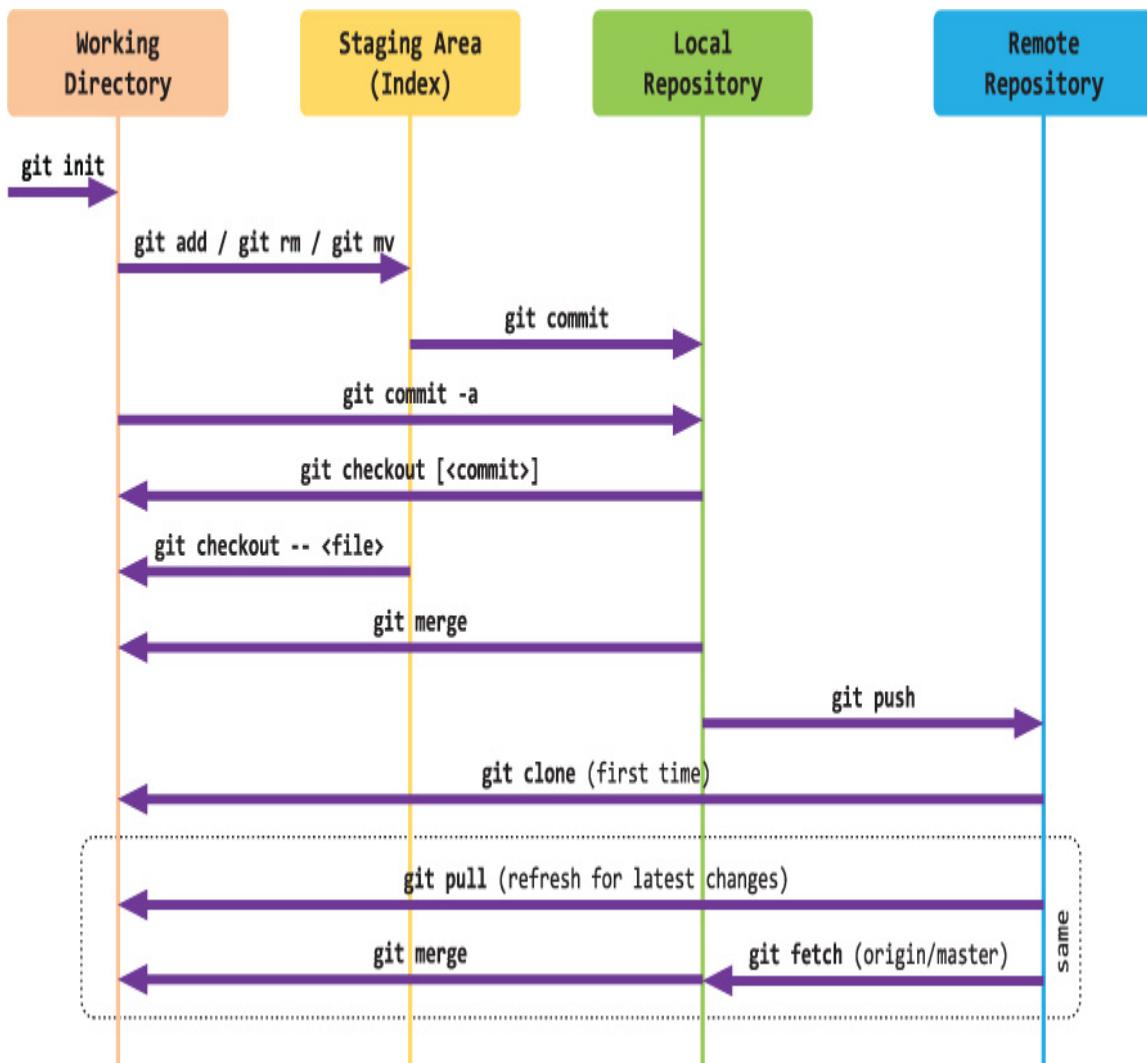
When you work on your project, you're making changes in your project's *working directory*, which is a regular directory on your computer's file system. All the changes you make will remain in the working directory until you add them to the *staging area* (via the `git add` command).

The *staging area* is a connection point between the working tree and the object database and is best described as a preview of your next commit. When you run the `git commit` command, Git takes the changes that are in the staging area and makes the new commit out of them.

After you commit your changes, they go into the *local repository*, where they are stored as commit, blob, and tree objects. You can use the local repository for small projects, but usually a *remote repository* is used as central storage (for example, GitHub). Individual contributors then make a copy of it to a local repository, work on a project, and then synchronize changes back to the remote repository.

The relationship between these areas and typical Git commands is shown in [Figure 9](#).

*Figure 9: Git Areas and Common Commands*



At the lower level, Git has two data structures: a mutable *index* (also called *stage* or *cache*), which contains information about the working directory and the next revision to be committed, and an immutable, append-only object database.

The object database contains these types of objects (there are a few more than listed here):

- A *blob* (binary large object) is the content of a file. Blobs have no proper filename, timestamps, or other metadata (internally, a blob's name is a hash of its content).

- A *tree object* is the equivalent of a directory. It contains a list of filenames, each with some type bits and a reference to a blob or tree object that is that file, symbolic link, or directory's contents. These objects are a snapshot of the source tree.
- A *commit object* links tree objects together into history. It contains the name of the top-level tree object, the author/committer information with a timestamp, the commit message, and names of zero or more parent commit objects.

Each object is identified by an SHA-1 hash of its contents. Git computes the hash and uses this value for the object's name.

Note: hashes are long strings; you can use just the first characters when you need to identify a specific hash in Git commands.

You can investigate the object relationship in the database with the `git log` and `git cat-file` commands, as shown in [Figure 10](#).

*Figure 10: Git Object Relationship*



The command `git log` (or its compact form) displays hashes of the commit objects, and `git cat-file -p` displays the content of the commit, tree, parent, and blob objects in a readable way.

Here, commit with the `c2db3feb4b44d854ffa95a6dc0fe226fc26c2921` hash (the shorter form is `c2db3fe`) has the `7467744...38f` parent (which is the first commit, as it does not have any parents) and contains the `60f26e...b71` tree object, which, in turn, contains pointers to blobs. In this case, blob `c200...12c` has the content of the working file.

## Git Branches

To allow working on several features in parallel and to keep the main code from potentially unstable changes, Git uses *branches*. A Git branch is essentially an independent line of development. You can take advantage of branching when

working on new features or bug fixes because it isolates your work from that of other team members.

When work is finished, the resulting code can be *merged* into the main branch or any other branch in the repository. It is a common practice to create a new branch for each task, as this allows others to easily identify what changes to expect and makes backtracking simple.

The implementation of branches in Git is very simple: a branch in Git is just a lightweight movable pointer to one of the commits. The default branch name in Git is “master,” which is automatically created as soon as you’ve made the first commit, but you can create as many new branches as needed.

To create a branch, use the `git branch <branch>` command; then, to switch to it, use the `git checkout <branch>` command. Alternatively, use `git -b checkout <branch>`, which does both at once.

When a new branch is created, no data is moved around (like in some other version control systems); just a new pointer to an existing commit is generated. When a new commit to a branch is made, this branch pointer moves forward automatically, so normally it points to the last commit to a branch.

Therefore, if several branches exist and we want to make a new commit, how does Git know which branch it will belong to? There’s another special pointer called “HEAD,” and it points to the local branch you’re currently on.

Let’s check out how it works (the `checkout` command will be reviewed later) by performing a series of commands and checking the Git log after each of them:

---

#	<b>Git Command</b>	<b>“git log” output</b>
1	git commit -am “First”	5cff54b (HEAD -> master) First
2	git branch bugfix	5cff54b (HEAD -> master, bugfix) First
3	git checkout bugfix	5cff54b (HEAD -> bugfix, master) First
4	git commit -am 'Second (to bugfix)'	2d3d278 (HEAD -> bugfix) Second (to bugfix)  5cff54b (master) First
5	git checkout master	2d3d278 (bugfix) Second (to bugfix)  5cff54b (HEAD -> master) First

#	Git Command	"git log" output
6	git commit -am 'Third (to master)'	<pre>7636243 (HEAD -&gt; master) Third (to master)  2d3d278 (bugfix) Second (to bugfix)  5cff54b First</pre>

1. Regular commit: “HEAD” points to “master” and “master” points to the commit “5cff54b.”
2. New “bugfix” branch is created: “HEAD” points to “master” and both “master” and “bugfix” branches point to the commit “5cff54b.”
3. New “bugfix” branch is activated: “HEAD” points to “bugfix” now, and both “master” and “bugfix” still point to the same commit (“5cff54b”).
4. Commit to the “bugfix” branch (as “HEAD” points to “bugfix”): “bugfix” points to the new commit “2d3d278”; “master” still points to the original commit (“5cff54b”).
5. The “master” branch is reactivated: “HEAD” points to “master” now, and “bugfix” and “master” didn’t change.
6. Commit to the “master” branch (as “HEAD” points to “master”): “bugfix” still points to commit “2d3d278”; “master” points to the new commit (“7636243”).

Another way to investigate Git and confirm branches and HEAD are just pointers is to check physical files in the repository's `.git/` directory. Branches are stored in the `.git/refs/heads` directory and HEAD in the `.git/HEAD` file. Here are their state after the preceding exercise:

[Click here to view code image](#)

```
$ ls -1 .git/refs/heads/*
.git/refs/heads/bugfix
.git/refs/heads/master

$ cat .git/refs/heads/master
76362436c76d9a40458c67cf3d840263fc924097

$ cat .git/refs/heads/bugfix
2d3d27828c03967384f6b636f87b0db33621abac

$ cat .git/HEAD
ref: refs/heads/master
```

This matches the output from the `git log` command: “HEAD” points to “master,” “master” points to the commit “7636243,” and “bugfix” points to the commit “2d3d278.”

## Git Commit Revision Selection

Git allows you to refer to commits in a number of ways. You can refer to any single commit by its full 40-character SHA-1 hash, but there are more human-friendly ways to do it:

- **Short reference:** Use the first few (at least four) characters of the hash, as long as no other object in the object database has a hash that begins with the same prefix. See examples in the preceding section.
- **Branch reference:** To refer to a tip of the branch, use the branch name. The commands `git show master` and `git show 5cff54b` are equivalent if “master” points to the commit “5cff54b.”

- **“HEAD” reference:** To refer to a tip of the active branch, use “HEAD.” The commands `git show HEAD` and `git show master` are equivalent if “master” is the active branch.

- **Ancestry references:**

- A caret (^) at the end of the reference means “parent of that commit.” Use a number after the caret to indicate which parent you want: first or second (for merge commits). It may be used multiple times to indicate the parent of the parent:  
`git show HEAD ^^2`.
- A tilde (~) at the end of the reference means “the first parent of commit,” so `master^1` and `master~` are equivalent. However, when a number is specified, it means how many parents would be traversed. It may be used multiple times as well: the commands `git show HEAD~~~` and `git show HEAD~3` are equivalent.

Caret and tilde syntax may be combined: `HEAD~3^2` refers to the second parent of the first parent of the first parent of the first parent of the “HEAD” commit.

- **RefLog references:** Git maintains a history (log) of where your “HEAD” and branch references have been for the last 30 days (by default). You can check it with the `git reflog` command, and you can refer to its entries using the `@{<index>}` syntax (for example, `git show HEAD@{3}`).

Note that commits that were removed from history are still temporarily visible in the RefLog and may still be used as references (for example, `git reset --hard HEAD@{1}` would recover your files after a wrongful `git reset`).

## 1.10.A Merge a branch

Git encourages branch creation as often as needed. Once work on a feature branch is complete, it would be typically *merged* with the original branch. This may be accomplished by using the `git merge` or `git rebase` command:

- **Merge**: Retains all changes to the merged branch and the complete history.
- **Rebase**: Maintains a cleaner, linear revision history since merged commits are appended at the end of the target branch. Conflicts may occur more often than in the merge method, and they need to be resolved immediately.

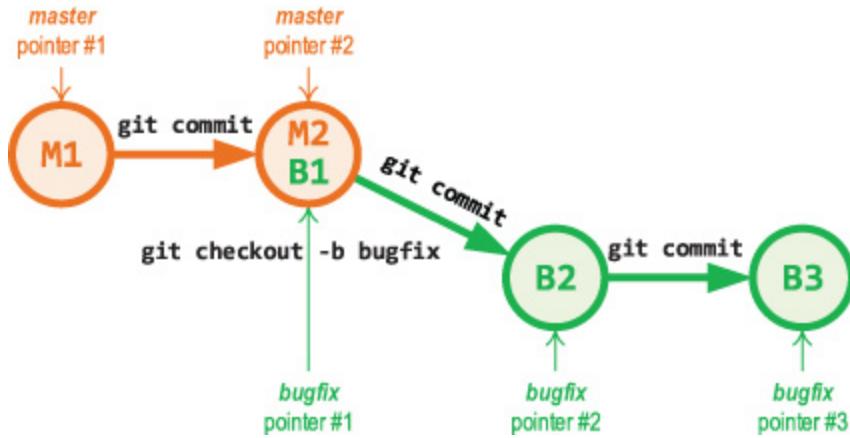
Before performing a merge, you need to take a few preparation steps to ensure the merge goes smoothly:

- Ensure that “HEAD” is pointing to the correct merge-receiving branch. If needed, execute `git checkout` to switch to the receiving branch.
- The working and staging areas need to be clean; all changes need to be *committed* or *stashed* before merging; otherwise, the merge will fail with a descriptive error.

The simplest method to merge a different branch into the active branch is the `git merge <branch>` command.

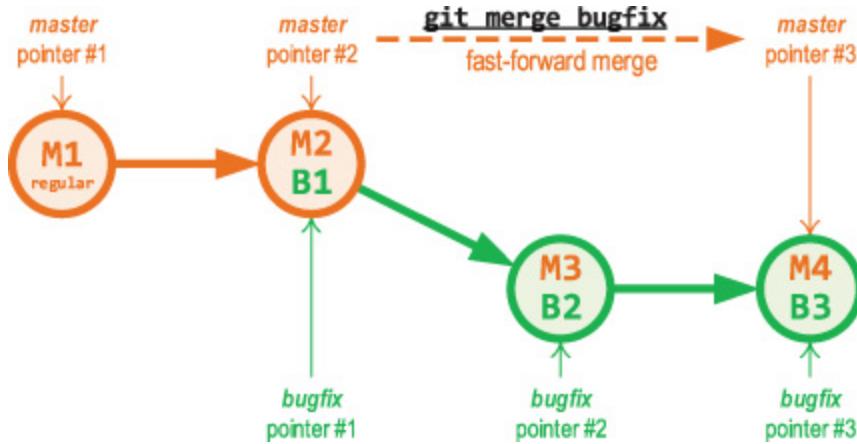
Let’s review some examples next. There are two branches: a “bugfix” branch with a few commits coming off the “master” branch, as shown in [Figure 11](#).

*Figure 11: Git Merge: Branching, Master Not Changed*



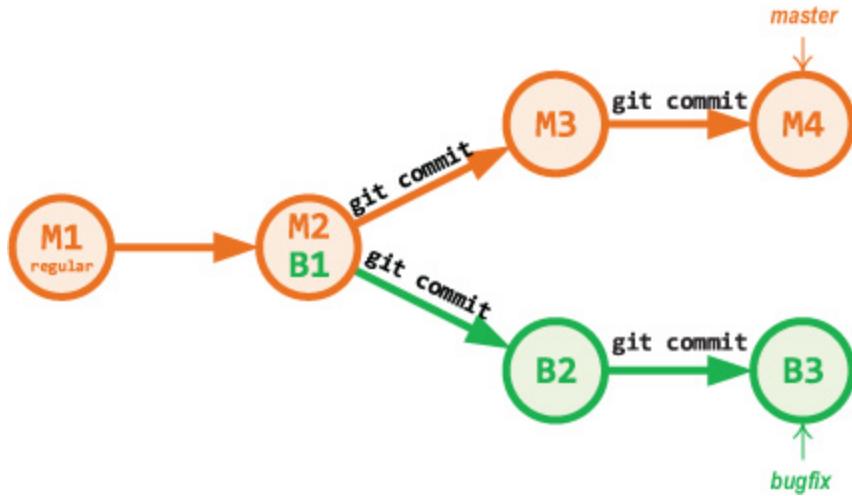
In the ideal case, all the development changes are committed to the new branch and there are no changes to the “master,” and its state has not changed since “bugfix” was created. In such a case, instead of merging branches, all Git has to do is just move the “master” branch tip pointer to the latest position of the “bugfix” pointer. This merge is called a “fast-forward”. With this merge, histories are effectively combined and “bugfix” commits will become visible as a part of the “master” history, as shown in [Figure 12](#).

*Figure 12: Git Merge: Fast-Forward Merge*



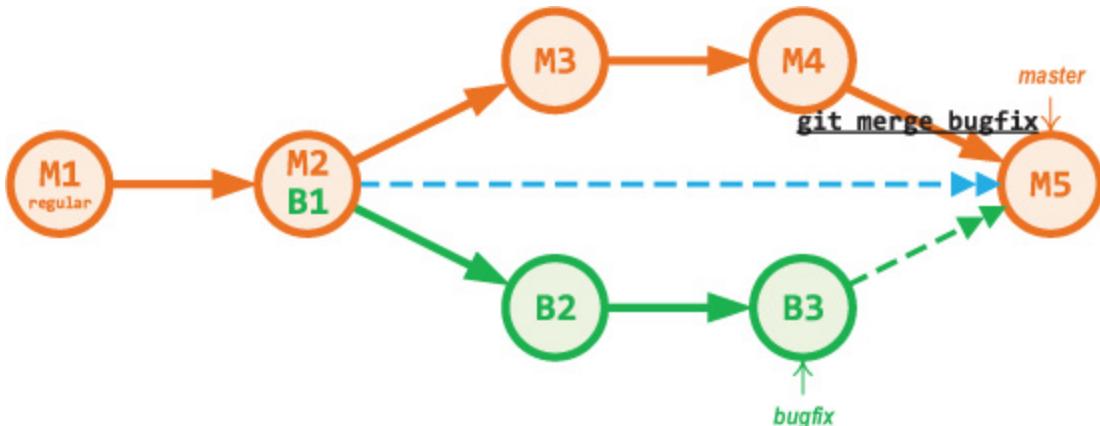
A more common scenario is that there would be new commits on the “master” by the merge time, as shown in [Figure 13](#).

Figure 13: Git Merge: Branching, Master Changed



In this case, *fast-forward* is not possible and Git tries to perform a *three-way merge*, as shown in Figure 14. It uses three commits to complete the merge: the two branch tips and their common ancestor. If an automatic merge is successful and there are no conflicts, a new *merge commit* is created to tie together these branches and their history (it will have two parents). If there are conflicts, this commit is not created.

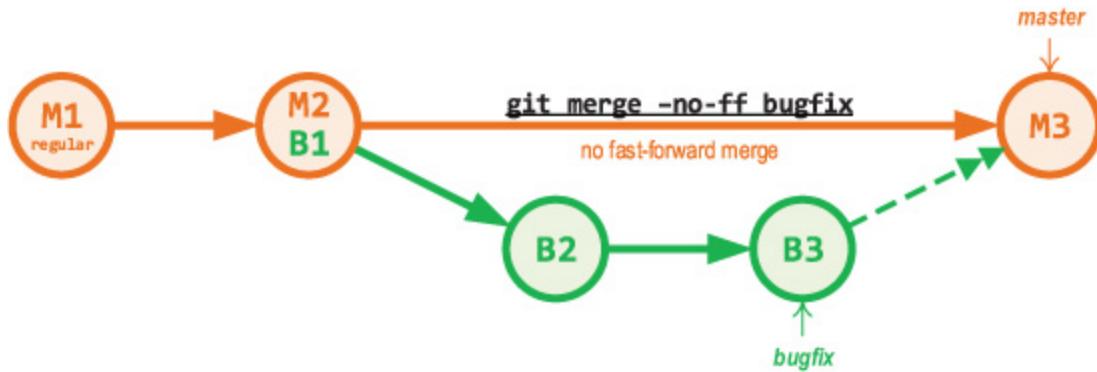
Figure 14: Git Merge: Three-Way Merge



Sometimes a regular *merge commit* is required for the documenting and tracking purposes, as shown in Figure 15, even though the fast-forward method is possible; `git merge`

with the `--no-ff` option does exactly that (compare to [Figure 12](#)).

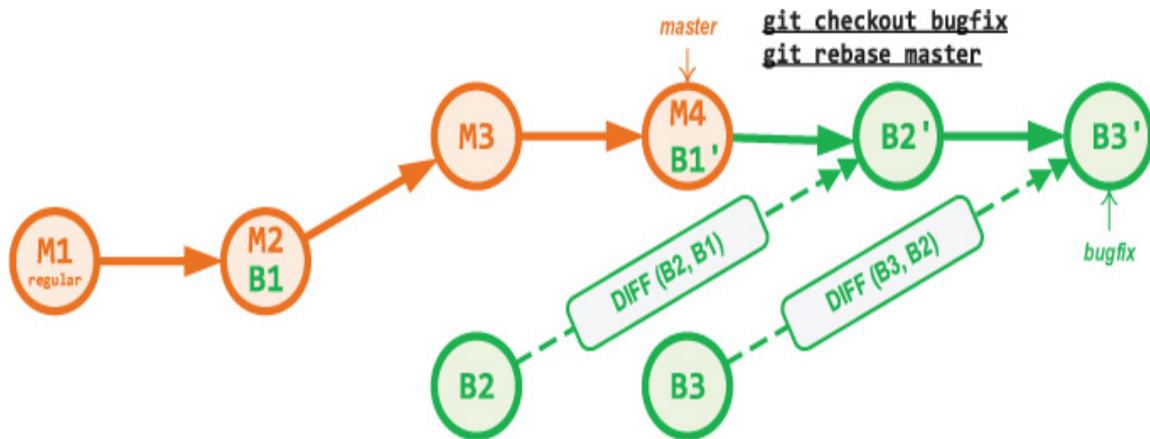
*Figure 15: Git Merge: No Fast-Forward Merge*



Git has a second way to integrate your branches called *rebase*.

Consider again the situation where commits were made in parallel in both the “master” and “bugfix” branches (refer to [Figure 13](#)), and then the `git rebase master` command is performed on the active “bugfix” branch, as shown in [Figure 16](#).

*Figure 16: Git Rebase*



This is what’s called *rebasing the “bugfix” branch onto the “master” branch*, with the outcome of all the changes that

were committed to the “bugfix” branch now replayed on a “master” branch, as follows:

- By identifying the common ancestor of the two branches (M2/B1)
- By getting diffs introduced by each commit in the “bugfix” branch (B1->B2 and B2->B3).
- By resetting the current branch to the same commit as the branch you are rebasing onto (B1'=M4)
- By applying each diff, one by one, to create new commits:  $B2' = B1' + \text{diff}(B1 \rightarrow B2)$ , then  $B3' = B2' + \text{diff}(B2 \rightarrow B3)$

At this point, you can go back to the “master” branch and do a simple fast-forward merge, as shown in [Figure 17](#).

*Figure 17: Git Fast-Forward Merge after Rebase*



As a result, the snapshot M6/B3' is exactly the same as M5 in the three-way merge example (refer to [Figure 14](#)). There is no difference in the outcome of the integration, but rebasing makes a cleaner history. Logging of a rebased branch shows a linear history, with all the work happening sequentially, even though originally it was done in parallel.

Note: The original B2 and B3 become “orphaned” and will eventually be removed by Git’s garbage collection process.

## 1.10.B Resolve conflicts

All the integration examples in the previous section assumed that the changes were non-interfering, so Git was able to automatically combine content.

In reality, quite often two people may have changed the same lines in a file in their branches, or one developer may have deleted a file while another was modifying it. In such cases, Git cannot automatically determine what is correct, so it marks the files as being conflicted and halts the merging process. This is called *merge conflict*, and it is then the developer's responsibility to manually resolve it.

Merge conflict may happen at two points: when starting and during a merge process.

If there are changes in either the working directory or staging area, Git stops the merge process since these pending changes could be overwritten:

[Click here to view code image](#)

```
$ git merge bugfix
error: Your local changes to the following files would be
overwritten by merge:
  file.txt
Please commit your changes or stash them before you merge.
Aborting
```

These changes need to either be saved (`git commit` or `git stash`) or dropped (`git reset`) before Git can start merging again.

A failure *during* a merge indicates a conflict between the current local branch and another developer's code. Git does its best to merge the files but will leave things in the conflicted files to be resolved manually. Let's investigate this process by running a script with the following sequence:

[Click here to view code image](#)

```
git init
echo -n >file.txt
git add file.txt

echo "Commit #1: common" > file.txt
git commit -am "Common"

git checkout -b bugfix
echo "Commit #2: bugfix branch" >> file.txt
git commit -am "Bugfix branch"

git checkout master
echo "Commit #3: master branch" >> file.txt
git commit -am "Master branch"
```

This script creates three commits with the **file.txt** file: one original and then one in each branch, both modifying the file's content. Now let's try to merge branches:

[Click here to view code image](#)

```
$ git merge bugfix
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the
result.
```

As expected, the automatic merge has failed, and Git lets us know that conflict happened and it needs to be resolved. To get more details, use the `git status` command:

[Click here to view code image](#)

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:  file.txt
```

At this point, we can either edit the file to resolve a conflict, stage it and commit the whole merge, or just roll back the merge process with the `git merge --abort` command. Git has indicated that some conflict has been detected in the **file.txt** file, so let's look into it:

[Click here to view code image](#)

```
$ cat file.txt
Commit #1: common
<<<<< HEAD
Commit #3: master branch
=====
Commit #2: bugfix branch
>>>>> bugfix
```

You can see all our commits as well as conflict dividers. The `=====` line is the “center” of the conflict. Everything between the center and the `<<<<< HEAD` line is content that exists in the current branch (“master”), and everything between the center and `>>>>> bugfix` is content that is present in the merging branch (“bugfix”).

The most direct way to resolve a merge conflict is to manually edit the conflicted file in your favorite editor. After that, stage it with the `git add <file>` command and then commit the branch. Git will see that the conflict has been resolved and create a new merge commit to finalize the merge:

[Click here to view code image](#)

```
$ cat file.txt
Commit #1: common
Commit #2: merged bugfix branch
Commit #3: merged master branch

$ git add file.txt
$ git commit -m "Merge with the resolved conflict"
[master a452e0d] Merge with the resolved conflict

$ git log --oneline --graph
*   a452e0d (HEAD -> master) Merge with the resolved conflict
```

```
| \
| * 2d75a4b (bugfix) Bugfix branch
* | 53f53ac Master branch
|/
* 2c0fd4a Common
```

For some of the files, rather than resolving conflict, you just need to choose one side (for example, for a binary file). The `git checkout` command, with the `--ours` (for the current branch) and `--theirs` (for the merging branch) options, does just that:

[Click here to view code image](#)

```
$ git checkout --ours
file.txt
$ cat file.txt
Commit #1: common
Commit #3: master branch
```

[Click here to view code image](#)

```
$ git checkout --theirs
file.txt
$ cat file.txt
Commit #1: common
Commit #2: bugfix branch
```

Note that a successful Git merge process is not a guarantee that the merge was a success from the project perspective. For example, if one branch modifies the name of the object and another keeps using the old one, `git merge` may be successful but the project won't compile now. You can do a few things to avoid this:

- Proactively compare branches before merging to preview conflicts (`git diff ..bugfix`)
- Use the `git merge --no-commit` command, which performs the merge but does not auto-commit, and thus gives us the opportunity to inspect and tweak the merge result before committing.

## 1.10.C git reset

As a refresher on the Git lifecycle, it has three main areas:

- The *working directory*, where changing project files reside
- The *staging area*, where the commit files are prepared
- The *repository*, where Git commits are stored.  
*Branches* are pointers to the latest commit in series, and “HEAD” is a pointer to the current commit, so effectively it’s a pointer to the latest current commit.

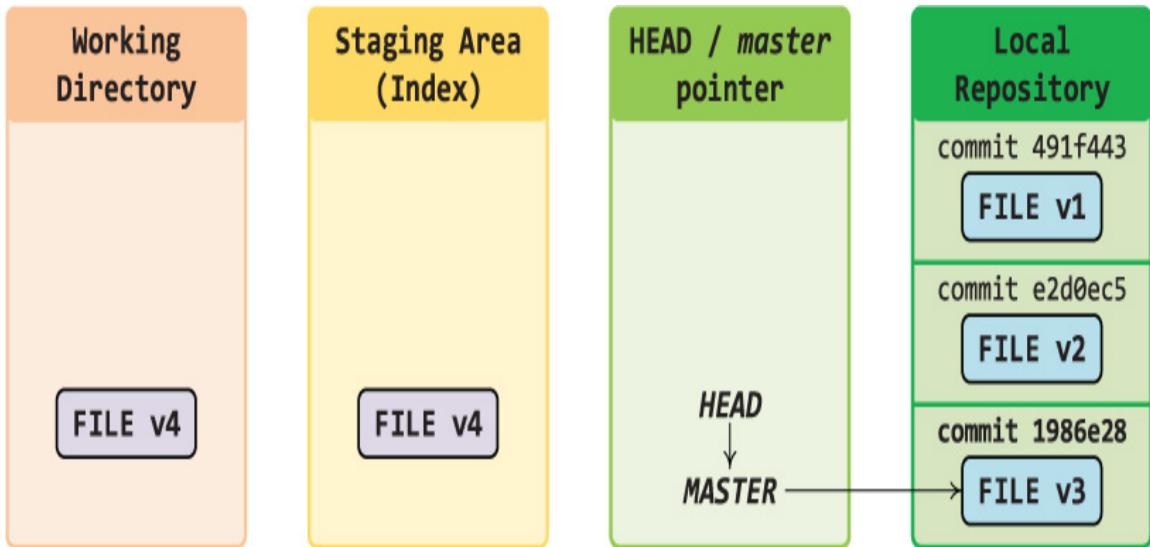
The most frequently used commands to interact with these areas are `git add`, to copy data from the working directory to the staging area, and `git commit`, to copy data from the staging area to a new commit in the repository and update branch pointers.

The command `git reset` is another common command to work with the same three areas. It undoes some or all changes made with the `git add` and `git commit` commands. It does *up to* three operations:

1. Move the branch pointer. “HEAD” is inherently changed since it’s just a pointer to a branch tip.
2. Update the staging area with the contents of the specified snapshot.
3. Update the working directory with the contents of the specified snapshot. You’ll lose all the changes in your working files unless you’ve saved them somewhere else.

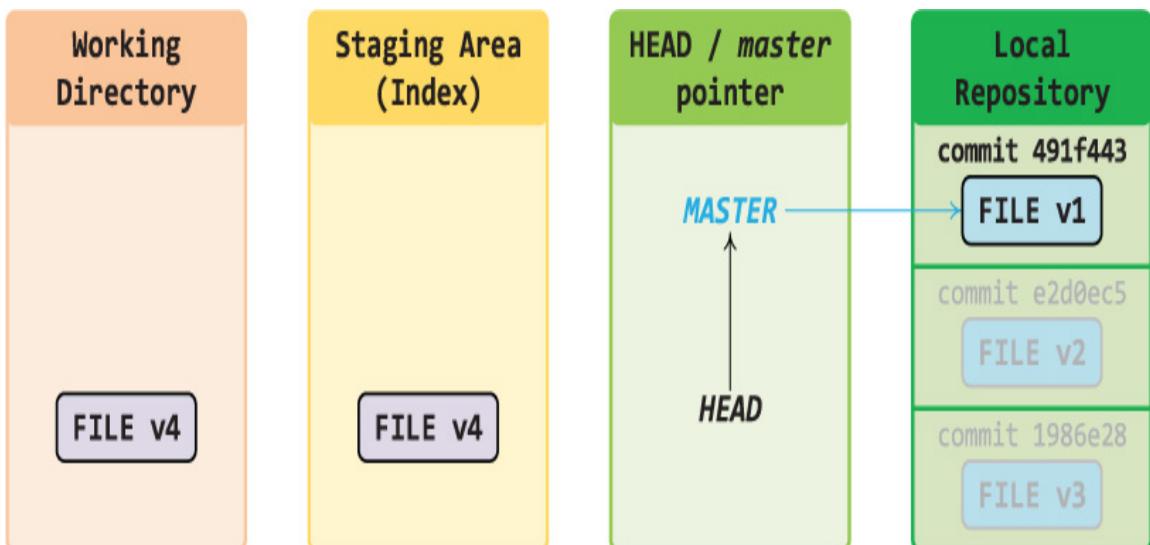
To illustrate Git reset functionality, we’ll use the Git project shown in [Figure 18](#) as an example.

*Figure 18: Git Reset: Before Reset*



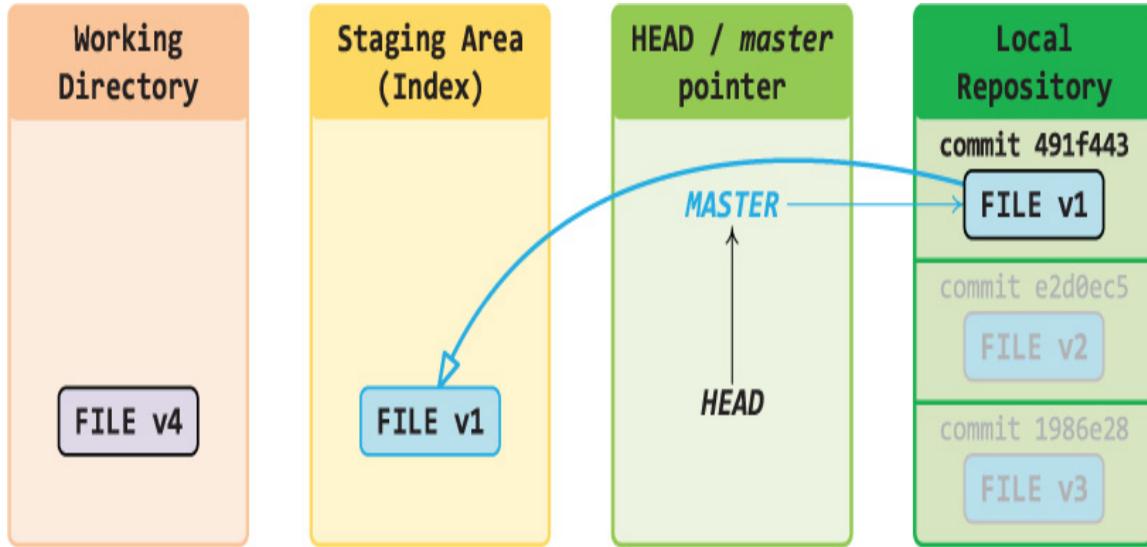
The command `git reset --soft` does only the first operation from the list; it just moves the branch's pointer. [Figure 19](#) shows the project's state after the `git reset --soft 491f443` command (or `git reset --soft HEAD~2`). A new commit to the master branch would have the commit “491f443” as a parent. Commits “e2d0ec5” and “1986e28” would be orphaned and eventually removed by the garbage collection process.

*Figure 19: Git Reset: The “Soft” Option*



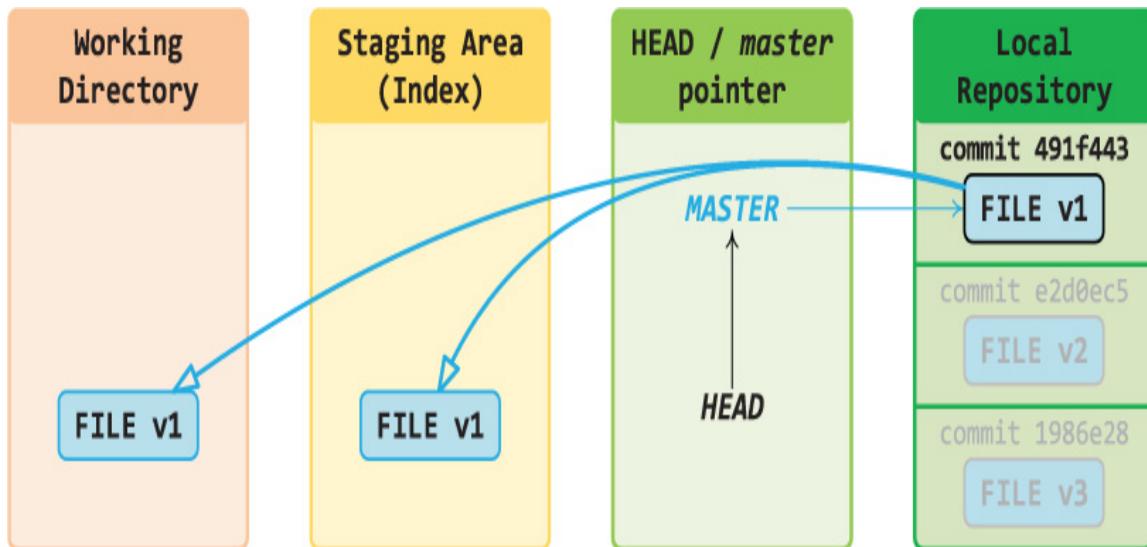
The command `git reset --mixed` (the default, when you execute `git reset`) does the first and second operations, as shown in [Figure 20](#).

*Figure 20: Git Reset: The “Mixed” Option*



The command `git reset --hard` does all three, as shown in [Figure 21](#).

*Figure 21: Git Reset: The “Hard” Option*



Here are some practical notes on using Git resets:

- The command `git reset --soft HEAD~1` moves the branch pointer to the previous commit and effectively undoes the latest commit. It allows you to make corrections and commit again in an equivalent of the `git commit -amend` command.
- The command `git reset HEAD~1` similarly undoes two commands: `git add` and `git commit`.
- To “squash” several commits (for example, three) into one, perform `git reset --soft HEAD~3 + git commit`.
- The command `git reset --hard` is the only destructive option, as it would wipe working files, so use it with care!
- The command `git reset` may be used with the filename argument (except with `--soft` and `--hard` options), in which case it would impact just a single file. For example, `git reset file.txt` would negate the `git add file.txt` command by placing **file.txt**’s content of the previous commit into the staging area.

## 1.10.D git checkout

The `git checkout <branch>` command is used to navigate between the branches. It does two things:

- It makes the “HEAD” point to a target branch. The next commits will be added to the activated branch.
- It updates files in the working directory and in the staging area to match a target branch’s latest snapshot.

Before updating files, Git verifies that they haven’t changed since the last commit to prevent unintentional data loss and proceeds only if no changes are detected (alternatively, use `git stash` if a commit is not ready):

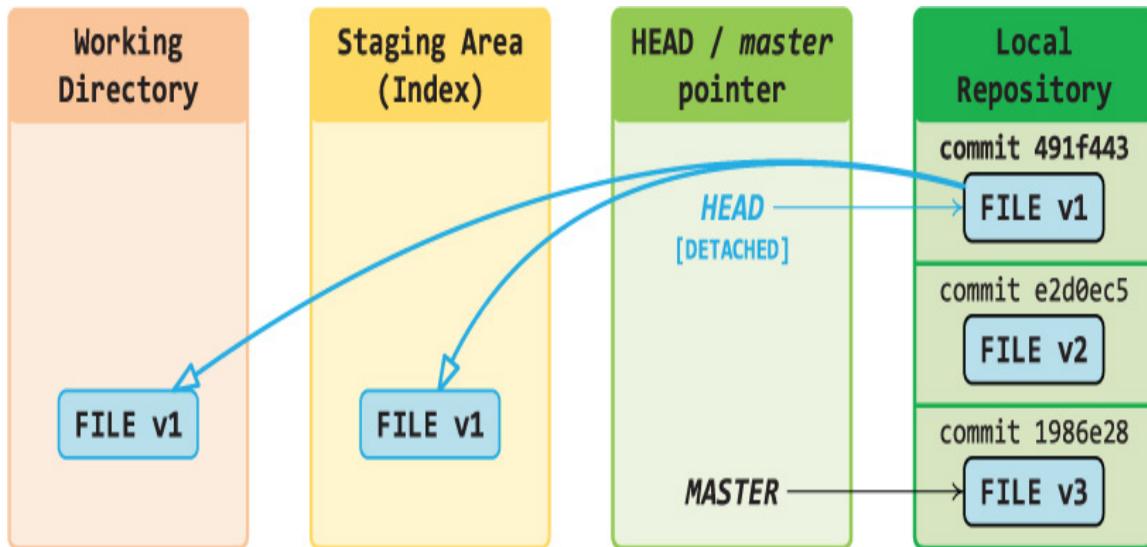
[Click here to view code image](#)

```
$ git checkout bugfix
error: Your local changes to the following files would be
overwritten by checkout:
  file.txt
Please commit your changes or stash them before you switch
branches.
Aborting
```

If you're interested in a specific commit, you may check it out with the `git checkout <commit-hash>` command. In this case, the "HEAD" would become *detached*, as it would not point to any of the branches but rather directly to a specific commit.

Using the same Git project (refer to [Figure 18](#)), the `git checkout HEAD~2` command would result in the state shown in [Figure 22](#).

*Figure 22: Git Checkout: Detached Head*



In this state, you may make commits but they won't be connected to any of your branches and would be lost when you check out back to any regular branch. However, you can save your work if you create a new branch with `git branch`

`<name>` and attach to it with `git checkout <name>`, thus returning to the “attached” state.

## Git Checkout versus Git Reset

If you compare outcomes of `git reset --hard` (in [Figure 21](#)) and `git checkout` (in [Figure 22](#)), you’ll see that they are pretty close, but there are two important differences:

1. The command `git checkout` is safer because it preserves modified content of the working directory, whereas `git reset --hard` just replaces all the files without checking.
2. The command `git reset` moves the branch pointer, whereas `git checkout` moves the “HEAD” pointer and keeps all the branch pointers untouched.

Like `git reset`, the command `git checkout` can work with the individual files (`git checkout [branch|commit] <file>`). Both commands copy the content of the file into the staging area, but `git checkout` also overwrites the file in the working directory. It behaves exactly like `git reset --hard [commit] <file>` would if it was allowed.

## 1.10.E git revert

Sometimes you need to undo some changes but need to maintain revision history, or changes are pushed to a public repository so you cannot undo them with the `reset/checkout` commands.

This is the use case for the `git revert <commit>` command, which creates a new commit that is the opposite of a specified commit. It leaves the files in the same state as if the commit that has been reverted never existed. For example, to undo the last commit, execute `git revert HEAD`.

In the following example, three commits are followed by the revert. As a result, the content of the file is the same as it was before the third commit. Files look the same as if the `git reset --hard fb49f98` command was executed; however, in this case, history is preserved and you can see that there was an additional change that was later undone.

[Click here to view code image](#)

```
$ echo "One line" >> file.txt
$ git commit -am "First"
$ echo "Second line added" >> file.txt
$ git commit -am "Second"
$ echo "Third line added" >> file.txt
$ git commit -am "Third"

$ git log --oneline
f656373 Third
fb49f98 Second
a02e20e First

$ cat file.txt
One line
Second line added
Third line added

$ git revert HEAD
[master eb2397f] Revert "Third"
1 file changed, 1 deletion(-)

$ git log --oneline
eb2397f (HEAD -> master) Revert "Third"
f656373 Third
fb49f98 Second
a02e20e First

$ cat file.txt
One line
Second line added
```

The process may continue further if the second commit was incorrect as well:

[Click here to view code image](#)

```
$ git revert fb49f98
[master 05d4e61] Revert "Second"
 1 file changed, 1 deletion(-)

$ git log --oneline
05d4e61 (HEAD -> master) Revert "Second"
eb2397f Revert "Third"
f656373 Third
fb49f98 Second
a02e20e First

$ cat file.txt
One line
```

Alternatively, a series of commits may be undone with a single command that specifies a commit range. The outcome of the `git revert HEAD~2..HEAD` command is completely equivalent to the two preceding individual reverts.

Like `git checkout`, the `git revert` command has the potential to overwrite files in the working directory, so it will prompt you to commit or stash changes before proceeding.

The `git revert` command cannot be applied to an individual file, only to the whole commit.

## Git Revert versus Git Reset

Functionally, these commands are close, as the content of the working directory could be the same.

With `git revert`, the “HEAD”/branch pointers move forward (with new commit), and the previous history is intact.

With `git reset`, the “HEAD”/branch pointers move back, and history might be wiped.

An argument for the `git revert` command is the ID of the commit you want to *remove*.

An argument for the `git reset` command is the ID of the commit you want to *keep* (and wipe anything thereafter)

from the history).

## 1.11 Explain the concepts of release packaging and dependency management

You created your application and provided a Python source to your users, only to hear back that the software is not working and throws errors about syntax, missing configuration files, wrong libraries, and the web server not running. This is happening because their user environment is different from yours.

The best approach to re-creating the same environment and deploying the same code at the destination system is to use *release packaging* and *dependency management* tools.

### Python Source Distributions

Python source code often consists of multiple files (modules), usually organized into a directory structure. The collection of modules grouped together is called an *import package* (or simply a *package*). Python's packaging system consists of three core elements:

- **Package format:** The simplest format in Python is called a *source distribution* (or *sdist*). It is a compressed archive (a `.tar.gz` file) that has a certain structure and contains one or more packages or modules. It includes package dependencies, if any, and also other metadata, such as the name of the package, the author, and so on. The standard system to build packages in Python is *setuptools*.
- **Package manager:** A program that installs and manages packages. The standard manager in Python is called *pip* (Package Installer for Python).

- **Repository:** A central place where all the packages are stored and where package managers can look for them. In the Python ecosystem, it's <https://pypi.org>.

The `sdist` package contains a copy of the original source code in an archive. Some Python libraries have extensions written in C++/Rust or other languages, and when such modules are packaged into `sdist`, the end user will have to use additional tools (for example, compiler and header files) to install them. Additionally, such installations may take a significant amount of time.

The `wheels` binary distribution format was created to address these problems. No compilation is needed, so the install process is simpler, faster, and more efficient (the download size is smaller). The pip manager always prefers wheels because installation is always faster.

## Library Dependencies

When you write your code in Python, you've likely used at least one external library that you manually added to your environment. When you distribute your code, you need to ensure the following:

- The same library is installed on the target system.
- The library has the same version as used during development.
- The version of the library you're installing won't break other applications that might be relying on a different version of the same library.

One way to address this problem is to create a new *virtual environment*, install all the needed libraries, and then run your code with that environment. The two most common tools to do this are the built-in `venv` module and the `virtualenv` package:

[Click here to view code image](#)

```
~/my_project$ python -V
Python 2.7.17
~/my_project$ python3 -m venv devnet
~/my_project$ source devnet/bin/activate
(devnet) ~/my_project$ python -V
Python 3.6.9
(devnet) ~/my_project$ deactivate
~/my_project$
```

To ensure that all the needed libraries of the correct versions are present in a user environment, you may use the pip manager. Run the `pip freeze > requirements.txt` command in your environment to obtain a complete list of installed packages (clean it up manually, if needed), and then execute the `pip install -r requirements.txt` command on the target system. Other tools to consider for dependency management are *pipenv* and *poetry* (which provide packaging functionality as well).

There are other ways to package your software in a way to address dependencies concerns; for example, you can create a binary executable, embedding the Python interpreter and any other dependencies into a single executable file. This approach is called *freezing* and may be done with, for example, *pyInstaller*.

As one step further, a distribution may be packaged into a *Docker container*, which would include a complete set of software, libraries, and any external programs your software may be dependent on.

## Software Versioning

Your published code most likely is not a single release, and there will be updates and bugfixes. Therefore, you need a way to uniquely identify each version of your software. It may be as simple as some serial numbering (1, 2, 3, and so on), but other methods are more popular.

Date-based versions include calendar information in the release version: for example, Ubuntu 18.04 was released in April 2018. Other options are **<YEAR>.<Month><Day>** and **<YEAR>.<SerialNumber>** (for example, Viptela 18.2 is the second release in 2018).

In some schemes, sequence-based identifiers are used to express the significance of changes between releases.

*Semantic versioning*, currently the best known and most widely adopted version scheme in this category, uses a sequence of three digits (Major.Minor.Patch), an optional pre-release tag, and an optional build meta tag.

In this scheme, risk and functionality are the measures of significance. Breaking changes are indicated by increasing the major number (high risk), new nonbreaking features increment the minor number (medium risk), and all other nonbreaking changes increment the patch number (lowest risk). The presence of a pre-release tag (-alpha, -beta) indicates substantial risk, as does a major number of zero (0.y.z), which is used to indicate a work in progress that may contain any level of potentially breaking changes (highest risk).

Some projects use the major version number to indicate incompatible releases. This method specifies compatibility using a three-part version number: major version; minor version; and patch. The patch number is incremented for minor changes and bugfixes that do not change the software's application programming interface (API). The minor version is incremented for releases that add new but backward-compatible API features, and the major version is incremented for API changes that are not backward-compatible. For example, software that relies on version 2.1.5 of an API is compatible with version 2.2.3, but not necessarily with 3.2.4.

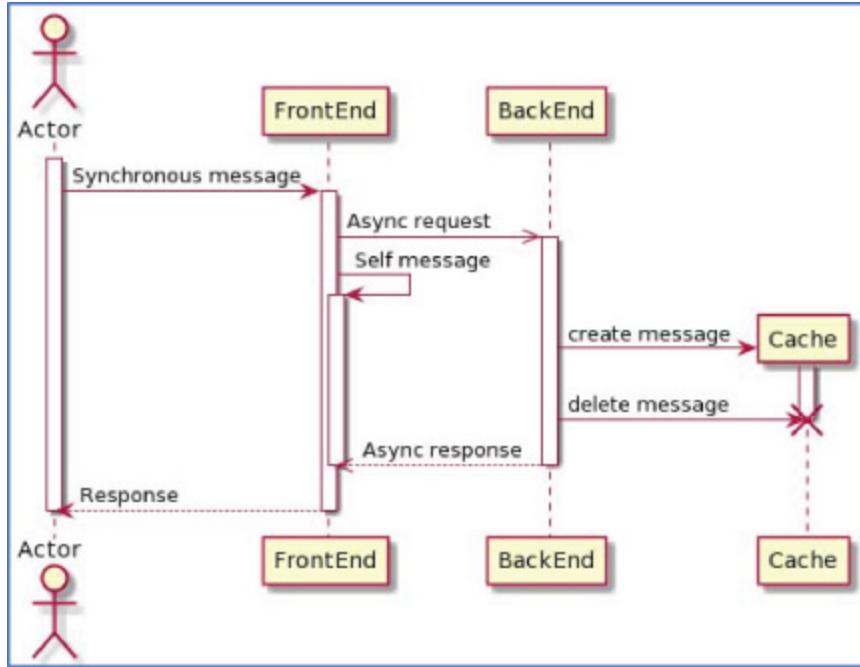
## 1.12 Construct a sequence diagram that includes API calls

Unified Modeling Language (UML) sequence diagrams are a popular modeling solution that visualizes object interactions arranged in time sequence. They depict the participants involved in the scenario and the sequence of messages exchanged between them needed to carry out the functionality of the scenario.

Sequence diagrams are used by software developers due to easier comprehension, a higher level of abstraction, and programming language neutrality for multiple purposes, such as understanding the requirements for a new system; modeling the logic of an application, service, procedure, or operation; documenting an existing process; and so on.

Sequence diagrams show different elements that live simultaneously as parallel vertical lines (called *lifelines*) and the messages exchanged between them as horizontal arrows, in the order in which they occur. The vertical axis represents time progressing down the page. [Figure 23](#) shows a simple sequence diagram.

*Figure 23: Simple Sequence Diagram*



Here are the sequence diagram's components:

- **Participants:** Components that interact with other components
- **Actors:** Specialized external participants that interact with the model (for example, user of the app)
- **Lifeline:** Represents an individual participant in the interaction as a thin rectangle during periods when it operates (*activated*) or as a vertical dashed line when it is *deactivated*.
- **Messages:** Various interactions between participants depicted by arrows. Here are some examples:
  - **Synchronous message:** Call to another participant; execution is stopped until a response is received.
  - **Asynchronous message:** Call to another participant; execution continues without waiting for the response.

- **Return message:** Pass back the information to the caller of a corresponding synchronous/asynchronous message.
  - **Create/delete message:** Instantiate/destroy a new lifeline instance.
  - **Self-message:** The participant interacting with itself.
- **Sequence fragments:** These are represented as a box, called a *combined fragment*, which encloses a portion of the interactions within a sequence diagram. An operator in the top-left corner indicates the type of fragment, which can be any of the following:
- **alt (alternatives):** Multiple fragments, but only the one where the condition is true (IF/ELSEIF/../ELSE logic) is executed.
  - **opt (optional):** The fragment executes only if the condition is true (simple IF logic).
  - **loop:** Execute the fragment multiple times (FOR logic).
  - **par (parallel):** Each fragment is run in parallel.

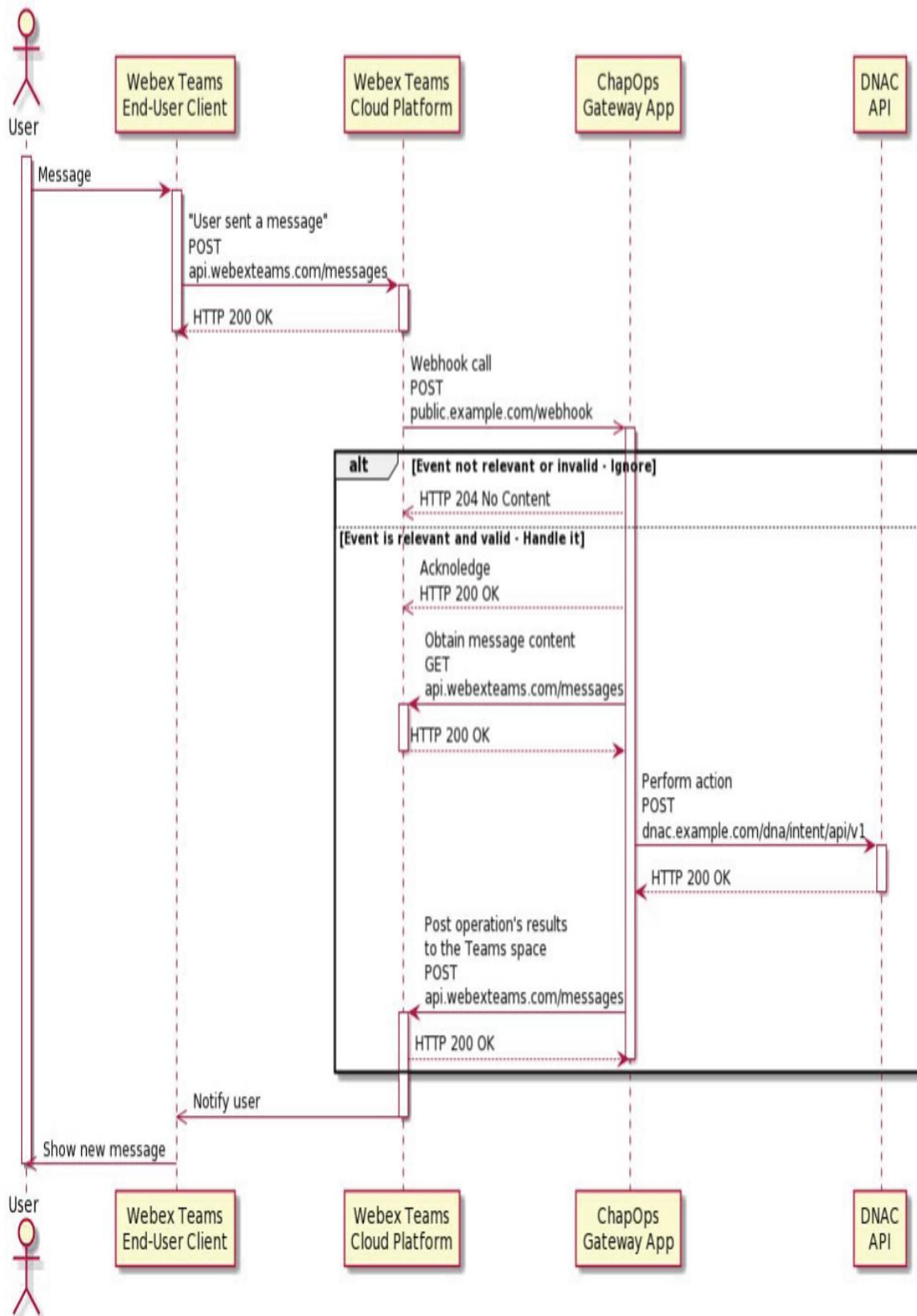
When you're writing an application that uses APIs, visualization of the components and their API interaction allows for a better and clearer understanding of the application architecture. HTTP is a standard protocol used for the API interactions, so all the API calls will be shown as an HTTP Request followed by an HTTP Response message.

If application components wait for a response after making an API call, such interaction is shown as synchronous messages. Data from the API, if any, would be included in the response message.

With the asynchronous interactions, shown as asynchronous messages, APIs immediately respond with the acknowledgment of the request (for example, HTTP 200 OK code), and the main code continues its execution. API data would only be available later, and the caller needs to either poll the API to find out if the request was completed or use a callback mechanism to get notified when the API processing is finished.

[Figure 24](#) shows an example of an API sequence diagram that explains high-level logic for a Webex chatbot application that implements control of the Cisco Digital Network Architecture (DNA) Center (DNAC) infrastructure (for example, get status of switch ports) via chat messages. It closely describes the logic implemented in [Section 3.1](#): Construct API requests to implement ChatOps with Webex API.

*Figure 24: Sequence Diagram with Cisco Webex and Cisco DNA Center API Calls*



To create a sequence diagram, you can work with graphical tools (Visio, Lucidchart, and many others) or use a tool that uses a textual notation to describe UML models and automatically renders the corresponding graphical UML diagram. PlanUML (<https://plantuml.com/sequence-diagram>) is the most well known UML tool in this category.

## 1.13 Chapter 1 Review Questions

**Q1: Which statement is true about the front end and back end in the distributed applications design?**

- A) The front end is a part of an application that executes the application's business logic.
- B) Some of the back-end functionality may be offloaded to the front end.
- C) The front end may be written in HTML, JavaScript, or Python.
- D) Load balancers can only distribute the load equally among back-end servers.

**Q2: Which statement is true about modular application design?**

- A) Modules contain everything necessary to execute a certain aspect of the application functionality and therefore are hard to replace with another module.
- B) A module may be easily replaced with another module as soon as their “implementation” sections are similar.
- C) Project work may be broken down into smaller independent projects.
- D) Modular code is more complicated to read and maintain compared to nonmodular.

**Q3: Which statement is true about scalable application design?**

- A) Scaling up is a simple way to increase the reliability of a system.
- B) Monolithic applications may be scaled out as easily as microservices.
- C) Implementing load balancing is one of the methods to scale applications horizontally.
- D) Performance scalability is the only way to achieve application scalability.

**Q4: Which statement is true about high availability in the application design?**

- A) Clustered applications continue to provide service even when some of their components fail.
- B) Availability is expressed as percentage of downtime over a period of time.
- C) Stateful data flows help achieve higher availability.
- D) As soon as the service is operational, there's no need to worry about its internal components.

**Q5: Which statement is true about resilience in the application design?**

- A) Using timeouts helps improve resilience, with higher timeout resulting in better resiliency.
- B) Parameter checking in functions and objects improves application resilience.
- C) Resilience tries to detect and work around the failure by replacing a broken component.
- D) If a request fails on a first attempt, the application should fail fast to prevent any further waste of system resources.

**Q6: Which statement is true about latency in the application design?**

- A) Latency is not a concern for the distributed microservices applications for which components are running within the same data center.
- B) End-user latency is based on the physical limitations (distance, speed of light) and therefore cannot be improved.
- C) For a better user experience, it is more optimal to load data in small blocks instead of doing it in one big chunk.
- D) Using UDP instead of TCP for video-conferencing data may result in unpredictable conversation latency and a bad user experience.

**Q7: Which statement is true about rate-limiting?**

- A) In REST API calls, an HTTP response with code 329 and a redirect URL indicates that the API is being overused.
- B) A rate-limiting mechanism may be used in processes other than REST API calls.
- C) An exponential token is a rate-limiting mechanism where the number of tokens in the bucket is doubled after each successful attempt.
- D) Disconnecting a user after an unsuccessful login attempt is an example of rate-limiting.

**Q8: Which statement is true about maintainability in the application design and implementation?**

- A) Automated testing helps achieve better software code.
- B) High-quality code requires more effort and therefore is harder to maintain.

- C) Nonmodular code is better documented and is easier to maintain.
- D) Modules are independent of each other, so it is a good idea to use the most suitable tools, libraries, and naming conventions when working on each of them.

Q9: Which statement is true about observability in the application design?

- A) Logs, metrics, and release packaging are the three pillars of observability.
- B) Distributed tracing brings visibility into the lifetime of a request across several systems.
- C) Excessively detailed logging puts an extra load on the application, so it should be avoided.
- D) Observability is just another name for traditional monitoring.

Q10: Which statement is true about using event logs to diagnose problems with an application?

- A) If an application is having problems but the event logs indicate there were no crashes, then the problem is external to the application and should be addressed by other teams.
- B) The downside of Syslog is that it does not include a timestamp in logs. It is resolved in Syslog-*ng*.
- C) It is easier to troubleshoot problems when logs contain enough context around events.
- D) If a Python application crashes with an exception, adding the “try / except” wrapping around the failed operation is enough to address the problem.

Q11: Which is *not* a primary factor to be considered when selecting a database system for the application?

- A) The kinds of data that will be stored
- B) Simplicity of the installation process
- C) The ability to perform complex queries
- D) Data model flexibility

Q12: Which statement is *not* true about relational databases?

- A) Transactions always result in a valid state of the database.
- B) All operations in a transaction succeed, or every operation is rolled back.
- C) Relational databases focus on the consistency and availability of data.
- D) A relational database is not a good choice for complex querying.

Q13: A manufacturing plant is equipped with various types of specialized sensors that send their readings to a custom application. What type of database is most suitable to store sensor data and allow real-time and historical data analytics?

- A) Key-value database
- B) Relational database
- C) Graph database
- D) Time-series database

Q14: Which statement is true about the monolithic architectural pattern?

- A) Monolithic applications are self-contained and have no dependencies on other applications.
- B) In monolithic applications, the user interface, business logic, and database are combined into a single executable.

- C) Monolithic applications are hard to scale both vertically and horizontally.
- D) Monolithic applications are a thing of the past, and creating them is a sign of poor project planning.

**Q15: Which statement is true about the service-oriented architecture (SOA) pattern?**

- A) Services are reusable blocks of code that are called directly from within the application code.
- B) SOA data models, protocols, and technologies are standardized, which allows services to call other services with ease.
- C) Services can be defined in business rather than technical SOA terms, which improves collaboration between business and IT.
- D) Services are stateful to allow visibility into the service state for the consumers.

**Q16: Which statement is true about the microservices architectural pattern?**

- A) Microservices are much smaller but more complicated compared to SOA.
- B) Microservices applications are easier to develop, maintain, and operate.
- C) Microservices are best developed when all teams use the same programming language and tools.
- D) With microservices, there is significant communication happening between different components of the system, which adds overhead and latency.

**Q17: Which statement is true about the event-driven architecture (EDA)?**

- A) Reactions to events are immediate.

- B) The event broker topology is better suited for complex situations than the mediator one.
- C) EDA is the most scalable but at the cost of higher implementation and maintenance complexity.
- D) The event broker provides details about event creators to event processors in case they need to obtain more information about events.

**Q18: Which statement is true about the Git branch merging?**

- A) HEAD should be pointing to the correct receiving branch when you issue the `merge` command.
- B) Content of the staging area is added to the receiving branch before the merge.
- C) Rebase operation retains all changes to and history of the merged branch.
- D) If the “fast-forward” merge is not possible, Git requires a manual intervention to resolve a conflict.

**Q19: Which statement is true about the Git merge conflict resolution?**

- A) The `git merge status` command shows extensive details about the merge conflict.
- B) If the merge completed successfully, then no further verification is needed.
- C) A merge conflict can either be resolved manually in a text editor or the merge may be rolled back.
- D) Binary files cannot be edited in a text editor, so the only way to resolve a merge conflict for them is to roll back.

**Q20: Which statement is true about the `git reset` command?**

- A) `git reset` is a safe command; it is always possible to roll back.

- B) `git reset` can be used to clean up commit history and “squash” several commits into one.
- C) `git reset HEAD^2` and `git reset HEAD~2` do the same action.
- D) `git reset --hard filename.txt` may be used to restore a previous version of a single file.

Q21: Which statement is true about the `git checkout` command?

- A) `git checkout` is a safe command; it is always possible to roll back.
- B) Commits can be made in the “Detached HEAD” state, but they would be eventually lost since they won’t belong to any branch.
- C) `git checkout` is a safe command because it only moves the HEAD pointer but does not change the content of the working area.
- D) `git checkout --hard filename.txt` may be used to restore a previous version of a single file.

Q22: Which statement is true about the `git revert` command?

- A) The `git revert HEAD` command erases the latest commit as if it never happened.
- B) Executing `git revert HEAD` twice or `git reset --hard HEAD~2` once produces the same content in the working area.
- C) Unlike `git reset`, the `git revert` command preserves the full commit history of the project.
- D) `git revert filename.txt` may be used to restore a previous version of a single file.

Q23: The Python code works on a developer’s machine but fails in the client’s newly created virtual environment. What could be a reason?

- A) Source code is distributed as a package and installed with pip.
- B) The developer forgot to create a list of library dependencies and include it with the code.
- C) Code is distributed as text, not a compiled binary.
- D) Python was not copied into a fresh virtual environment when it was created.

**Q24: Which statement is true about sequence diagrams?**

- A) The horizontal axis shows time progression.
- B) When participants interact with each other, their lifelines intersect.
- C) Synchronous messages are calls to other participants, and a participant's call to itself is called asynchronous.
- D) API calls are visualized as HTTP Request/Response sequences.

---

## *2. Using APIs*

---

The two types of API call processing are synchronous and asynchronous. With the synchronous calls, clients make an API request and then wait for the response. Once the request has been completed, the API sends back the HTTP response. The response will indicate whether the execution was successful with HTTP status codes and include any data that needs to be returned. If the request takes a long time to complete (or the API fails to send a response at all for any reason), the requesting app may time out or even hang forever.

An alternative is to use the asynchronous approach. In this case, the API responds right away, notifying the client that it has received the request for processing. No data or success status is included because the requested operation has not yet been completed. The client can only obtain this information later, either by periodical API polling or by providing the callback URL to the API in the request (if supported). This callback URL is then contacted by the API when processing is complete.

The asynchronous approach allows parallel request execution; however, tasks do not necessarily complete in the order they were requested, and an application should be able to handle it properly.

### **2.1 Implement robust REST API error handling for timeouts and rate limits**

REST API is a service relying on a network. We expect networks to be reliable, but problems do happen, and properly written applications should be able to handle these problems correctly.

A REST API call consists of several steps:

- Create the request at the client side.
- Transfer the request over the network to the server.
- Execute the request at the server.
- Transfer the response over the network to the client.

In this sequence, errors may happen, and they can be divided into two types: network errors and API errors, indicated by HTTP response codes.

When a request cannot reach the destination server or a response is lost on the way back to a requesting client, it's a network error. Most often it will be expressed as the *connection timeout* or *connection refused* error due to host/port unreachable, typically due to a network or host outage or some firewall rules.

Network timeouts are most common, and it's not a good idea to assume they won't happen or they won't have an impact. Timeouts may happen for two main reasons: the network is slow or the server is busy, so they need some extra time, or the network/server is down and will never respond. You may tune the timeout value in the first case, but increasing it in the second case would only increase delays and worsen the user experience.

Network problems are often transient, so one way to handle a timeout is to retry the same request again after some pause. However, if the retry interval value is low, this approach may result in an increased load on the network and server. A more advanced approach is to use an

exponential backoff timer. It uses three parameters: *connection timeout*, *initial wait time*, and *wait reset time*. The logic is simple: the wait time between calls starts with the *initial wait time* but it's doubled after each timeout until it reaches *wait reset time*, when it's reset back to the initial value (to avoid very long timeouts).

When implementing such retry functionality, it is important to be aware of REST API *idempotency*.

GET, PUT, DELETE, and HEAD HTTP methods are all *idempotent*, which means they can be called many times with the same outcome. It would not matter if the method is called only once or ten times, the result should be the same.

However, HTTP POST, when used to create a new object, is not idempotent, so calling it several times may result in several new objects created. If you handle timeouts with retries, an extra check might be needed to confirm the object does not exist before repeating the POST call.

An *HTTP response code* is used to indicate a problem if a request was received by the server but it could not be processed for some reason and the server needs to deliver a notification back to the client. There are more than 70 HTTP status codes, but most API providers use a small subset of them (for example, Netflix uses just nine in total).

Response codes are classified into five categories, based on the first digit of the code:

- 1xx Informational
- 2xx Success
- 3xx Redirection
- 4xx Client error
- 5xx Server error

Many APIs have built-in denial-of-service functionality that limits the number of requests per time interval, or they use throttling for some customer tiers (for example, one request per second for a free tier and unlimited for the paid one). When the allowed rate of requests is exceeded, the API server responds with an HTTP 429 code (“Too Many Requests”).

Unfortunately, there’re no established standards on how to implement this. One common approach is to use “X-Rate-Limit-Limit,” “X-Rate-Limit-Remaining,” and “X-Rate-Limit-Reset” response headers (for example, Twitter and GitHub), but values may have different meanings depending on the implementation.

The Webex API is another API that implements rate limiting of requests. If it responds with the HTTP 429 “Too Many Requests” response, an application needs to back off and retry the request after the duration specified in the “Retry-After” header of the response.

You can use two approaches to handle rate-limiting restrictions for a specific API:

- *Perform requests as normal.* If an HTTP 429 response is received, make a pause (the duration may be hinted at in the response, depending on API) and then continue with a retry.
- Find out API restrictions and limit the rate of your requests, thus preventing an HTTP 429 error.

The following partial code provides an example of a function that interacts with an API (for example, Webex) while handling network timeouts and rate-limiting errors:

[Click here to view code image](#)

```
def API_Read_Safe (url, timeout = 5, attempts = 3):  
    """ Read API URL, handle HTTP 429 and timeout errors """
```

```
tries = 0
while (tries < attempts):
    tries += 1
    try:
        # read target URL
        response = requests.get(url, headers=headers,
timeout = timeout)

        # if successful, return the response
        if response.ok:
            print ("Completed successfully")
            return response

        # If response is not "ok", handle error code 429:
        # - wait "Retry-After" seconds or 1 sec, if not
present
        # - retry
        if response.status_code == 429:
            try:
                retry_after =
int(response.headers.get('Retry-After'))
            except Exception:
                retry_after = 1

                print (f'Retry after {retry_after} seconds')
                sleep(retry_after)
                continue

            # if not OK and not 429, then it's something
unrecoverable
            print(f'HTTP error: {response.status_code}')
            response.raise_for_status()

## if timeout error, retry. If any other error, leave it
unhandled
        except requests.exceptions.ConnectTimeout:
            print ("Timeout, retry")
            continue

# return "None" if unsuccessful after all attempts
print ("Unable to get any response")
return None
```

## 2.2 Implement control flow of consumer code for unrecoverable REST API errors

Network and REST API error handling in your application is one of the important development tasks. Good REST API error flow control should start with separation of errors into the following categories:

- **Unrecoverable errors:** These typically require user interaction for continuation.
- **Recoverable errors:** These require an application to wait and retry the process again without user interaction.

When an application receives a REST API response that falls into the unrecoverable error category, it should collect and display/store diagnostic information (response status code, headers, and body) for the troubleshooting process. No programmatic workaround is available.

However, if the response status code indicates a recoverable error, program logic should be implemented to act accordingly.

Here are some examples of unrecoverable errors:

- **400 Bad Request:** You did not send a valid request, and the remote side is refusing to process further. You can get more information from HTTP headers or the body of the response.
- **401 Unauthorized:** Authentication is required (API key, API token, or username and password). If the API was previously accessible, be aware that keys and tokens can expire, and usernames and passwords can be changed.

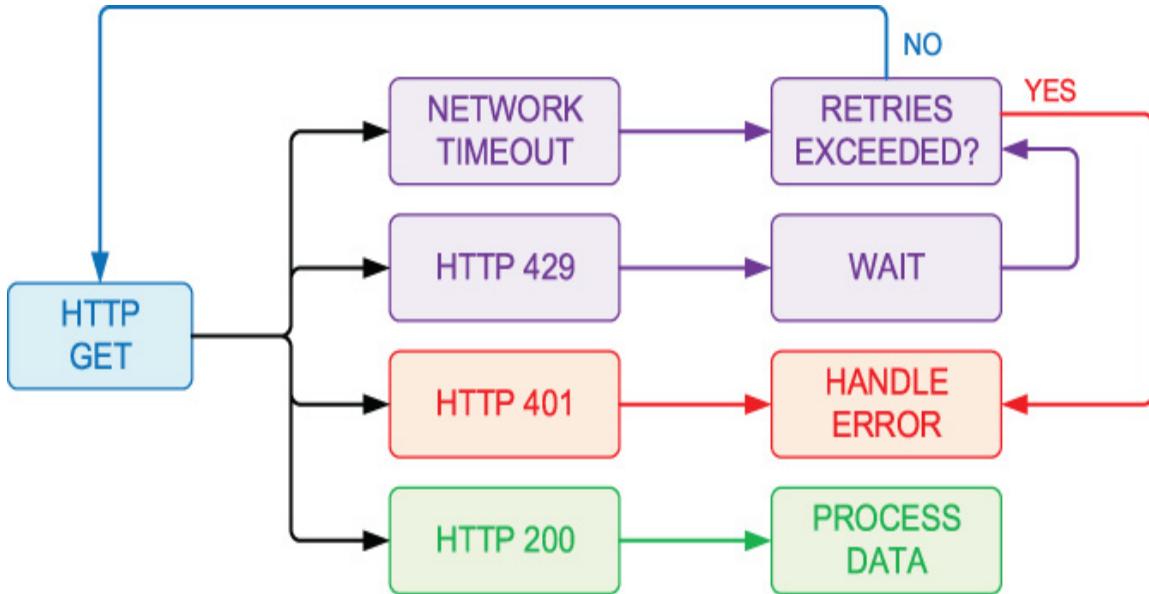
- **403 Forbidden:** You are authenticated and authorized but do not have the permission to access the resource.

Here are some examples of recoverable errors:

- **405 Method Not Allowed:** Indicates an incorrect method for accessing a resource. The response must include the HTTP “Allow” header that lists valid methods.
- **408 Timeout:** Indicates a timeout sending a request to the server (for example, when POSTing large file). Your requests should be faster if you want the server to accept it.
- **429 Too Many Requests:** Indicates too many requests sent to the server. You should wait for some time before you issue a new one. The Cisco Webex REST API responds with the additional “Retry-After” HTTP header, which holds a value (in seconds) that the API consumer must wait before the rate-limit counter resets.

[Figure 25](#) shows an example of the control flow in a sample application.

*Figure 25: Control Flow for REST API Error Handling*



## 2.3 Identify ways to optimize API usage through HTTP cache controls

When API load increases due to growth of user requests and application data, hardware requirements increase as well. There are a few ways to address this: you can scale hardware (horizontally or vertically) or optimize your application, or both. The goals of the application optimizations are as follows:

- Improve response time in user-interactive applications.
- Decrease the required bandwidth.
- Decrease processing times.
- Lower resource utilization.

These goals may be achieved with pagination (see [Section 2.4](#)), caching, and compression.

## HTTP Caching

HTTP caching is a technique that stores copies of web resources and serves them back when requested. Typically limited to caching responses to “GET” requests, HTTP caches intercept these requests and return local copies instead of redownloading resources from originating servers.

Caching may be implemented at different places: at the client (for example, in a web browser), locally as an organization’s proxy server, on the server side as a standalone reverse proxy, or as a part of an API gateway.

Caching reduces the amount of network traffic and the load of web servers. It also improves the user experience since HTTP requests may be completed in less time. However, resources may change, and it is important to cache them properly: only until they change, not longer. To address this, caches use the concept of “freshness.”

A stored resource is considered “fresh” if it may be served directly from the cache. As web servers cannot contact caches and clients when a resource changes, they simply indicate an expiration time for the provided content. Before this expiration time, the resource is “fresh,” and after the expiration time, the resource is “stale.”

“Stale” content is not ignored and might still be reused, but it needs to be validated first. During validation, a check is performed on the original resource to verify that the cached copy is still current. Such checks usually involve only HTTP headers (HTTP “HEAD” request) and no data, so they are “cheap” compared to retrieving the whole resource.

Validation checks increase the response time; however, they are more reliable than a time-based mechanism because they ensure data is up to date.

Website content may change when the cache is still “fresh,” so some servers may want to force validation on every request to avoid inconsistencies. One way to achieve that is

to set the explicit expiration time in the past and thus indicate that the response is already stale when it's received.

Validation can only happen if the origin server previously provided a *validator* for a resource, which is some value that describes a specific version of a resource. There are two types of validators:

- The date of last modification of the document, provided in the "Last-Modified" HTTP header.
- Some string that uniquely identifies a version of a resource, called the Entity Tag (ETag) and provided in the "ETag" HTTP header. It is not specified how ETag values are generated, so it may be a hash of the content, a hash of the last modification timestamp, or just a revision number.

Validators may be *strong* when they guarantee that two resources are completely byte-to-byte identical to each other. With the *weak* validators, two versions of the document are considered as identical if the content is equivalent (for example, same web page but with the different ads on it). ETags may be used as strong validators (by default) or as weak ones (prefixed with "W/"; for example, W/"5fbfb1b822b0f7").

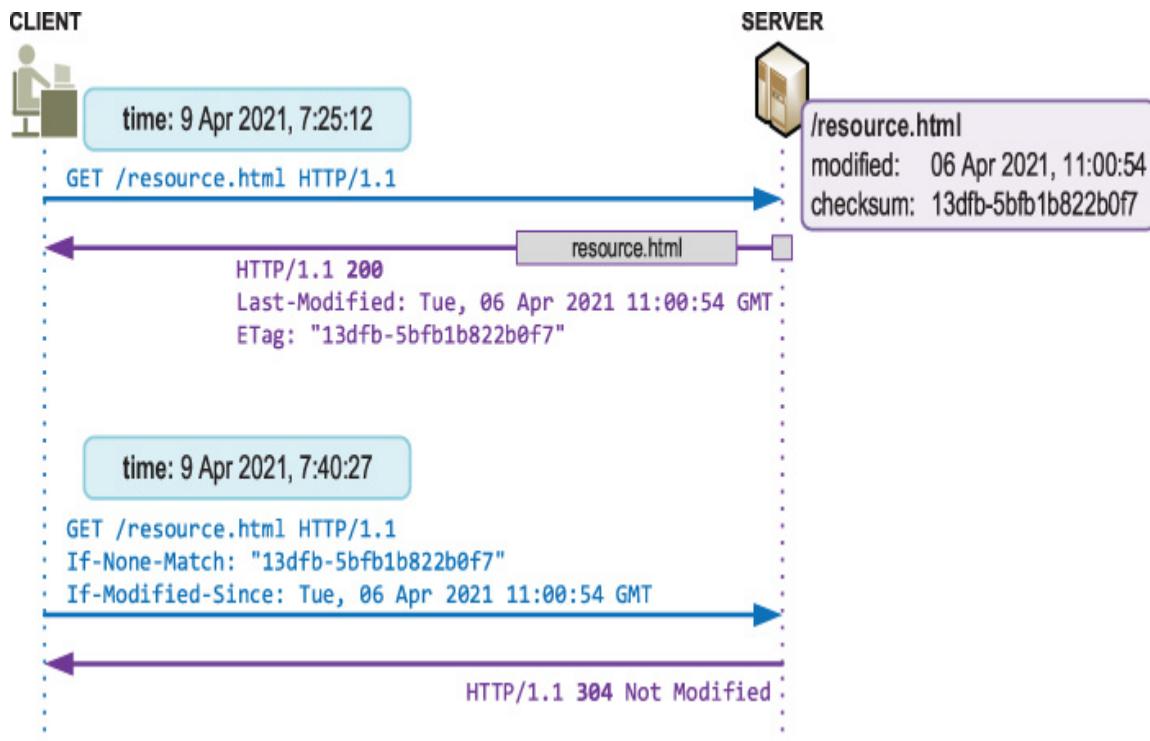
## **HTTP Conditional Requests**

With HTTP *conditional requests*, servers' responses are based on the result of a comparison between the current version of a resource with validators included in a request. Conditional requests enable an efficient validation process but have other uses as well: to verify the integrity of a document (for example, when resuming a download) or, when clients act in parallel, to prevent accidental deletion or overwriting of one version of a document with another.

Special headers are used to specify preconditions in requests, and their logic depends on the HTTP request type: safe (read) or unsafe (write), as presented in [Figure 26](#). In these examples, the “Last-Modified” and “ETag” headers are already known from previous requests.

With safe methods (GET and HEAD), conditions are used to fetch a resource *only when needed*.

*Figure 26: HTTP Conditional Requests: Fetch the Resource*



The following headers specify a condition for the request:

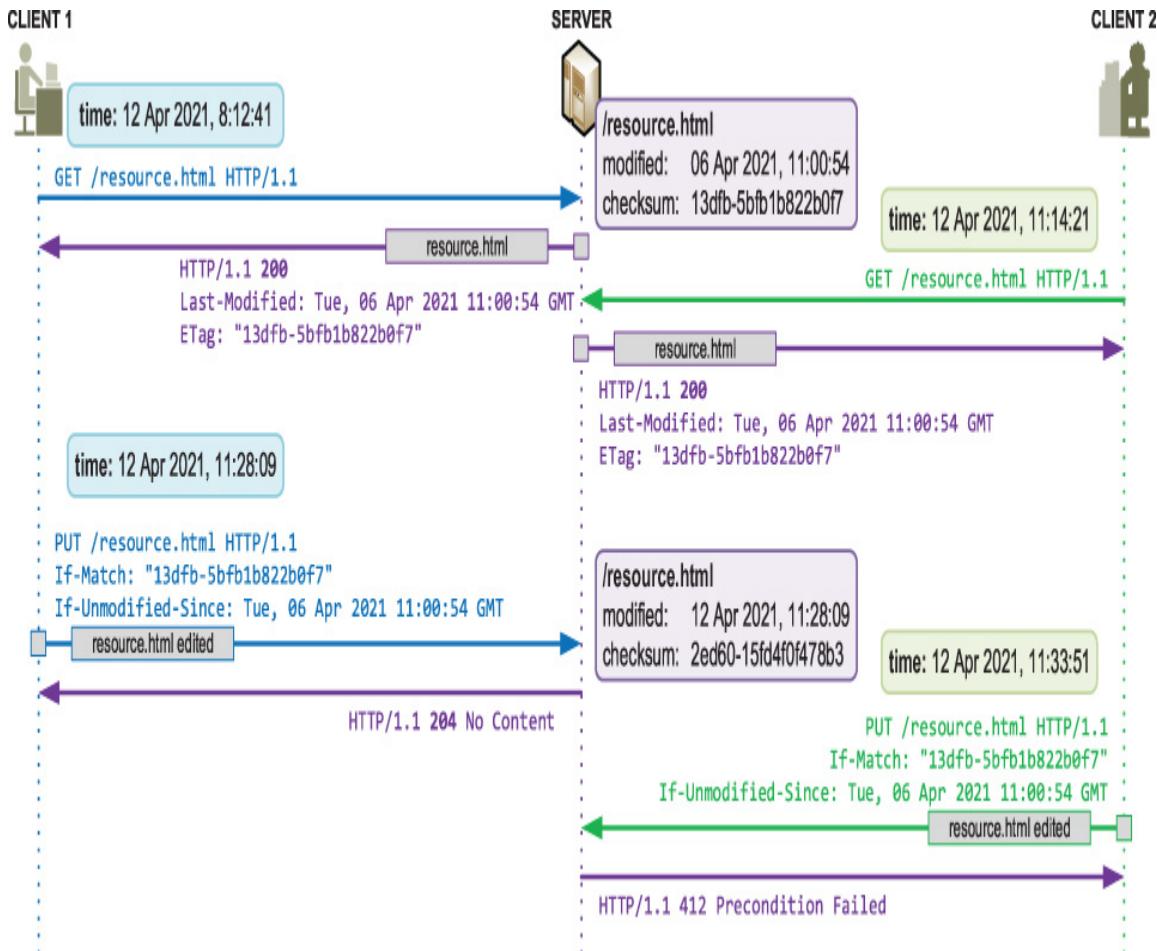
- **“If-None-Match”**: Set to previously known “ETag” value. The condition is true if the ETag of the resource on a server is different from the one given in this header. Both strong and weak ETags may be used.
- **“If-Modified-Since”**: Set to previously known “Last-Modified” value. The condition is true if the date of the

resource on a server is more recent than the one listed in this header.

If the condition is true (that is, the resource has changed), then the server responds with the HTTP 200 code and the complete resource (for the GET method). If the resource hasn't changed, the server must respond with the HTTP status code 304 "Not Modified" and an empty body.

With unsafe methods (PUT and DELETE), conditions are used to allow modifications to a resource *only when it's still the same and hasn't changed since the last interaction* (see Figure 27).

*Figure 27: HTTP Conditional Requests: Modify the Resource*



The following headers specify a condition for the request:

- **“If-Match”**: Set to previously known “ETag” value. The condition is true if the ETag of the resource on a server is equal to the one supplied in this header. Only a strong ETag may be used, as weak tags never match under this comparison.
- **“If-Unmodified-Since”**: Set to previously known “Last-Modified” value. The condition is true if the date of the resource on a server is the same or older than the one listed in this header.

If the condition is true (that is, the resource has not changed), then the server proceeds with the update as normal. If the condition is not met, then the change is rejected with a 412 “Precondition Failed” error.

## Caching Controls

Sometimes, caching may interfere with the application behavior, and the server administrator may wish to provide explicit directions to cache engines. The *Cache-Control* HTTP header is used for this purpose. Some of the most often used options are:

- **no-store**: The response may not be stored in any cache; disable caching.
- **no-cache**: Allow caching, but stored response *must* be validated first before using it.
- **max-age**: Time in seconds a cached copy is valid. `max-age=0` is similar to `no-cache`.
- **public**: Can be cached by anyone.
- **private**: Only browser/client may cache.

Example: `Cache-Control: no-cache, max-age=0`

## **HTTP Data Compression**

By using HTTP compression, less data needs to be transferred over the network, which results in faster response times and lower bandwidth utilization.

When making an HTTP request, a client may include an HTTP “Accept-Encoding” header to indicate a list of compression algorithms that the client supports. Common examples are compress, deflate, gzip, and bzip2.

Compression is not always accepted by the server, as it may cause issues with antivirus software, next-gen firewalls, or proxies, and/or it may violate company policies.

When the server receives a request that indicates compression support, it can honor that request or ignore it and send an uncompressed response. If one of the client’s compression methods is supported, and the server decides to use it to compress the response, the server must include it in the “Content-Encoding” HTTP header.

## **2.4 Construct an application that consumes a REST API that supports pagination**

When a client makes a REST API request, the response can be very large (for example, event log). Quite often, the client is not interested in all the data. For example, it might only need log entries for yesterday or the 50 latest chat messages. Pagination is a method to split resulting data into manageable chunks to allow better handling of large data sets and faster request processing.

The main reasons for using pagination in REST APIs are as follows:

- **To improve response times and end-user experience:** Because paginated responses are much smaller, they can be handled by the server and the network faster, and faster responses provide better user experiences.
- **To save resources:** Providing small responses demands much less compute and network resources.

One of the simplest types of pagination is the “Offset/Page”-based pagination. With this method, the client supplies the data offset and data size as query parameters. Here are two examples:

- **[https://example.com/events?](https://example.com/events?offset=100&limit=20)**  
**offset=100&limit=20:** Returns 20 entries, starting with 100th
- **[https://example.com/events?](https://example.com/events?page=6&per_page=20)**  
**page=6&per\_page=20:** Same logic, different syntax

Typically, if offset or limit parameters are omitted, the server will use defaults (for example, offset=0 and limit=50).

Offset-based pagination is simple to implement and to use; however, it has its downsides:

- Adding or deleting entries (known as *page drift*) between calls may cause confusion. Some results may be skipped over (for example, some item is deleted and now the first item on a new page moved to the previous) and some may be included more than once (when items are inserted).
- It stumbles when dealing with large offset values. With an offset setting of 1000000, for example, the API would have to scan through a million database entries and then discard them.

Another approach for implementing REST API pagination is the *cursor*- or *keyset*-based method, which addresses drawbacks of the previous type.

This method uses some “key” value to indicate a starting item for the next set of data. The value of the key is based on the results from the previous page, and the next call will provide results relevant to a given pointer. Keys depend on the specific API implementation (for example, for Webex API). Here are two examples:

- <https://webexapis.com/v1/messages?roomId={room}&max=5&before=2020-07-07T12:34:00>
- <https://webexapis.com/v1/messages?roomId={room}&max=5&beforeMessage=ZBCdefXYZ>

You can build links to each page manually, but REST APIs have a way to simplify this process by providing additional information in HTTP headers or responses, such as direct links to first/previous/next/past page(s) as well as some general aids.

There are several methods of sending this information with the REST API response.

One method is using the HTTP “Link” header, which provides a list of resources (URLs) and the type of relationship between a resource and the current page, defined in the “rel” parameter. The client can use links and relationships to navigate through the data.

For example, <https://example.com/events?offset=100&limit=20> call may return these headers:

[Click here to view code image](#)

```
Link: <https://example.com/events?offset=100&limit=20>;  
rel="self",  
<https://example.com/events?offset=80&limit=20>; rel="prev",  
<https://example.com/events?offset=120&limit=20>; rel="next",
```

```
<https://example.com/events?offset=0&limit=20>; rel="first",
<https://example.com/events?offset=220&limit=20>; rel="last"
```

This method is used by the Cisco Webex API. The following sample code uses Webex pagination functionality to print all the messages in the given room, obtaining content in small batches (the *requests* library automatically parses “Link” headers and makes them easily consumable as the “links” property of the Response object):

[Click here to view code image](#)

```
import requests

# Use your own access token
bearer_token = 'NjVkJMmIx...cae0e10f'

# Use your favorite room ID.
# Find it by calling https://webexapis.com/v1/rooms with proper
authentication
room_id="Y2lzY29z...YTI5Zjg5"

messages_per_page = 10
base_url = 'https://webexapis.com/v1'
headers = {
    "Accept": "application/json",
    "Content-Type": "application/json",
    "Authorization": f"Bearer {bearer_token}",
}

def main():

    #URL to read first messages in the room
    url = f'{base_url}/messages?roomId={room_id}&max={messages_per_page}'

    while True:
        #read next batch of messages with primitive error
        checking
            response = requests.get(url, headers=headers, timeout=3)
            response.raise_for_status()

            #iterate through the received messages and print content
            for msg in response.json().get("items"):
                msg_text = msg.get("text") if msg.get("text") else "
<image>"
```

```

        print (f'Posted by {msg["personEmail"][:20]}:
{msg_text[0:60]})')

        # check the content of the "Link" header.
        # if present, use it's value as URL for the next
iteration
        # if not, break the loop (header won't be present in the
last batch)
        if response.links:
            url=response.links["next"]["url"]
        else:
            break;

if __name__ == "__main__":
    main()

```

Another method is to include metadata directly into the REST API response body. This metadata may include information such as the current page number, total number of pages, number of records per page, and the total number of records, along with links to the first, last, previous, and next records.

This functionality is used by Cisco Firewall Management Center (FMC)/Firepower Device Manager (FDM) APIs. An example of a Cisco FMC response to the object listing API request is shown next:

[Click here to view code image](#)

```
{
  "links": {
    "self": "https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=6&limit=2"
  },
  "items": [
    {
      "name": "database",
      "id": "32",
      "links": {
        "self": "https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories/32"
      }
    }
  ]
}
```

```
6d9ed49b625f/object/applicationcategories/32"
    },
    "type": "ApplicationCategory"
},
{
    "name": "download manager",
    "id": "45",
    "links": {
        "self":
"https://fmcrestapisandbox.cisco.com/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories/45"
    },
    "type": "ApplicationCategory"
}
],
"https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=4&limit=2",
"https://fmcrestapisandbox.cisco.com/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=2&limit=2",
"https://fmcrestapisandbox.cisco.com/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=0&limit=2"
],
    "next": [
"https://fmcrestapisandbox.cisco.com/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=8&limit=2",
"https://fmcrestapisandbox.cisco.com/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=10&limit=2",
"https://fmcrestapisandbox.cisco.com/api/fmc\_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=12&limit=2",
```

```
"https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?  
offset=14&limit=2"  
    ],  
    "pages": 21  
}  
}
```

The example includes the following pagination details:

- The current page starts with the sixth element of the list (offset = 6).
- A limit of two elements per page.
- The total number of elements is 42.
- The total number of pages is 21.
- Direct links to three previous pages (with the offset of 4, 2, and 0, respectively).
- Direct links to four next pages (with the offset of 8, 10, 12, and 14, respectively).

The following sample code uses FDM API pagination functionality to list all the network objects, obtaining content in small batches (authorization is out of scope for this exercise):

[Click here to view code image](#)

```
import requests  
from auth_token import get_token  
  
page_size = 3  
  
def main():  
  
    # Using FirePower Threat Defense REST API sandbox  
    base_url = "https://10.10.20.65/api/fdm/latest"  
    token = get_token(base_url)  
  
    headers = {  
        "Accept": "application/json",
```

```

        "Authorization": f"Bearer {token}",
    }

    #URL to read first object data
    url=f"{base_url}/object/networks"
    while True:
        #read next batch of messages with primitive error
        checking
        response = requests.get(
            url, headers=headers, params={"limit": page_size},
        verify=False
        )
        response.raise_for_status()

        body = response.json()

        #iterate through the received data and print content
        for net in body["items"]:
            print(f"Network object: '{net['name']}'"
                  f"type: {net['subType']}, value:
{net['value']}")

            # check content of the ["paging"] metadata (the "next"
            entry)
            # if list is present, use its 1st element as URL for the
            next iteration
            # if not, job is done ("next" entry will be empty in the
            last batch)
            if body["paging"]["next"]:
                url=body["paging"]["next"][0]
            else:
                break

if __name__ == "__main__":
    main()

```

## 2.5 Describe the steps in the OAuth2 three-legged authorization code grant flow

In the traditional client/server authentication model, the client authenticates with the resource owner's credentials (for example, username/password) when it requests a

restricted resource on a server. When a third-party application requires access to such resources, the resource owner has to share its credentials with the third party, which creates security problems and limitations.

OAuth addresses these issues by introducing an authorization layer and separating the roles of the client and the resource owner.

In OAuth, instead of using the resource owner's credentials, the client obtains an *access token* that has a specific scope, lifetime, and other access attributes. Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner.

OAuth version 2.0 (referred to as OAuth 2.0 or OAuth2, or simply OAuth, as v1.0 has long been obsolete) defines four roles:

- **Resource owner:** The end user
- **Client:** The application that requests access to the resource owner's data
- **Resource server:** Hosts the protected content that the client requests access to
- **Authorization server:** Verifies identity and issues tokens to the client

OAuth is designed for use with HTTP and specifies several *authorization grant* types to address different use cases and device capabilities. Grant types can be broadly split into *two-legged* and *three-legged* variants, referring to the number of parties involved in the authentication process. The two-legged process does not involve user interaction and is typically used for machine-to-machine authorization.

To be able to use OAuth, the client application needs to be registered with the authorization provider first. The provider

may request some common information about the application, such as its name, description, icon, and so on, as well as the application's redirect URL(s) for security purposes.

Once the application is registered, the provider will issue client credentials in the form of a *Client Identifier* (Client ID) and a *Client Secret*. The Client ID is used by the service API to identify the application (like a username), and the Client Secret is used to authenticate the identity of the application and must be kept private between the application and the API (like a traditional password).

At this point, the client application is ready to use the OAuth process with one of the following grant types:

- **Authorization Code grant:** Exchange an authorization code for an access token (three-legged).
- **Client Credentials grant:** Obtain an access token without a user (two-legged).

The client application requests an access token, which is granted based on the Client ID and Client Secret.

Here is a sample request:

[Click here to view code image](#)

```
POST /token HTTP/1.1
Host: authorization-server.com

grant_type=client_credentials&
client_id=iuhfvsdz78fs9dujtn35wigybv&
client_secret=859403985utjfk98u389oijrfu89i
```

Here is a sample response:

[Click here to view code image](#)

```
HTTP/1.1 200 OK
{
  "access_token": "2YotnFZFEjr1zCsicMWpAA",
```

```
    "token_type": "example",
    "expires_in": 3600,
    "example_parameter": "example_value"
}
```

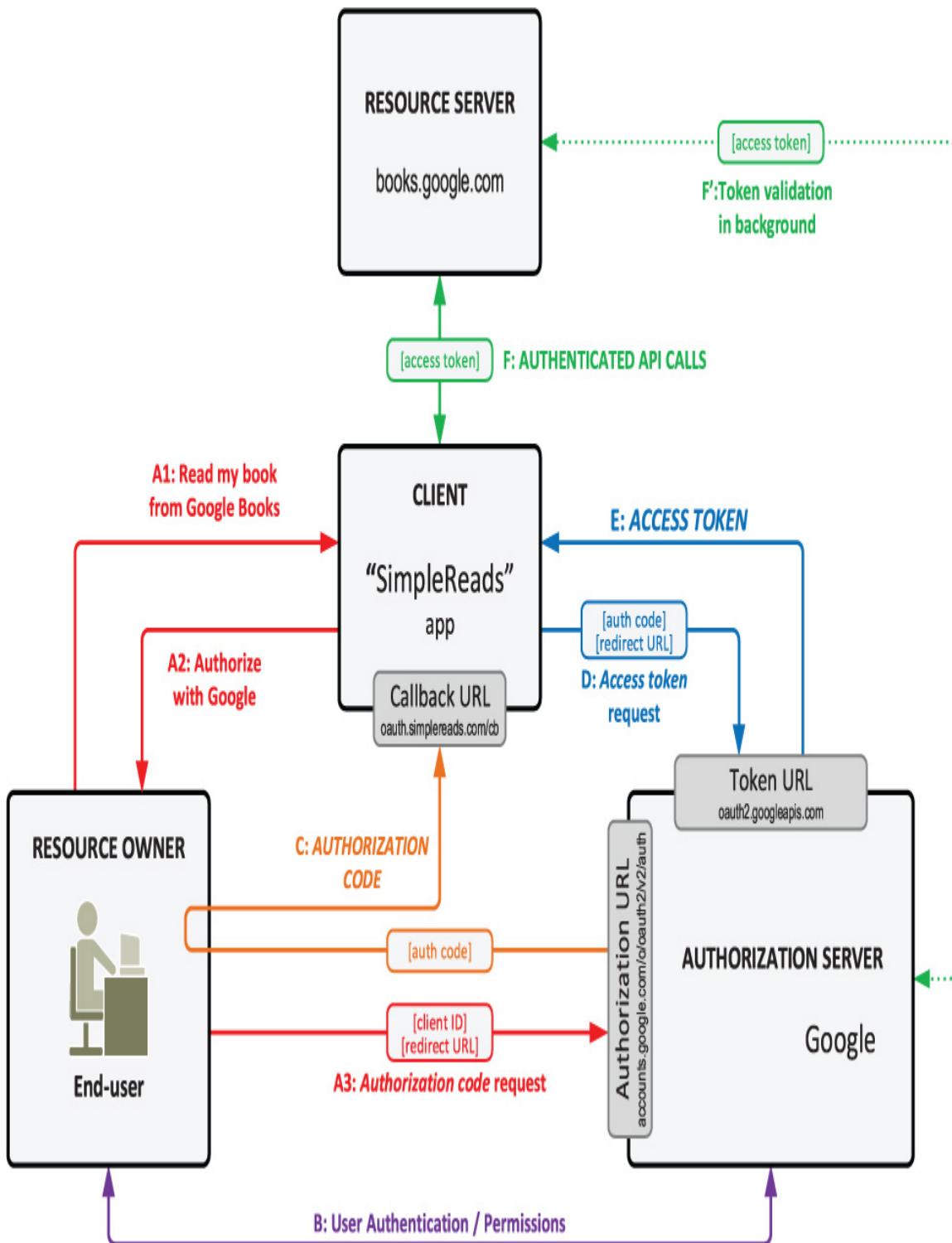
- **Refresh Token grant:** Refresh an access token without reauthentication.
- **Device Code grant:** Used on browser-less devices.
- **Password Credentials grant (legacy):** The user's credentials are exchanged for an access token.

## Authorization Code Grant Flow

To review the authorization code grant process, we'll use the fictitious “SimpleReads” book-reading application, which allows users to access personal content from a variety of online services. In the scenario shown in [Figure 28](#), a user has his books stored in Google Books but he does not want to share Google credentials with the app due to security concerns, so “SimpleReads” will use the authorization code grant flow to get access to the content.

This flow includes the following steps:

*Figure 28: OAuth2 Three-Legged Authorization*



- **(A)** The user wants to read one of his books, opens the app, and clicks the “Log in to Google” button. The **client** then initiates the OAuth flow by directing the

**resource owner**'s browser to the **authorization server**'s endpoint. The request includes the following elements:

- **Response type**: Sent to the server to indicate the desired grant type; must be “code.”
- **Scope**: Used to limit access only to resources the application needs (optional).
- **Client identifier**: Obtained during the client registration process.
- **State**: Any string value to maintain state between the authorization request and response. It will be included in the redirection URL and may be used to protect against attacks such as cross-site request forgery (CSRF; see [Section 4.10](#) for more details).
- **Redirection URI**: Where the user is redirected after completing the authorization. The value must match redirect URIs (uniform resource identifiers) established during the client registration process.

[Click here to view code image](#)

```
GET https://accounts.google.com/o/oauth2/v2/auth?  
response_type=code&  
scope=https%3A//www.googleapis.com/auth/books?  
client_id=123456789.apps.googleusercontent.com&  
state=random_custom_string&  
redirect_uri=https://oauth.simplereads.com/cb
```

- **(B)** In this step, the **resource owner** decides whether to grant the requested access to the **client** application. In the case of Google, it would display a consent window that shows the name of the application, Google services that it is requesting access to, and a summary of the scopes of access to be granted. The user can then consent to grant access or refuse the request.

- **(C)** If the **resource owner** grants access, the **authorization server** redirects the user's browser back to the **client** using the redirection URI provided earlier (in the request or during client registration). The redirection URI includes an authorization code and any local state provided by the client earlier (for request validation):

[Click here to view code image](#)

```
https://oauth.simplereads.com/cb?
code=Spxl0BeZQQYbYS6WxSbIA&
state=random_custom_string&scope=https%3A//www.googleapis.co
m/auth/books
```

If the user does not approve the request, the response contains an error message:

[Click here to view code image](#)

```
https://oauth.simplereads.com/cb?error=access_denied&
state=random_custom_string
```

- **(D)** The **client** exchanges the authorization code for the access token by making a POST request to the **authorization server**'s token endpoint with the following parameters:

- o **Grant type:** Must be set to "authorization\_code."
- o **Client authentication:** Client ID and Client Secret either via HTTP Basic Auth or as POST body parameters.
- o **Code:** Authorization code received in the previous step.
- o **Redirect URL:** If it was included in the initial authorization request, it must be included in the token request as well and must be identical.

[Click here to view code image](#)

```
POST /token HTTP/1.1
Host: oauth2.googleapis.com

grant_type=authorization_code&
client_id=123456789.apps.googleusercontent.com&
client_secret=your_client_secret&
code=Splxl0BeZQQYbYS6WxSbIA&
redirect_uri=https://oauth.simplereads.com/cb
```

- **(E)** The **authorization server** authenticates the **client**, validates the authorization code, and checks that the redirection URI matches the one used in step (A). If the request is valid, the authorization server responds with an access token and, optionally, a refresh token:

[Click here to view code image](#)

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
  "access_token": "1/fFAGDNJrz1FTz75BzhT3Zn",
  "token_type": "Bearer",
  "expires_in": 3600,
  "refresh_token": "tGzv3J0kF0XG5Qx2TlKWIA",
}
```

- **(F)** The **client** accesses protected resources, using the access token for authentication with the **resource server**. The resource server validates the access token and ensures that it has not expired and that its scope covers the requested resource

You may wonder why the two-step process is required—why not provide the access token instead of the authorization code right away (step C)? It is done this way to address security concerns related to the fact that the user's machine and/or browser may be compromised and content may be leaked or stolen by some malicious code. The two-step OAuth process ensures the following:

- **Access token privacy:** The authorization code is exchanged for an access token by the client application, server to server, with no user browser involved. As a result, when the access token is issued, it's known to the client only and not exposed to the user/browser.
- **Client Secret privacy:** Used for the authentication only during the server-to-server code exchange (step D), so it does not need to be exposed to the user. As a result, even if attackers somehow steal the authorization code, they cannot exchange it for the access token and get access to the resource server.

## 2.6 Chapter 2 Review Questions

Q1: Which statement is true about REST API timeout errors?

- A) Timeout handling is only needed for external connections with high latency.
- B) Timeout indicates that the server is down, so it's best to exclude it from subsequent calls.
- C) The backoff timer method uses decreasing (backing off) timeouts between calls.
- D) Setting the timeout value both too low and too high has a negative impact.

Q2: Which statement is true about REST API rate-limiting server response codes?

- A) The server responds with the HTTP 229 status code to indicate that the request was executed but the next call should be paused.
- B) The server responds with the HTTP 329 status code to indicate a redirect to a new URL that needs to be called to confirm it's not a denial of service.

- C) The server responds with the HTTP 429 status code to indicate to a client that it's making too many requests.
- D) The server responds with the HTTP 529 status code to indicate the server failed to execute a request and it should be retried.

Q3: Which statement is true about timeout and rate-limiting error handling with REST API?

- A) Any API call may be safely repeated after a small pause following a timeout or rate-limit error.
- B) Rate-limiting response indicates how long to wait before making the next call. If no such indication is included, a retry can be made immediately.
- C) Using rate-limiting requests from within the application is a proper solution to avoid errors when API restrictions are known and documented.
- D) Requests can be retried as many times as needed until a successful response is received.

Q4: Which statement is true about REST API unrecoverable errors?

- A) Although not desired, unrecoverable errors are expected and should be properly treated.
- B) Upon receiving an unrecoverable REST API error, program execution should be immediately stopped.
- C) The HTTP 403 "Forbidden" error code indicates the authentication token has expired and the user needs to re-authenticate.
- D) Both recoverable and unrecoverable errors may be handled without user interaction.

Q5: Which statement is true about HTTP cache controls in API calls?

- A) A stored fresh response can be reused only after validation at the origin server.
- B) If a response is not fresh, it's unusable and should be removed from the cache.
- C) ETag is one of the headers used for cache content freshness validation.
- D) A freshness timer is a faster and more reliable mechanism than content validation.

**Q6: Which statement is true about HTTP Cache-Control headers?**

- A) Both strong and weak ETags may be used in the “If-None-Match” header in HTTP GET operations.
- B) Both strong and weak ETags may be used in the “If-Match” header in HTTP PUT operations.
- C) The “If-Unmodified-Since” header is used in HTTP GET operations
- D) The “no-cache” value of the Cache-Control HTTP header disables caching of the returned response.

**Q7: Which statement is true about pagination support by REST APIs?**

- A) Cursor-based pagination is a simple and robust way of doing pagination.
- B) Offset/page-based pagination is a simple and robust way of doing pagination.
- C) Pagination improves the user experience at the cost of increased resource utilization.
- D) Pagination links are typically constructed by the application that uses the REST API.

**Q8: Which is *not* a valid actor in the OAuth2 authorization?**

- A) Resource owner

- B) Resource server
- C) Authorization application
- D) Authorization server

Q9: Which parameter is fixed and always the same in the first (Authorization) client request of the three-legged OAuth2 flow?

- A) response\_type
- B) client\_id
- C) state
- D) redirect\_uri

Q10: Which statement is true about the second (Access Token) client request of the three-legged OAuth2 flow?

- A) The “grant\_type” parameter is set to “client\_credentials.”
- B) The client requests an access token by providing the Authorization code received in the previous step.
- C) “redirect\_uri” will be used to supply the newly obtained access token.
- D) The response to this request includes an access token that is valid until the user logs out.

---

## 3. Cisco Platforms

---

### 3.1 Construct API requests to implement ChatOps with Webex API

Note: Webex Teams was renamed to Webex in December of 2020. At the time of writing, the exam blueprint has not been updated to reflect it, but this book uses the new naming.

Chat applications are widely used these days for digital communication within companies. Creating and opening APIs for these applications opens new ways of using them, transforming them into conversation, collaboration, and operations hubs.

Chats are no longer just conversation spaces; they turn into infrastructure management workplaces with the following benefits:

- Centralized infrastructure management
- A conversational approach to operations
- Tighter collaboration
- Transparent workflow

In a *ChatOps* environment, operations and engineering teams use real-time chats to communicate with their applications and infrastructure with the help of *chatbots*, programs that simulate a conversation. Chatbots can join group chats, access messages, and react to them according

to the programmed logic: they can simply respond to questions or execute automated workflows.

Unlike regular users who register with usernames and passwords, chatbots are typically assigned specially generated API access tokens with proper access permissions.

Most popular chat services offer their own API (Cisco Webex, Microsoft Teams, Google Hangouts, Slack, and so on). Chatbot frameworks and online services allow easy integration if you have a mix of messaging platforms.

Chatbots can be used as follows:

- **Notifiers:** Monitor for specific events and send notifications to the chat room when a change happens (for example, a new git commit, VM deployment completion, or service becomes unavailable).
- **Controllers:** Listen for specific phrases in a chat room and execute predefined tasks based on that text (for example, create a VM or disable wireless SSID).
- **Personal assistants:** Use natural language processing to respond to human language as a normal person would.

When ChatOps is implemented with Webex, the chatbot needs to listen at a webhook URL for Webex notifications, execute logic (for example, make API calls based on received commands), and send responses back to Webex. Practically speaking, it is a small web server that is serving HTTP requests and has to be publicly available. From the implementation perspective, the following components are usually required:

- Webex access token to interact with Webex API (chatbot property)

- Webhook listener (easy to implement with the Python “flask” module)
- HTTP library to make own requests (typically Python “requests” module)
- Authentication credentials for the infrastructure API you planning to use (for example, Meraki API key)

To use Cisco Webex chatbot, first, you need to create it:

- Log in to the personal Webex account for developers at <https://developer.webex.com/>.
- Click the Bots icon or access it directly at <https://developer.webex.com/docs/bots>.
- Click “Create a Bot” and provide the following information:
  - The bot name as it will appear in Webex.
  - The bot username (<name>@webex.bot). Users will use it to add this bot to their space. It cannot be changed later.
  - An icon (default or custom 512x512 pixels in JPG or PNG format)
  - A description (what this bot does and how to use it)
- When the bot is created, you will be provided with the following items:
  - A bot ID. This is a unique system-generated ID.
  - An access token for the new bot. The bot’s access token will only be displayed once. Make sure to copy the token and keep it somewhere safe. If you misplace it, you can regenerate a new access token.

An access token is required for authentication with Webex API to be able to send and receive messages.

Here is a usage example:

[Click here to view code image](#)

```
bearer_token = 'ZTZhZmJi...d65305c2ce82'  
base_url = 'https://webexapis.com/v1/'  
headers = {  
    "Authorization": f"Bearer {bearer_token}",  
    "Content-Type": "application/json"  
}  
result = requests.get(f'{base_url}<function>', headers=headers)
```

To start a direct conversation or to add a chatbot to the existing chat room (aka “space”), use the bot’s username (like any other user). This would allow your bot to send messages to you or your space.

To send messages to the bot, you need to register a *webhook*. Webhooks are provided in the form of an external public URL that is called after a specific event occurs inside Webex. Ngrok is an example of software that allows for the creation of publicly reachable URLs for a private location.

Webhook calls only notify that some event has occurred (for example, that a new message was posted) but do not provide message content. Normally your event handler will make a call to the Webex REST API and get full details.

The webhook has to be registered with the API call as documented at

<https://developer.webex.com/docs/api/v1/webhooks/create-a-webhook>. The following sample code shows examples of how to create, list, and delete webhooks:

[Click here to view code image](#)

```
import requests  
  
base_url = 'https://webexapis.com/v1'  
bearer_token = 'ZTZhZmJi..d65305c2ce82'
```

```

headers = {
    "Accept": "application/json",
    "Content-Type": "application/json",
    "Authorization": f"Bearer {bearer_token}",
}

webhook = {
    "name": "My Awesome Webhook",
    "targetUrl": "http://377cf7274b2d.ngrok.io/webhook",
    "resource": "messages",
    "event": "created",
}

#read current webhooks and delete them as a cleanup
response = requests.get(f"{base_url}/webhooks", headers=headers,
json=webhook)
response.raise_for_status()

#delete them one by one
for item in response.json()["items"]:
    print (f'Deleting webhook \'{item["name"]}\'...')
    del = requests.delete(f'{base_url}/webhooks/{item["id"]}',
headers=headers)
    del.raise_for_status()
    print (del.status_code)

#create a new one
response = requests.post(f"{base_url}/webhooks",
headers=headers, json=webhook)
response.raise_for_status()

webhook_id = response.json()["id"]
print(f"Webhook for {webhook['targetUrl']} added with
ID\n{webhook_id}")

```

Webhook notification data will be sent via HTTP POST in JavaScript Object Notation (JSON) format to a provided URL each time it is triggered. This URL must be publicly reachable by Webex, with your application listening for inbound HTTP requests. A response is also required to confirm notification delivery. If Webex does not receive a successful HTTP response in the 2xx range from the server, your webhook will be disabled after 100 failed attempts within a 5-minute period.

Here is an example of barebone chatbot application code that echoes the message content back to the chat room:

[Click here to view code image](#)

```
from flask import Flask, request, json
import requests

app = Flask(__name__)
port = 5010

bearer_token = 'ZTZhZmJi..d65305c2ce82'
base_url = 'https://webexapis.com/v1/'

class Messenger():
    def __init__(self, base_url=base_url,
bearer_token=bearer_token):
        self.base_url = base_url
        self.bearer_token = bearer_token
        self.headers = {
            "Accept": "application/json",
            "Content-Type": "application/json",
            "Authorization": f"Bearer {bearer_token}",
        }

    def get_message(self, message_id):
        received_message_url =
f'{self.base_url}/messages/{message_id}'
        message_text = requests.get(received_message_url,
headers=self.headers).json().get('text')
        return (message_text)

    def post_message(self, room_id, message):
        data = {
            "roomId": room_id,
            "text": message,
        }
        post_message_url = f'{self.base_url}/messages'
        post_message = requests.post(post_message_url,
headers=self.headers,
                               data=json.dumps(data))

msg = Messenger()

@app.route('/webhook', methods=['POST'])
def webhook():
```

```

        if 'application/json' in request.headers.get('Content-Type'):
            data = request.get_json()

            room_id = data.get('data').get('roomId')
            message_id = data.get('data').get('id')
            message = msg.get_message(message_id)

            reply = f'Bot received message "{message}"'
            msg.post_message(room_id, reply)

            return data
        else:
            return ('Wrong data format', 400)

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=port, debug=True)

```

Note: In group rooms, bots only have access to messages in which they are mentioned. This means that webhooks would only be called when the bot is specifically mentioned in a room (with @).

## 3.2 Construct API requests to create and delete objects using Firepower Device Management (FDM)

The following is the general process for interacting with the API:

- Obtain an access token to authenticate your API calls.
- Build a JSON payload if needed (except when you read data).
- Send a request to the FDM device using one of the HTTP methods and the specific resource URL.
- Consume the returned JSON response.

- If you make configuration changes, deploy the changes.

You can use built-in API Explorer to research API functions at the <https://fdm-host/#/api-explorer> URL, where “fdm-host” is IP address/name of the FDM device.

Before making API calls, you need to obtain an authentication token and include it in the “Authorization” header in all the subsequent calls.

Here's an example of obtaining the token:

[Click here to view code image](#)

```
import requests

base_url = "https://10.10.20.65/api/fdm/latest"
auth_url = f"{base_url}/fdm/token"

def get_token():

    headers = {
        "Content-Type": "application/json",
        "Accept": "application/json",
    }

    data = {
        "grant_type": "password",
        "username": "admin",
        "password": "Cisco1234",
    }

    response = requests.post(auth_url, headers=headers,
json=data, verify=False)
    response.raise_for_status()

    access_token = response.json()["access_token"]

    return access_token
```

And here's the response data dump for reference:

[Click here to view code image](#)

```
{
    "access_token": "eyJhbGciOiJIUzI1NiJ9eyJpYXQiOjE2MDE1...553sTnbJUm329A",
        "expires_in": 1800,
        "token_type": "Bearer",
        "refresh_token": "ehbGciOiJIUzI1NiJSI6kIafAZxUopvfm0I...CyfwqwYbNEgIlg",
        "refresh_expires_in": 2400
}
```

The following code provides an example of how to work with the *network* objects via FDM API. It includes authentication, network object read/create/delete, and configuration deployment API calls.

[Click here to view code image](#)

```
import requests

new_net_object = {
    "name": "GOOGLE-DNS",
    "description": "Google public DNS server",
    "subType": "HOST",
    "value": "8.8.8.8",
    "type": "networkobject"
}

class FDM_API:

    def __init__(self, host, username, password):
        """ Initialize FDM/FTD API object """

        self.base_url = f"https://{{host}}/api/fdm/latest"
        self.username = username
        self.password = password
        self.headers = {
            "Content-Type": "application/json",
            "Accept": "application/json",
        }
        self.get_token()

    def do_api_call(self, action, url, fdm_object = None):
        """
        Wrapper for API calls to avoid repeating code.
        Parameters:
        - action: HTTP verb: GET/POST/DELETE etc.
        """
        # Implementation of do_api_call function
        pass
```

```

        - url: API-specific part of URL (prefixed with
$base_url)
        - fdm_object: data for the POST calls
    Return: dict with the response results
    """

    api_call = requests.request(
        action,
        f"{self.base_url}/{url}",
        headers = self.headers,
        json = fdm_object,
        verify = False,
        timeout = 3,
    )
    api_call.raise_for_status()

    # debug output
    print (f" >> {action} to {api_call.url} / HTTP
{api_call.status_code}")

    # HTTP 204 responses return empty output, .json() would
fail with that
    if api_call.text:
        return api_call.json()

def get_token(self):
    """ Obtain authentication token and use it in future
calls """
    # full URL: "https://host/api/fdm/latest/fdm/token"
    api_path = "fdm/token"
    data = {
        "grant_type": "password",
        "username": self.username,
        "password": self.password,
    }

    response = self.do_api_call ("POST", api_path, data)
    access_token = response.get("access_token")
    self.headers['Authorization'] = f'Bearer {access_token}'

def deploy (self):
    """
    Call "https://host/api/fdm/latest/operational/deploy"
    to deploy prepared configuration changes
    """

    response = self.do_api_call("POST",

```

```

"operational/deploy")
    return True

def find_object_by_name (obj_name, obj_list):
    """
        Helper function
        Find if the object with a given "obj_name" exists in the
    "obj_list" list
        Return: ID of the existing object
    """
    for obj in obj_list:
        if obj['name'] == obj_name:
            return obj['id']
    return None

def main():
    # Disable certificate warnings
    requests.packages.urllib3.disable_warnings()

    # Settings for the "Firepower Threat Defense REST API"
    DevNet sandbox
    fdm=FDM_API("10.10.20.65", "admin", "Cisco1234")

        # read and print current network objects
        fdm_net_objs = fdm.do_api_call("GET", "object/networks")
        for net_obj in fdm_net_objs['items']:
            print(f"Existing {net_obj['name']} object, ID:
{net_obj['id']}")

        # Check if object with the same name already exists
        # if so, delete it first to avoid errors
        obj_id = find_object_by_name (new_net_object,
fdm_net_objs['items'])
        if obj_id:
            print(f"{new_net_object['name']} object already exists,
deleting...")
            fdm.do_api_call ("DELETE", f"object/networks/{obj_id}")

        # Create a new network object
        new_object = fdm.do_api_call ("POST", "object/networks",
new_net_object)
        print(f"Created {new_object['name']} object, ID:
{new_object['id']}")

        # Configuration changes need to be deployed to be activated
        # Note it will fail in the DevNet sandbox when it's Read-
Only

```

```
fdm.deploy()  
  
if __name__ == "__main__":  
    main()
```

Here is some sample output:

[Click here to view code image](#)

```
>>> POST to https://10.10.20.65/api/fdm/latest/fdm/token / HTTP  
200  
    >>> GET to https://10.10.20.65/api/fdm/latest/object/networks /  
HTTP 200  
Existing GOOGLE-DNS object, ID:41eb558b-447b-11eb-b176-  
39fc4b8b65b3  
Existing any-ipv4 object, ID:12e56f78-eea2-11ea-baa0-  
0f39a2dff5dc  
Existing any-ipv6 object, ID:12ffd549-eea2-11ea-baa0-  
8dfd05c8eadf  
GOOGLE-DNS object already exists, deleting...  
    >>> DELETE to  
https://10.10.20.65/api/fdm/latest/object/networks/41eb558b-  
447b-11eb-b176-39fc4b8b65b3 / HTTP 204  
    >>> POST to https://10.10.20.65/api/fdm/latest/object/networks  
/ HTTP 200  
Created GOOGLE-DNS object, ID:a9b130f2-447b-11eb-b176-  
5d762101a342  
    >>> POST to  
https://10.10.20.65/api/fdm/latest/operational/deploy / HTTP 200
```

The preceding example shows how to work with the “network” type object. For the other types, the URL and objects need to be updated accordingly. For example, to work with the “URL” type objects, the “url” parameter for API calls changes from “object/networks” to “object/urls” and the “new\_object” data will be the following:

[Click here to view code image](#)

```
new_url_object = {  
    "name": "Block_badurl.com",  
    "description": "Sample URL object",  
    "url": "badurl.com",  
    "type": "urlobject",  
}
```

### 3.3 Construct API requests using the Meraki platform to accomplish these tasks

The Meraki Dashboard API is an interface for software to interact directly with the Meraki cloud platform and Meraki managed devices. The Dashboard API is a modern RESTful API that uses HTTPS and JSON as a human-readable format. It contains a set of tools for use cases such as provisioning, bulk configuration changes, monitoring, and role-based access controls, and it can be used for purposes such as the following:

- Adding new organizations, admins, networks, devices, VLANs, and SSIDs
- Provisioning thousands of new sites in minutes with an automation script
- Automatically onboarding and offboarding new employees' teleworker devices
- Building your dashboard for store managers, field techs, or unique use cases

Overall, the API mirrors the structure of the Meraki Dashboard: a single *account* may own several *organizations*, organizations consist of *networks*, which contain their own *devices*, *SSIDs*, and other specific settings, as shown in [Figure 29](#).

*Figure 29: Meraki API Structure*



To interact with the Dashboard API, you must first obtain an API key, as it's used to provide authorization for each request:

- Open your Meraki dashboard (<https://dashboard.meraki.com>).
- Once logged in, navigate to the Organization | Settings menu.
- Ensure that API Access is set to “ Enable access to the Cisco Meraki Dashboard API.”
- Click your username in the top-right corner and go to “My profile” to generate the API key. The Dashboard does not store API keys in plaintext for security reasons, so copy and store your API key in a safe place, as this is the only time you will be able to record it.

If you lose your API key, you will have to revoke it and generate a new one (be aware that only two API keys can be valid at the same time). Note that if you don't have access to your Meraki dashboard, you can use the “Meraki Always-On” DevNet sandbox at <https://devnetsandbox.cisco.com/RM/Topology>.

There are two versions of the Dashboard API at the moment: the classic v0 and the newer v1. The following examples use v0 with the following base URL to the Meraki cloud:

<https://api.meraki.com/api/v0>

Note: API requests are limited to five calls per second (per organization).

### 3.3.A Use Meraki Dashboard APIs to enable an SSID

The following is the typical sequence to manage a network setting (for example, SSID):

1. List the available organization(s) and find the target organizationId.
2. List the available networks for the organizationId and find the target networkId.
3. List the available SSID for the networkId and find the target SSID number.
4. Get detailed info or update the SSID configuration by the SSID number.

The following code provides an example of SSID manipulation. The script will flip the state of the SSID defined by the organization, network, and SSID names in a text form. It sequentially finds organizationId, networkId, and SSID number and then reads the current SSID state and flips it to the opposite.

[Click here to view code image](#)

```
import requests
import json

MY_ORG_NAME = "network"
MY_NET_NAME = "wireless"
MY_SSID_NAME = "devnet"

unknown_ssid = 255

base_url = "https://api.meraki.com/api/v0"
```

```

headers = {
    "Accept": "application/json",
    "Content-Type": "application/json",
    "X-Cisco-Meraki-API-Key": "f9...xxx...8a",
}

def do_API_call(api_url, action = "GET", json = None):
    """
        Basic wrapper for API calls. Parameters:
        - resource: API-specific part of URL (prefixed with
$base_url)
        - action: HTTP verb: GET/POST/DELETE etc.
        - json: data for PUT/POST calls
        Return: API response results
    """

    api_call = requests.request(
        method = action,
        url = f"{base_url}/{api_url}",
        headers = headers,
        json = json,
        timeout = 3,
    )

    # debug output
    print (f" >>> {action} to {api_call.url} / HTTP
{api_call.status_code}")

    api_call.raise_for_status()
    if api_call.text:
        return api_call.json()

def find_networkId(org_name, net_name):
    """
        Parameters:
        - org_name: text organization name, may be partial as soon
as unique
        - net_name: text network name, may be partial as soon as
unique
        Return: ID of the network for API calls, or None if not
found
    """

    # get a list of all organizations under account
    orgs = do_API_call("organizations")

    #loop through them to find org_name and its ID
    organizationId = 0

```

```

for org in orgs:
    if org_name.lower() in org["name"].lower():
        print (f"Found a match for the {org['name']} org")
        organizationId = org["id"]
        break

    # if found, get a list of the networks and
    if organizationId:
        nets =
do_API_call(f"organizations/{organizationId}/networks")

    # loop through them to find net_name and its ID
    for net in nets:
        print (f"Found a match for the {net['name']} network")
        if net_name in net["name"].lower():
            return net["id"]

    # by default (not found) return None
    return None

def find_ssid_nr(networkId, ssid_name):
    """
    Return target SSID's ID in the network or "unknown_ssid" if
    not found
    """

    # get a list of all SSIDs for the network
    ssids = do_API_call(f"networks/{networkId}/ssids")

    # loop through them to find SSID and its ID
    for ssid in ssids:
        if ssid_name.lower() in ssid["name"].lower():
            print (
                f"Found a match for the {ssid['name']} network,"
                f"SSID# {ssid['number']}"
            )
            return ssid["number"]

    # Use SSID value of 255 to indicate "not found"
    return unknown_ssid

def state_name(state):
    """ Return printable SSID state name """
    return "enabled" if state else "disabled"

def main():

```

```

# obtain networkId from the network name
my_network = find_networkId(MY_ORG_NAME, MY_NET_NAME)
if not my_network:
    print (f"Network {MY_NET_NAME} not found in the
{MY_ORG_NAME} org")
    exit ()

# obtain SSID # from the SSID name
my_ssid_nr = find_ssid_nr (my_network, MY_SSID_NAME)
if my_ssid_nr == unknown_ssid:
    print (f"SSID {MY_SSID_NAME} not found in the network
{MY_NET_NAME}")
    exit ()

#Read the current state
ssid_state =
do_API_call(f"networks/{my_network}/ssids/{my_ssid_nr}")
print (f'Current state is
{state_name(ssid_state["enabled"])}')

#print SSID data structure
print (json.dumps(ssid_state, indent=4))

# toggle SSID's "enabled" value
new_ssid_state = not ssid_state["enabled"]

#Update SSID state
update_req = do_API_call(
    api_url = f"networks/{my_network}/ssids/{my_ssid_nr}",
    action = "PUT",
    json = {"enabled": new_ssid_state},
)
# print a returned new state to confirm
print (f'New state is {state_name(update_req["enabled"])}')

if __name__ == "__main__":
    main()

```

Here is some sample output:

[Click here to view code image](#)

```

>>> GET to https://api.meraki.com/api/v0/organizations / HTTP
200
Found a match for the Home Network org

```

```

>>> GET to
https://api.meraki.com/api/v0/organizations/888888/networks /
HTTP 200
Found a match for the Home Network - Wireless network
>>> GET to
https://api.meraki.com/api/v0/networks/L_634444590000000000/ssids
/ HTTP 200
Found a match for the DEVNET-TEST network, SSID# 9
>>> GET to
https://api.meraki.com/api/v0/networks/L_634444590000000000/ssids
/9 / HTTP 200
Current state is enabled
{
    "number": 9,
    "name": "DEVNET-TEST",
    "enabled": true,
    "splashPage": "None",
    "ssidAdminAccessible": false,
    "authMode": "psk",
    "psk": "WirelessPassword",
    "encryptionMode": "wpa",
    "wpaEncryptionMode": "WPA2 only",
    "ipAssignmentMode": "NAT mode",
    "minBitrate": 11,
    "bandSelection": "Dual band operation",
    "perClientBandwidthLimitUp": 0,
    "perClientBandwidthLimitDown": 0,
    "visible": true,
    "availableOnAllAps": true,
    "availabilityTags": []
}
>>> PUT to
https://api.meraki.com/api/v0/networks/L_634444590000000000/ssids
/9 / HTTP 200
New state is disabled

```

### 3.3.B Use Meraki location APIs to retrieve location data

Using the physical placement of each wireless access point (AP), the Meraki cloud aggregates raw client location data and provides a real-time estimate on the location of Wi-Fi (associated and non-associated) and Bluetooth Low Energy (BLE) devices in real time. The Scanning API delivers this

data in real time from the Meraki cloud to location application, data warehouse, or business intelligence systems.

Meraki developed location services with privacy in mind. Because the location data contains raw MAC addresses, Meraki implemented several security mechanisms to anonymize the data in an irreversible fashion. The Meraki cloud stores only hashed, salted, and truncated versions of the MAC addresses so that they are not identifiable.

The elements are exported via an HTTP POST of JSON data to a specified destination server. The JSON posts occur frequently, typically batched every minute for each AP. The location data accuracy can vary based on a number of factors and should be considered a best-effort estimate. AP placement, environmental conditions, and client device orientation can influence X/Y coordinates estimation; experimentation can help improve the accuracy of results or determine a maximum acceptable uncertainty for data points.

The Location API is configured in the Meraki Dashboard and will capture all Wi-Fi observations by default. Follow these steps to enable it:

- Open your Meraki dashboard (<https://dashboard.meraki.com>).
- Once logged in, navigate to the Network Wide | General menu.
- In the “Location and scanning” section, set the Scanning API dropdown box to Scanning API enabled.
- Record the Validator string.
- Specify a post URL, authentication secret, location API version, and radio type.

Location and scanning

Analytics: Analytics enabled

Scanning API: Scanning API enabled

Validator: 191aa6b59e2a3a6c108e4eb9bf47e1c4fde24e9

Post URLs	Status	Post URL	Secret	API Version	Radio Type
locationscan.example.com	.....	Show secret	V2	WiFi	Validate X
locationscan-bt.example.com	.....	Show secret	V2	Bluetooth	Validate X

Add a Post URL

- Configure and host your HTTP server to receive JSON objects.

Validator is used by the Meraki cloud to validate your application: on the first connection, it will check that the server returns the organization-specific validator string as a response, which will verify the organization's identity. The Meraki cloud will then begin performing JSON posts.

The secret is used by your HTTP server to validate that the JSON posts are coming from the Meraki cloud. It is included in every POST request.

Here is a location data example:

[Click here to view code image](#)

```
{
  {
    "version": "2.1",
    "secret": "supersecret",
    "type": "DevicesSeen",
    "data": {
      "apMac": "00:18:0a:13:dd:b0",
      "apFloors": [],
      "apTags": [
        "dev",
        "entrance",
        "office"
      ],
    }
  }
}
```

```

"observations": [
    {
        "ipv4": "/192.168.0.38",
        "location": {
            "lat": 52.5355157,
            "lng": -0.06990350000000944,
            "unc": 1.233417960754815,
            "x": [],
            "y": []
        },
        "seenTime": "2016-09-24T00:06:23Z",
        "ssid": ".interwebs",
        "os": null,
        "clientMac": "18:fe:34:fc:ff:ff",
        "name": "Mission Control",
        "seenEpoch": 1474675583,
        "rssi": 47,
        "ipv6": null,
        "manufacturer": "Espressif"
    },
    {
        "ipv4": "/192.168.0.15",
        "location": {
            "lat": 52.5355157,
            "lng": -0.06990350000000944,
            "unc": 1.5497743004111961,
            "x": [],
            "y": []
        },
        "seenTime": "2016-09-24T00:06:40Z",
        "ssid": ".interwebs",
        "os": "Generic Linux",
        "clientMac": "74:da:38:56:ff:ff",
        "name": "Sat-1",
        "seenEpoch": 1474675600,
        "rssi": 47,
        "ipv6": null,
        "manufacturer": "Edimax Technology"
    }
]
}

```

### 3.4 Construct API calls to retrieve data from Intersight

Cisco Intersight provides a cloud-based RESTful API to manage Cisco Unified Computing System (UCS) and Cisco HyperFlex systems across multiple data centers.

To retrieve data, Intersight provides a rich query language based on the OData standard. The query parameters are included in the URL of the HTTP GET request in a form of expression supplied after the question mark (“?”) character. Query expressions refer to named properties and may be used to filter, select, count, sort, aggregate, expand, or search them.

The `$filter` query option allows clients to retrieve a subset of resources based on some expression. It supports multiple operators to filter managed objects: `eq`, `ne`, `gt`, `ge`, `lt`, `le`, `and`, `or`, `not`, and `in` operators, as well as a set of string functions (`contains`, `endswith`, and so on). Here are a few examples:

[Click here to view code image](#)

```
GET /api/v1/compute/RackUnits?$filter=Name eq 'WZP211704KM'  
GET /api/v1/compute/RackUnits?$filter=Model ne 'UCSC-C240-M5SN'  
GET /api/v1/RackUnits?$filter=NumCpuCores ge 6 and  
AvailableMemory gt 65000  
GET /api/v1/RackUnits?$filter=not(Model eq 'HX220C-M5SX' or  
Model eq 'HX220C-M5S')  
GET /api/v1/RackUnits?$filter=Model in ('HX220C-M5SX', 'UCSC-  
C240-M5SN')  
GET /api/v1/RackUnits?$filter=contains(Model, 'C240')  
GET /api/v1/RackUnits?$filter=endswith(Model, 'M5')  
GET /api/v1/aaa/AuditRecords?$filter=CreateTime gt 2020-06-  
20T08:00:0.000Z
```

The Lambda `any` operator allows finer object filtering by executing a custom function against items in a list. It applies a Boolean expression to each member of a collection and includes only those resources where that expression is true.

In the following example, compute RackUnits resources might have multiple `tags` assigned (each tag is just a text

“key”/“value” pair), and we need to find all compute resources that have a tag called “Site”:

[Click here to view code image](#)

```
GET /api/v1/compute/RackUnits?$filter=Tags/any(t:t/Key eq 'Site')
```

- Iterate through the `Tags` elements (`filter=Tags/any`) for each compute resource.
- At each iteration, `Tags` entries are set to variable `t` within the function `((t:))`.
- Check that the tag's “key” has a value of “Site” (`t/Key eq 'Site'`).
- If true, the Lambda function returns true and the element passes the filtering condition.

Like any other filter option, the Lambda `any` operator may be repeated and combined with other options. For example, the following query selects resources where the “Site” tag is present and set to “London” AND the “Environment” tag is present and set to “Production”:

[Click here to view code image](#)

```
GET /api/v1/compute/RackUnits?$filter=Tags/any(t:t/Key eq 'Site' and t/Value eq 'London') and Tags/any(t:t/Key eq 'Environment' and t/Value eq 'Production')
```

Date and time functions such as `now()` may be used to apply filtering to *time* fields. For example, to query the rack unit servers that were created fewer than 30 days ago, the following query filter may be used:

[Click here to view code image](#)

```
GET /api/v1/compute/RackUnits?$filter=CreateTime gt now() sub P30D
```

The `$select` query option allows clients to request a specific set of properties for each entity. The value of the `$select` option is a comma-separated list of property names. Here's an example:

[Click here to view code image](#)

```
GET /api/v1/compute/RackUnits?$select=Vendor,Model,Serial
```

The `$top` and `$skip` query options are used to paginate the results.

The `$orderby` query option allows clients to request resources in a particular order.

Options can be combined in a query, as in the following example:

[Click here to view code image](#)

```
GET /api/v1/compute/RackUnits?  
$select=Model,Serial&filter=endswith(Model,'M5')  
&orderby=Model&top=10
```

## 3.5 Construct a Python script using the UCS APIs to provision a new UCS server given a template

The Cisco UCS Manager XML API is a programmatic interface that allows management of Cisco UCS compute, network, and storage resources. The API accepts XML documents through HTTP(S) (no JSON).

Internally, configuration and state information for Cisco UCS is stored in a hierarchical tree structure known as the *management information tree*, and it is completely accessible through the XML API. UCS API supports operations on a single object or an object hierarchy in this tree.

The operation of the API is transactional. When multiple managed objects are being configured, the API operation stops if any of them (a virtual NIC, for example) cannot be configured. In that case, the management information tree is rolled back to the state before the operation and an error is returned.

The API operates in a “forgiving” mode. Missing attributes are replaced with applicable default values, and incorrect attributes are ignored.

## Cisco UCS Management Information Model

All the physical and logical components that comprise Cisco UCS are represented in a hierarchical management information model (MIM), also referred to as the MIT. Each node in the tree represents a *managed object* or group of objects that contains its administrative state and its operational state. Managed objects are abstractions of the Cisco UCS resources, such as fabric interconnects, chassis, blades, rack-mounted servers, and so on.

The hierarchical structure starts at the top (sys) and contains parent and child nodes. Each node in this tree is a managed object, and each object in Cisco UCS has a unique distinguished name (DN) that describes the object and its place in the tree. Here’s an example:

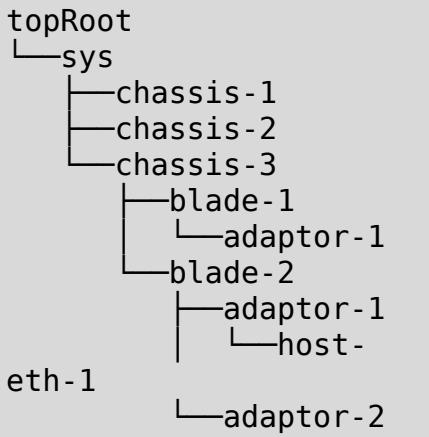
[Click here to view code image](#)

```
dn = "sys/chassis-5/blade-2/adaptor-1/host-eth-2"
```

The information model is centrally stored and managed by the *data management engine (DME)*, a user-level process running on the fabric interconnects. When a user initiates an administrative change to a Cisco UCS component (for example, applying a service profile to a server), the DME first applies that change to the information model and then

applies the change to the actual managed endpoint. This approach is called a *model-driven framework*.

The following is a sample diagram that starts at “sys” from the “topRoot” of the Cisco UCS management information tree and shows a few elements below along with their DNs:

Tree	DN
<a href="#">Click here to view code image</a>   <pre>topRoot └── sys     ├── chassis-1     ├── chassis-2     ├── chassis-3     │   ├── blade-1     │   │   ├── adaptor-1     │   │   └── blade-2     │   │       ├── adaptor-1     │   │       └── host-     │   └── eth-1     └── adaptor-2</pre>	<a href="#">Click here to view code image</a>  <pre>""" "sys" "sys/chassis-1" "sys/chassis-2" "sys/chassis-3" "sys/chassis-3/blade-1" "sys/chassis-3/blade-1/adaptor-1" "sys/chassis-3/blade-2" "sys/chassis-3/blade-2/adaptor-1" "sys/chassis-3/blade-2/adaptor-1/host-eth-1" "sys/chassis-3/blade-2/adaptor-2"</pre>

API methods are split into four categories: authentication methods, query methods, configuration methods, and event subscription methods.

*Authentication methods* are used to authenticate and maintain the session. To perform regular API calls, the XML API cookie needs to be included in the XML structure—this is not a browser cookie! The cookie is obtained and maintained with these methods:

- **aaaLogin**: Logs in and obtains the XML API cookie
- **aaaRefresh**: Refreshes the current authentication cookie
- **aaaLogout**: Exits the current session and deactivates the corresponding authentication cookie

*Query methods* are used to obtain information on the current state of a managed object. Note that the `inHierarchical` argument allows a return of all the child objects, so it should be used with care to avoid very large responses. The following are query examples:

- **configResolveDn**: Retrieves objects by DN
- **configResolveDns**: Retrieves objects by a set of DNs
- **configResolveClass**: Retrieves objects of a given class
- **configResolveClasses**: Retrieves objects of multiple classes
- **configResolveChildren**: Retrieves the child objects of an object
- **configResolveParent**: Retrieves the parent object of an object

*Configuration methods* are used to make configuration changes to managed objects. These changes can be applied to the whole tree, a subtree, or an individual object. Here are some examples:

- **configConfMo**: Configures a single managed object (MO; for example, a DN)
- **configConfMOS**: Configures multiple subtrees (for example, several DNs)

- **configConfMoGroup**: Makes the same configuration changes to multiple subtree structures or MOs.

Finally, we have *event subscription methods*. Applications get state-change information via regular polling or event subscription. For more efficient use of resources, event subscription is the preferred method of notification.

As an example of the UCS API, the following code provisions a new UCS server from a template:

[Click here to view code image](#)

```
import os
import requests
import xmltodict

class UCS_API:
    def __init__(self, host, username, password):
        self.host = host
        self.username = username
        self.password = password
        self.cookie = None

    def api_request(self, body):

        """
        Wrapper for API calls to avoid repeating code.
        Parameter: body is a complete XML request
        Return: dict with the response status and result
        """

        api_response = requests.post(
            f"http://{self.host}/nuova",
            headers={"Content-Type": "application/x-www-form-urlencoded"},
            data=body,
        )
        api_response.raise_for_status()

        status = api_response.status_code          # HTTP status
        data = xmltodict.parse(api_response.text)  # API data as
dictionary rather than XML text

        return (status, data)
```

```

def login(self):
    body = f'<aaaLogin inName="{self.username}" inPassword="'
    body += self.password + '" />'

    response = self.api_request(body)
    if response[0] == 200:
        self.cookie = response[1]["aaaLogin"]["@outCookie"]
    return self.cookie

def logout(self):
    body = f'<aaaLogout inCookie="{self.cookie}" />'
    self.api_request(body)

def create_server_from_profile(self, name, template):
    body = (
        f'<configConfMo dn="" cookie="{self.cookie}">'
        f'  <inConfig>'
        f'    <lsServer dn="org-root/ls-{name}"'
        f'      name="{name}"'
        f'      srcTemplName="{template}">/'
        f'  </inConfig>'
        f'</configConfMo>'
    )
    response = self.api_request(body)
    return response

if __name__ == "__main__":
    UCS_HOST = os.environ.get('UCS_HOST', '10.10.20.113')
    UCS_USER = os.environ.get('UCS_USER', 'ucspe')
    UCS_PASS = os.environ.get('UCS_PASS', 'ucspe')
    ucs = UCS_API(UCS_HOST, UCS_USER, UCS_PASS)

    # supply template and target instance names
    template = "DevNet_ServiceProfile_Template"
    name = "DevNet_ServiceProfile_Instance"

    response = ucs.create_server_from_profile(name, template)

    resp_status = response[1]["configConfMo"]["outConfig"]
    ["lsServer"].get("@status")
    if response[0] == 200 and resp_status == "created":
        print(f'The service profile {name} created successfully.')
    else:
        print(f'The service profile {name} was not created.')

```

```
ucs.logout()
```

## 3.6 Construct a Python script using the Cisco DNA center APIs to retrieve and display wireless health information

Cisco DNA Center Intent APIs provide an easy way for the developer to get an overview of the network health in three categories: Site Health, Network Health, and Client Health. Each API separates the information into more health categories: for example, wired versus wireless, status groups (good, fair, idle, and so on), places in the network (Core, Access, and so on) per site or overall, and so on.

Site Health provides the health of every site (area and building) in Cisco DNA Center. Network Health provides health information by device category (Access, Core, Wireless, and so on). Client Health returns the information by client type (wired or wireless) but also it drills down to show details by category (poor, fair, good, idle, no data, and new).

Cisco DNA Center REST API requires user authentication (Basic authentication scheme) before making API calls. Once authenticated, the user receives a token from the API endpoint, which needs to be included in every request as the `X-Auth-Token` header.

The following sample code performs user authentication followed by the Site and Client Health checks to display network wireless health details per site as well as client health summary per category:

[Click here to view code image](#)

```
import requests  
import json
```

```
# default values for the Cisco DevNet sandbox
SANDBOX = "sandboxdnac2.cisco.com"
USERNAME = "devnetuser"
PASSWORD = "Cisco123!"

class DNAC_API:

    def __init__(self, host, username, password):
        self.system_url = f"https://{host}/dna/system/api/v1"
        self.intent_url = f"https://{host}/dna/intent/api/v1"

        self.username = username
        self.password = password
        self.headers = {
            "Content-Type": "application/json",
            "Accept": "application/json",
        }

        # Obtain Authentication Token and
        # add it to "headers" for the future API calls
        self.headers["X-Auth-Token"] = self.get_token()

    def get_token(self):
        """ Perform Basic Authentication and obtain API Token
        """
        auth = (self.username, self.password)
        auth_resp = requests.post(
            f"{self.system_url}/auth/token",
            auth=auth,
            headers=self.headers,
            verify=False
        )
        auth_resp.raise_for_status()
        return auth_resp.json()["Token"]

    def api_read (self, api_url):
        """ Perform "GET" DNA Center API call """

        response = requests.get(
            f"{self.intent_url}/{api_url}",
            headers=self.headers,
            verify=False
        )
        response.raise_for_status()
        return response.json()
```

```

def main():

    requests.packages.urllib3.disable_warnings()
    dnac = DNAC_API(SANDBOX, USERNAME, PASSWORD)

    print ("*** Network Wireless Health status ***")
    site_health = dnac.api_read("site-health")['response']
    for site in site_health:
        if site['wirelessDeviceTotalCount']:           # skip empty
sites
            print(f"Site: {site['siteName']}:\n"
                  f"  Wireless Network Health:
{site['networkHealthWireless']}% "
                  f"({site['wirelessDeviceGoodCount']})/"
                  f"({site['wirelessDeviceTotalCount']}) OK) \n"
                  f"  Wireless Client Health:
{site['clientHealthWireless']}% "
                  f"for {site['numberOfWirelessClients']} clients"
            )

    print ("\n*** Client Wireless Health status ***")
    client_health = dnac.api_read("client-health")['response'][0]
    for score in client_health["scoreDetail"]:
        print(
            f"Health score for {score['scoreCategory']}"
            f"'value']: "
            f"{score['scoreValue']}% for {score['clientCount']} "
clients"
        )
        # drill down into score categories
        scorelist = score.get('scoreList',{})
        for scoreitem in scorelist:
            if scoreitem['clientCount'] > 0:           # skip empty
ones
            print(
                f"  {scoreitem['scoreCategory']['value']}%"
                f": {scoreitem['clientCount']} clients"
            )

if __name__ == "__main__":
    main()

```

The following is sample output for the Cisco DevNet sandbox:

[Click here to view code image](#)

```
*** Network Wireless Health status ***
Site: MX14:
    Wireless Network Health: 100% (6/6 OK)
    Wireless Client Health: 39% for 33 clients
Site: All Buildings:
    Wireless Network Health: 100% (11/11 OK)
    Wireless Client Health: 29% for 56 clients
Site: HQ:
    Wireless Network Health: 100% (4/4 OK)
    Wireless Client Health: 0% for 17 clients
Site: System Campus:
    Wireless Network Health: 100% (6/6 OK)
    Wireless Client Health: 39% for 33 clients
Site: All Sites:
    Wireless Network Health: 100% (11/11 OK)
    Wireless Client Health: 29% for 56 clients

*** Client Wireless Health status ***
Health score for ALL: 29% for 82 clients
Health score for WIRED: 100% for 2 clients
    GOOD: 2 clients
Health score for WIRELESS: 28% for 80 clients
    FAIR: 58 clients
    GOOD: 22 clients
```

### 3.7 Describe the capabilities of AppDynamics when instrumenting an application

The traditional monitoring approach is siloed: each piece (network, database, docker, web front end, and so on) is monitored and then processed individually. AppDynamics was purposely built to address this and other monitoring challenges.

At a high level, the AppDynamics architecture consists of agents and a controller.

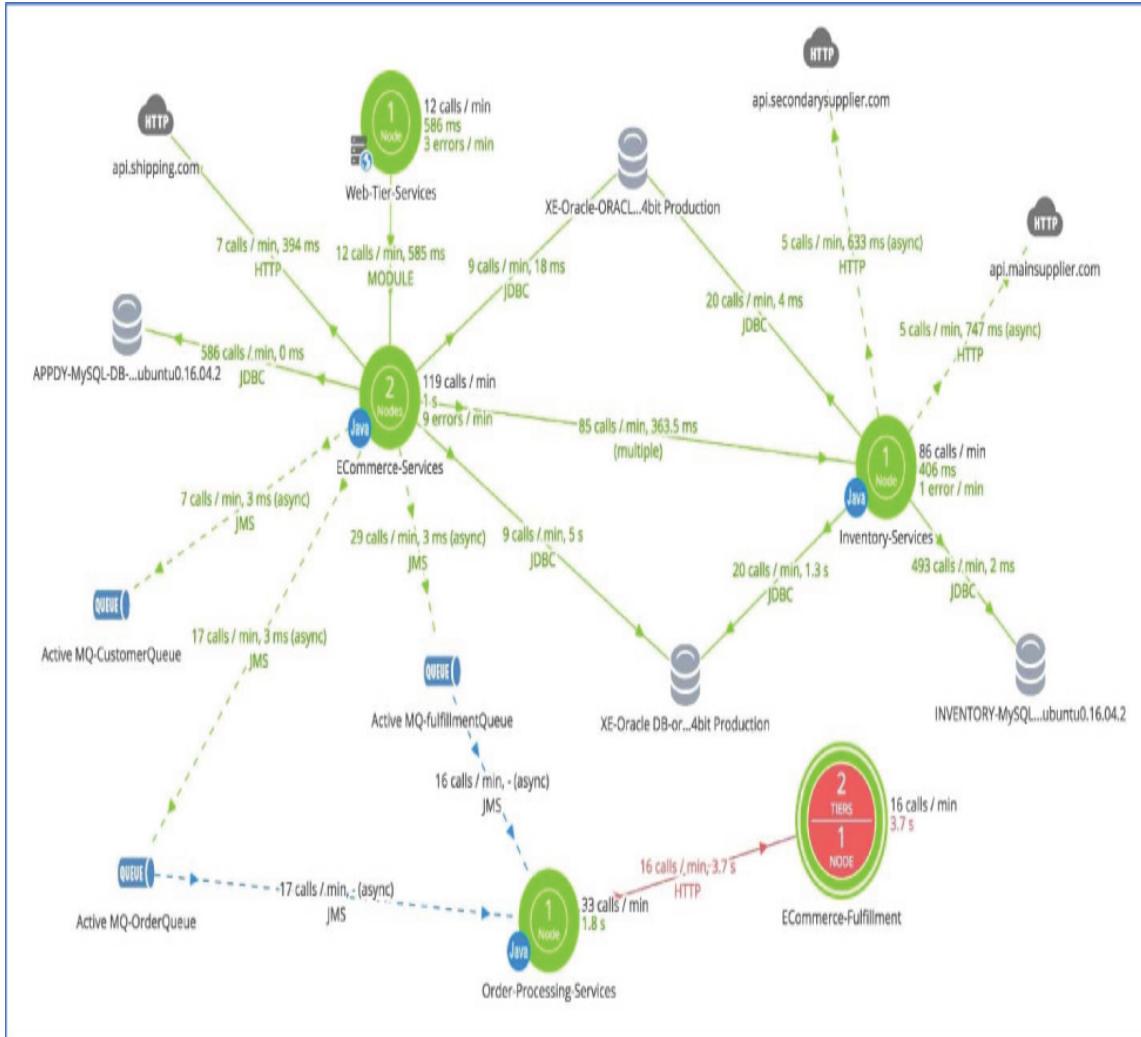
*Agents* are located across the entire application ecosystem (including supporting infrastructure) and monitor the performance and behavior of the application code. Agents monitor every line of code, with unique tags assigned to

every method call and every request header. This allows AppDynamics to trace every transaction from start to finish —even in modern, distributed applications.

The *controller* is updated by agents in real time, even in hyper-complex applications with thousands of agents. The controller helps monitor, troubleshoot, and analyze the entire application landscape, from back-end infrastructure to the end user, in one simple interface.

AppDynamics auto-discovers the flow of all traffic requests in the environment and dynamically creates a topology map to visualize performance across the application ecosystem (see [Figure 30](#) for an example). When baselines are set, status and deviations are represented by red, yellow, and green flow lines.

*Figure 30: AppDynamics Topology Map Example*



AppDynamics automatically discovers business transactions, both known and unknown. A *business transaction* is made up of all the required services within the environment that are called upon to fulfill and deliver a response to a user-initiated request. These are typically things like login, search, checkout, and so on, that will invoke various applications, web services, third-party APIs, and databases.

Business transactions reflect the logical way users interact with your applications. As every line of code executes, AppDynamics monitors traffic patterns and establishes baselines of acceptable performance (see [Figure 31](#)). When

important business transactions (for example, “Add to Cart”) slow down, AppDynamics automatically triggers diagnostic actions and isolates the root cause of any issue.

*Figure 31: AppDynamics Business Transactions View*

Business Transactions										
Details	Filters	Actions	View Options	Configure	Showing 10 of 34					
Name	Original Name	Health	Response Time (ms) ↓	Max Response Time (ms)	Calls	Calls / min	% Errors	Total Errors	CPU Used (ms)	Type
Checkout	ViewCart.sendItems	⚠	2,186	66,205	726	6	6.9	50	59	Struts A...
User Login	User.Login	🔴	1,769	20,899	1,552	13	68.7	1,066	~	Web
Order.update	Order.update	🟢	1,405	11,887	649	5	+	+	4	Struts A...
Fetch Catalog	ViewItems.getAllItems	🔴	484	19,233	968	8	2.8	27	12	Struts A...
Add to Cart	ViewCart.addToCart	🟢	19	383	930	8	6.6	61	19	Struts A...
ViewCart.address	ViewCart.address	🟢	6	275	711	6	+	+	5	Struts A...
ViewCart.confirmorder	ViewCart.confirmorder	🟢	6	239	650	5	+	+	4	Struts A...
ViewCart.paymentinfo	ViewCart.paymentinfo	🟢	6	238	681	6	+	+	5	Struts A...
Homepage	/appdynamicspilot/	🟢	5	31	2,291	19	0	+	4	Servlet
/appdynamicspilot/404.jsp	/appdynamicspilot/404.jsp	🟢	0	15	720	6	+	+	0	Servlet

The AppDynamics APIs let you extend and customize various aspects of the AppDynamics platform. Generally, they can be categorized as platform-side APIs (served by the controller and Events Service) and agent-side APIs:

- **Controller APIs:** Used to administer the controller, to configure, monitor, query metrics, and so on.
- **Analytics Events API:** Used to send your own custom analytics events to the Events Service.
- **Standalone Machine Agent APIs:** Available at the machine agent for uploading custom metrics.

- **Database Agent APIs:** Used to get, create, update, and delete DB Monitoring Collectors.
- **Application Agent Instrumentation APIs:** Language-specific APIs to control and customize transaction detection and correlation, along with exit point detection.
- **Cloud Connector API:** Used to integrate cloud auto-scaling features with new platforms.

### 3.8 Describe steps to build a custom dashboard to present data collected from Cisco APIs

Most modern network solutions are managed by their management systems or controllers, in the case of software-defined networking (SDN). They normally include some kind of “single pane of glass” in the form of GUI dashboards that display some summarized information about the network: how many failures are happening in the network, how many users are connected, how many security incidents happened over the last hour, and so on. These dashboards provide lots of useful insights and often are highly customizable, but sometimes you need to go beyond that (for example, aggregate information from several systems on a single screen).

An efficient solution for this is to build your own dashboard as a custom code that would collect data from network devices, process it, and visualize it for its users.

Collecting data from the network used to be a complicated job; however, most network devices these days provide some sort of API that greatly simplifies this task. More than one API type exists, but the most popular are REST APIs that

use the HTTP Request/Response model and encode data as JSON or XML objects.

A typical process of building a custom dashboard would involve these steps:

- Decide what information will be provided by the dashboard to users.
- Understand what information you need to obtain from the network devices. At this step, you would research API documentation to better understand how to construct your API calls.
- Research API authentication methods.
- Write application code to execute API calls (using the Python “requests” module or specific product SDK libraries), process received data, and store it, if needed.
- Present the data on customized web pages using your favorite web design framework.

## 3.9 Chapter 3 Review Questions

Q1: Which are the elements of the Cisco Webex ChatOps application workflow? (Choose three.)

- A) Create a Cisco Webex chatbot and obtain its access token.
- B) Register a webhook via the Cisco Webex API.
- C) Create an application to handle a webhook call from the Cisco Webex API.
- D) Type a message in the Cisco Webex client.

Q2: Which statement is true about ChatOps implementation with Cisco Webex API?

- A) The ChatOps bot is notified about all the messages in all the rooms it belongs to.
- B) The POST method handler is only needed if the ChatOps bot intends to post messages to Webex rooms.
- C) Webhook notification is JSON-formatted.
- D) The webhook call contains full details about the posted message.

**Q3: Which statement is true about API interactions with Cisco Firepower Device Manager?**

- A) Username and password are included in headers of every API call for authentication.
- B) When creating objects, the “type” field of the object is reflected in the target URL. For example, the URL for the “networkobject” type would be <https://host/api/fdm/latest/object/networks>.
- C) Configuration changes made via an API are immediately activated, so they should be performed with caution.
- D) The FDM API supports both JSON and XML data structures.

**Q4: Which action is *not* a part of the SSID management workflow with the Meraki Dashboard API?**

- A) Read a list of accounts and find a correct AccountId.
- B) Read a list of organizations and find a correct OrganizationId.
- C) Read a list of the networks and find a correct NetworkId.
- D) Read a list of the SSIDs and find a correct SSID.

**Q5: Which statement is true about SSID management with the Meraki Dashboard API?**

- A) Username/password authentication is required to obtain an API token that is used for API calls.
- B) SSIDs are created and updated with the HTTP POST operation.
- C) Reading SSID settings with the HTTP GET, changing needed values only (like “name”, “enabled”, and “authMode”), and sending them back with the HTTP PUT operation is a safe way to use Meraki API.
- D) Setting the “Accept” header to the “application/xml” value directs Meraki API to produce XML data structures.

**Q6: Which statement is true about the Meraki Location API?**

- A) In the Scanning API settings, “Secret” is a string that is used by the Meraki cloud to validate the receiving HTTP server.
- B) The API reports both client’s IP and physical MAC address for simplified user tracking.
- C) The API reports both geo-location (latitude and longitude) and map (X/Y) coordinates.
- D) The API needs to be polled at least once a minute per access point to avoid loss of statistics.

**Q7: Which Intersight API query returns a list of compute resources with the ‘Site’ tag set to ‘Rome’?**

- A) GET /api/v1/compute/RackUnits?\$filter=Site eq ‘Rome’
- B) GET /api/v1/compute/RackUnits?  
\$filter=Tags/any(t:t/Key eq ‘Site’ and t/Value eq  
‘Rome’)
- C) GET /api/v1/compute/RackUnits?\$filter=Tags eq ‘Site’  
and TagValue eq ‘Rome’
- D) GET /api/v1/RackUnits?\$filter=contains(‘Site’,’Rome’)

**Q8: Which statement is true about API interactions with Cisco UCS?**

- A) Username and password are included in headers of every API call for authentication.
- B) The “configConfMo” configuration method can be used to configure one or many managed objects.
- C) API operations are transactional and rolled back completely in case of any error.
- D) The UCS API supports both JSON and XML data structures.

**Q9: Which statement is true about API interactions with Cisco DNA Center?**

- A) Username and password are included in headers of every API call for authentication.
- B) The “/dna/intent/api/v1/site-health?site=HQ” API call returns health information for the HQ site.
- C) The “/dna/intent/api/v1/network-health” API call returns health information about access switches and devices connected to them.
- D) The “/dna/intent/api/v1/client-health” API call returns client health information per client category.

**Q10: Which statement is true about the capabilities of AppDynamics?**

- A) AppDynamics architecture consists of agents, controllers, and applications.
- B) AppDynamics automatically discovers business transactions for the application and establishes baselines for them.
- C) The controller oversees agents and sends them details about which parts of the applications to monitor.

D) The Analytics Events API is a part of the agent-side API group.

---

## *4. Application Deployment and Security*

---

### *4.1 Diagnose a CI/CD pipeline failure (such as missing dependency, incompatible versions of components, and failed tests)*

CI/CD is a method to frequently deliver apps to customers by introducing automation and testing to the integration and delivery/deployment process. This process is typically implemented as *CI/CD pipelines*, which are a series of steps that must be performed in order to deliver a new version of the application.

The “CI” part of CI/CD is *continuous integration*, which is an automation process for developers. Successful CI means new code changes to an app are regularly built, tested, and merged to a shared repository. For the containerized application it means that a new application image is created and tested.

The “CD” in CI/CD refers to continuous delivery and/or continuous deployment. Both are about automating further stages of the pipeline, but they’re used separately to indicate how much automation is happening.

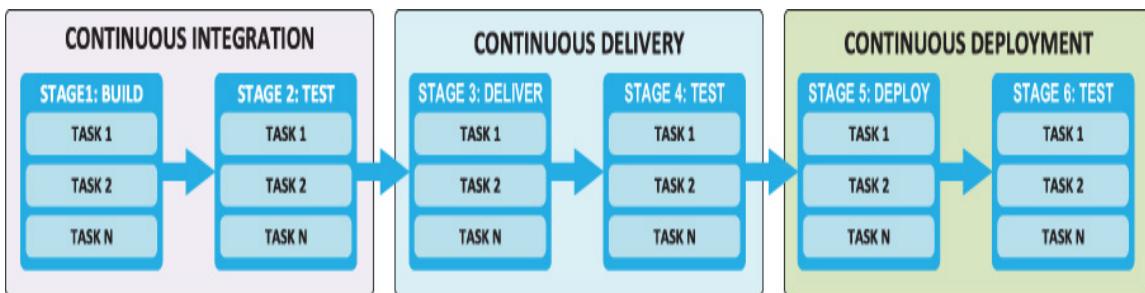
*Continuous delivery* usually means a developer’s changes to an application are automatically bug-tested and uploaded to a repository (like GitHub or a container registry), where they can then be deployed to the production environment by the

operations team. The purpose of continuous delivery is to ensure that deploying new code takes minimal effort.

*Continuous deployment* refers to automatically releasing application changes from the repository to production, where it is immediately usable by customers. It builds on the benefits of continuous delivery by automating the next stage in the pipeline.

CI/CD pipelines are organized into *stages*, which group together distinctive sets of tasks (or jobs), as shown in [Figure 32](#).

*Figure 32: CI/CD Pipeline Example*



CI/CD pipelines may be triggered by some event (for example, repository code commit) or run manually.

Stages are executed sequentially, one after another, while jobs within a stage may run either sequentially or in parallel, depending on CI/CD platform capabilities and settings. Each job either succeeds or fails, and if it fails, the corresponding stage is considered failed and the entire CI/CD process stops with an error. In such a case you need to fix the problem and restart the CI/CD pipeline from the beginning.

The first step in fixing a problem is to understand what went wrong, what stage and what task failed, and why. Depending on the CI/CD platform, you may observe GUI, check pipeline logs, or read notification emails to identify the cause of the failure.

Common failures at the integration stage include the following:

- **Pipeline misconfiguration:** Syntax errors in the pipeline configuration file, logical errors in stages/jobs definitions, and so on.
- **Missing dependency:** The needed module is not installed in the environment, so Python fails to run the application with the ModuleNotFoundError error message. It can be resolved by including the required module in the requirements.txt or “Pipfile” file if you’re using the Pipenv package. Here’s an example:

[Click here to view code image](#)

```
$ python app_test.py
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
ModuleNotFoundError: No module named 'requests'
ERROR: Job failed: exit status 1
```

- **Incompatible versions:** An older version might not include functions or objects that are used in the application code, or variables might be called differently; therefore, this error might exhibit itself with a variety of messages:

[Click here to view code image](#)

```
TypeError: my_module() missing 1 required positional
argument: 'method'
AttributeError: module 'my_module' has no attribute 'APIv2'
NameError: name 'ENV_PASS' is not defined
...
```

Fix it by specifying a correct version of the module in the requirements.txt or “Pipfile” file.

- **Failed unit or integration tests:** Specific functions and the overall application are tested functionally; certain parameters are passed, and the result is

compared with the expected outcome. These tests are usually very customized but are expected to produce a straightforward output of test results and failures. For example, the popular Python “unittest” framework generates “AssertionError” messages:

[Click here to view code image](#)

```
=====
=====
FAIL: test_even (__main__.NumbersTest) (i=1)
-----
-----
Traceback (most recent call last):
  File "subtests.py", line 32, in test_even
    self.assertEqual(i % 2, 0)
AssertionError: 1 != 0
```

Note that code may be correct while the test itself may have errors or wrong logic. This would result in a pipeline failure as well, so tests should be written with care.

To proactively reduce the chance of pipeline failures, developers should first build and test their code locally in the environment as close to the CI/CD environment as possible before committing code to the main repository.

## 4.2 Integrate an application into a prebuilt CD environment leveraging Docker and Kubernetes

Modern applications are shifting from the monolithic to microservices architecture. Rather than one or a few tightly coupled components, microservices applications consist of many small, independently running services, where each component may be developed, deployed, updated, and scaled individually.

Packaging and running these services as containers is called *application containerization*. It's quite easy to deploy and manage individual containers, but when an application consists of dozens of containers, administration becomes more complicated. You need to know at which host to deploy containers to achieve better resource utilization, how to scale them up and down when traffic patterns change, how to handle failures and performance issues, and so on. Doing it all manually is simply impossible; therefore, some automated container management solution is required. The most popular is Kubernetes (often referred to as K8s), developed by Google and now maintained by the Cloud Native Computing Foundation.

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services that facilitates both *declarative configuration* and *automation*. It provides a framework to run distributed systems resiliently, specifically:

- **Service discovery and load balancing:** Kubernetes can expose a container using its DNS name or IP address. If traffic to a container is high, Kubernetes can load-balance and distribute the network traffic so that the deployment is stable.
- **Storage orchestration:** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
- **Automated rollouts and rollbacks:** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers, and adopt all their resources to the new container.

- **Self-healing:** Kubernetes restarts containers that fail, replaces or kills containers that don't respond to health check, and doesn't advertise them to clients until they are ready to serve.
- **Secret and configuration management:** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

It is important to note that Kubernetes provides the building blocks for developer platforms, but its default solutions are optional and pluggable to provide user choice and flexibility where needed.

In contrast to traditional orchestration systems with defined workflows, Kubernetes contains a set of independent, composable control processes that *continuously* drive the current state toward the declaratively provided desired state. It's worth noting what Kubernetes is not:

- It does not limit the types of supported applications. As soon as an application can run in a container, it should run great on Kubernetes.
- *It does not deploy source code and does not build your application.* CI/CD workflows are determined by organization cultures and preferences as well as technical requirements.
- It does not provide application-level services, such as middleware, databases, caches, cluster storage, and so on, as built-in services. However, such components can run *on* Kubernetes.

- It does not dictate logging, monitoring, or alerting solutions.
- It does not provide or mandate a configuration language/system but provides a declarative API.

## Kubernetes Components

A *pod* is the smallest deployable unit of computing that you can create and manage in Kubernetes. It is a group of one or more containers, with shared storage/network resources, and a specification for how to run the containers. A pod's containers are always co-located and co-scheduled and run in a shared context. When you scale in Kubernetes, you scale the pods, and having multiple containers inside the same pod means that they cannot scale separately.

Therefore, a common use case is to have only one container per pod.

When you deploy Kubernetes, you get a *cluster*. A Kubernetes cluster consists of a set of worker machines, called *nodes*, that run containerized applications. Every cluster has at least one *worker node*. Worker nodes host *pods*.

The control plane runs on *master* nodes and manages *worker* nodes and pods in the cluster. In the production environments, the control plane usually runs across multiple computers, and the cluster usually runs multiple nodes, providing fault-tolerance and high availability.

A control plane's components make global decisions about the cluster; they are as follows:

- **kube-apiserver:** A component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

- **etcd:** A consistent and highly available key-value store used for all the cluster data.
- **kube-scheduler:** Watches for newly created pods with no assigned node and selects a node for them to run on, taking into account individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.
- **kube-controller-manager:** Runs *controller* processes. Controllers watch the state of the cluster, then make (or request) changes where needed, trying to move the current cluster state closer to the desired state.

Node components run on every node, maintaining running pods; they are as follows:

- **kubelet:** An agent that runs on each node in the cluster. It takes a set of *PodSpecs* that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy.
- **kube-proxy:** A network proxy that runs on each node in the cluster, implementing part of the Kubernetes Service concept. It maintains network rules that allow network communication to the pods from inside or outside of the cluster.
- **Container runtime:** Software that is responsible for running containers (for example, Docker).

## **Understanding Kubernetes Objects**

Kubernetes objects are persistent entities in the Kubernetes system, used to represent the state of the cluster.

Specifically, they can describe the following:

- What containerized applications are running (and on which nodes)
- Resources available to those applications
- Policies around how those applications behave, such as restart policies, upgrades, and fault tolerance

By creating a Kubernetes object, you're effectively telling the Kubernetes system what you want your cluster's workload to look like; this is your cluster's desired state.

Work with Kubernetes objects, whether to create, modify, or delete them, is always performed via the Kubernetes API. For example, when using the `kubectl` command-line interface, it makes the necessary Kubernetes API calls to execute commands. The Kubernetes API may be used directly in your programs with the help of one of the client libraries.

Almost every Kubernetes object includes two nested object fields that govern the object's configuration: the *object spec* and the *object status*. Object spec is set when the object is created, describing the characteristics a resource should have: its *desired state*.

The status describes the *current state* of the object, supplied and updated by the Kubernetes system. The Kubernetes control plane continually and actively manages every object's actual state to match the supplied desired state.

In the following example, you can see the Deployment object that represents the “devcor-app” application running on the cluster. The Deployment spec declares that *three* replicas of the application need to be running. The Kubernetes system reads the Deployment spec and starts three instances of the application, updating the status to

match this spec. If any of those instances should fail (a status change), the Kubernetes system will respond to the difference between the spec and status by making a correction. In such a case, it would start a replacement instance.

To create an object, most often configuration is put into a .yaml file and then applied to resources using the `kubectl apply -f <filename>` command. Here's an example:

[Click here to view code image](#)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: devcor-app
spec:
  replicas: 3 # tells deployment to run 3 pods matching the
template:
  selector:
    matchLabels:
      app: devcor-app
  template:
    metadata:
      labels:
        app: devcor-app
    spec:
      containers:
        - name: devcor-container
          image: devcor-container:latest
          ports:
            - containerPort: 5000
```

Although Deployment objects represent applications, *Service* is an abstract way to expose an application as a network service. Pods are dynamic (they can die or move), but Service exposes a single static address that other containers can rely on. For example, the following Service opens external HTTP access:

[Click here to view code image](#)

```
apiVersion: v1
kind: Service
```

```
metadata:
  name: devcor-service
spec:
  selector:
    app: devcor-app
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 5000
  externalIPs:
    - 192.0.2.100
```

## CI/CD Integration

CI/CD is a set of practices that automate the application integration and delivery steps and standardize application configurations. Every time developers make changes to the application code and check it in into the version control system, CI/CD pipelines run a sequence of builds, tests, data migrations, application deployments, service calls, and other scripted procedures to make these code changes available in targeted environments.

Docker (or containers in general) and Kubernetes are two very important building blocks in declarative, quick, and reliable application delivery.

Typically, the Continuous Delivery pipeline ends with the Staging stage. The idea of this stage is to deploy a new build on a staging server so it can be inspected to see if it works properly. Final deployment to the production environment will be done periodically and manually.

If your environment is a Kubernetes cluster, then your application must be containerized before it can be deployed into the cluster. This container packaging can be done as part of the Staging stage or in an entirely separate stage. How to build containers with Docker is explained in [Section 4.4: Utilize Docker to containerize an application](#). The end

result of this stage is a new container image (or images) available in the container repository.

To complete application deployments, you need to prepare appropriate Kubernetes Deployment and Service objects and push them in a job to the K8s cluster. Kubernetes will update the deployment to the desired state in updated objects. Note that when you update an application to a newer version, *rolling updates* will incrementally update pod instances with new ones, one by one, with zero downtime.

### 4.3 Describe the benefits of continuous testing and static code analysis in a CI pipeline

*Continuous testing* is a software testing type in which the product is evaluated early, often, and throughout the entire CI/CD process. Continuous testing uses automated tests to ensure developers receive immediate feedback to quickly fix bugs and mitigate as many risks as possible before software is released to production.

The benefits of continuous testing are as follows:

- Improved code quality, as errors are found before being released to production
- Accelerated software delivery due to continuous feedback mechanism
- Accelerated testing, as automated tests run in parallel
- An agile and reliable process taking just hours instead of weeks and months
- Eliminates the disconnect between traditionally siloed development, testing, and operations teams

- Improved customer loyalty due to continuous improvements and better quality
- Reduced business risks, as potential problems are assessed before they become real

Traditional testing, where code is executed and the output is validated against the expected outcome, is called *dynamic testing*. A different type of testing, where source code is examined for quality, reliability, and security without executing it is called *static code analysis*.

Static code analysis complements dynamic testing to provide additional advantages:

- **Error detection:** Static code analysis can identify hundreds of classes of bugs related to concurrency, tainted data, data flow, and static and dynamic memory. Some bugs are nearly impossible to detect with the dynamic testing.
- **Security vulnerabilities detection:** Static code analysis can detect common vulnerabilities, such as those identified by OWASP, in the code and imported libraries.
- **Low cost:** Static code analysis may be easily automated without the overhead of writing test cases, instrumenting the code, and program execution.
- **Coding standards compliance:** Static analysis tools can analyze source syntax and enforce coding standards.
- **Better source code:** Static code analysis tools can identify the unused code.

## 4.4 Utilize Docker to containerize an application

There are two popular approaches to application virtualization: using virtual machines (VMs) and using containers.

VMs emulate the whole computer system and are very efficient compared to running each application on its own physical computer. VMs, however, consume lots of system resources because they need to run a virtual copy of all the hardware they need and a full separate copy of an operating system (OS). For some applications this approach may be overkill; that's why containers were developed.

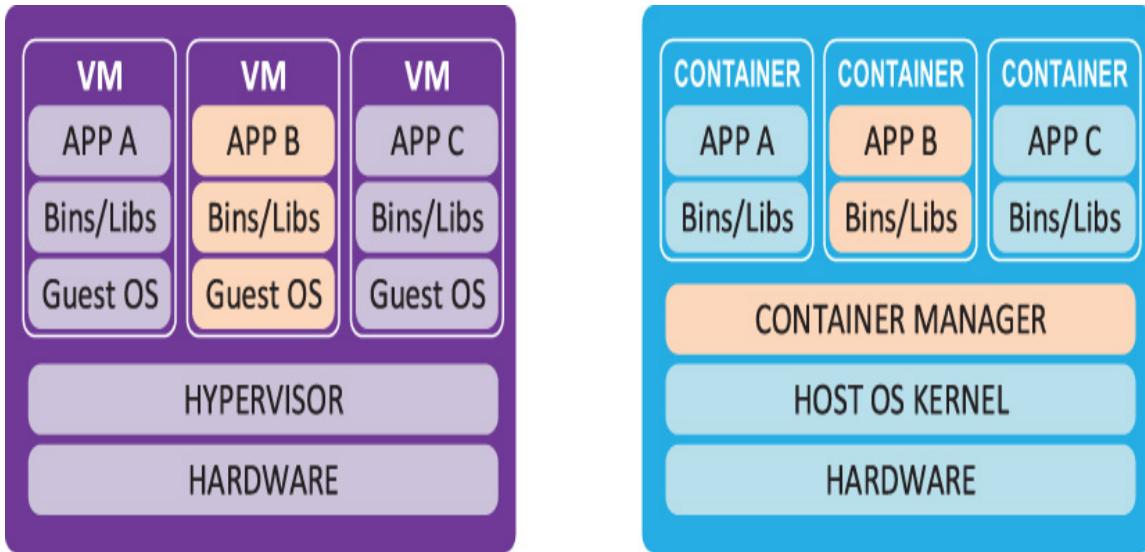
A container is a lightweight, standalone package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings.

Multiple containers can run on the same host as well as share (read-only) the host OS kernel with other containers, each running as an isolated process in user space. Sharing these resources means there's no need to include them in the container.

As a result, containers are exceptionally light compared to equivalent VMs: their size may be just megabytes, and they take just seconds to start. Unlike VMs, containers share the host OS kernel, so you cannot, for example, run a Windows container on a Linux box (but you can run a container with CentOS on the Ubuntu box).

These features make containers much better suited for the 12-factor applications (see [Section 4.5](#)). [Figure 33](#) provides a comparison of containers and VMs.

*Figure 33: Containers vs. Virtual Machines (VMs)*



One of the most popular tools to work with containers is *Docker*, which is a software platform that provides a simple way of building, running, managing, and distributing container applications.

*Docker Engine* is a core component of Docker responsible for the overall functioning of the platform. It is a client/server-based application and consists of three main components:

- *Server* runs a daemon known as “dockerd,” which is responsible for creating and managing Docker images, containers, networks, and volumes on the Docker platform.
- *Rest API* specifies how the applications can interact with the server
- *Client* is a command-line interface that allows users to interact with Docker.

*Docker Image* is a template that contains the application and all the dependencies required to run that application on Docker. *Docker Container* is a running instance of Docker Image.

Note: Visit the live online Docker playground to explore Docker and its functionality at <https://labs.play-with-docker.com/>.

To run a container, use the `docker run` command with several parameters, including the image name:

[Click here to view code image](#)

```
$ docker run -it python
Python 3.8.5 (default, Sep 1 2020, 18:44:24)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> print ("Hello, docker!")
Hello, docker!
>>>
```

Another example, `docker run -dit --name web-container -p 8080:80 httpd:2.4` would start a local web server in just a few seconds. Here are some frequently used parameters for `docker run`:

- **-d (detached)**: Run container in the background (by default, it would run in the foreground).
- **-i (interactive)**: Attach STDIN (that is, allow user interaction).
- **-t (terminal)**: Allocate pseudo-terminal (no screen output otherwise). For the interactive terminal session, it is required to use both `-i` and `-t` (or `-it`).
- **--name: container name**: Optional (otherwise, the daemon will generate a random string name).
- **-p (publish)**: Expose the incoming port. Map a public port to a local port on a container (8080 to 80 in the preceding example).
- **<image>**: Docker image name in the `<image>:<version>` format. Omit the `<version>` part to use the latest version

(for example, `python` is equivalent to `python:latest`).

The last parameter may be used to specify the program to be executed instead of image's default:

[Click here to view code image](#)

```
$ docker run -it python bash
root@c53f5f79a8b1:~# echo "Hello, bash!"
Hello, bash!
```

The `docker ps` command may be used to check the status of running containers or obtain container IDs:

[Click here to view code image](#)

```
$ docker ps
CONTAINER ID        IMAGE       COMMAND       CREATED             NAMES
88c7dd2c5604        python      "python3"     5 seconds ago      suspicious_brown
Up 4 seconds
a7d3709d880b        httpd       "httpd-foreground"   27 seconds ago     stoic_cartwright
Up 25 seconds
0.0.0.0:8080->80/tcp
```

To attach to a running container, use the `docker attach <container ID>` command. To stop the container, run the `docker stop <container ID>` command (or just exit from the interactive one).

## Docker Images

All images are stored in a local image cache. If Docker Image is not available locally, it will be first downloaded from the remote image repository (Docker Hub by default) on the fly. Here's an example:

[Click here to view code image](#)

```
$ docker run -it python
Unable to find image 'python:latest' locally
latest: Pulling from library/python
d6ff36c9ec48: Pull complete
c958d65b3090: Pull complete
```

```
edaf0a6b092f: Pull complete
80931cf68816: Pull complete
7dc5581457b1: Pull complete
87013dc371d5: Pull complete
dbb5b2d86fe3: Pull complete
4cb6f1e38c2d: Pull complete
c2df8846f270: Pull complete
Digest:
sha256:bc765f71aaa90648de6cfa356ec201d50549031a244f48f8f477f3865
17c5d1b
Status: Downloaded newer image for python:latest
Python 3.8.5 (default, Sep 1 2020, 18:44:24)
```

To download an image in advance, use the `docker pull <image>` command.

Docker images are not monolithic; they consist of several layers. These layers (also called intermediate images) are generated when the commands in the *Dockerfile* are executed during the Docker Image build. Layers can be reused by multiple images, thus saving disk space and reducing time to build images.

In the preceding Python example, you can see that each layer is downloaded independently. To view all the layers that make up the image, use the `docker history <image>` command:

[Click here to view code image](#)

```
$ docker history python
IMAGE                  CREATED          CREATED BY
SIZE
a7cd474cef4        3 days ago      /bin/sh -c #(nop)  CMD
["python3"]
<missing>            3 days ago      /bin/sh -c set -ex;
wget -O get-pip.py "$P...  7.24MB
<missing>            3 days ago      /bin/sh -c #(nop)  ENV
PYTHON_GET_PIP_SHA256... 0B
<missing>            3 days ago      /bin/sh -c #(nop)  ENV
PYTHON_GET_PIP_URL=ht... 0B
<missing>            3 days ago      /bin/sh -c #(nop)  ENV
PYTHON_PIP_VERSION=20... 0B
<missing>            3 days ago      /bin/sh -c cd
/usr/local/bin  && ln -s idle3...  32B
```

```

<missing>          3 days ago      /bin/sh -c set -ex  &&
wget -O python.tar.x... 53MB
<missing>          3 days ago      /bin/sh -c #(nop) ENV
PYTHON_VERSION=3.8.5    0B
<missing>          3 days ago      /bin/sh -c #(nop) ENV
GPG_KEY=E3FF2839C048B... 0B
<missing>          3 days ago      /bin/sh -c apt-get
update && apt-get install... 17.9MB
<missing>          3 days ago      /bin/sh -c #(nop) ENV
LANG=C.UTF-8          0B
<missing>          3 days ago      /bin/sh -c #(nop) ENV
PATH=/usr/local/bin:/... 0B
<missing>          3 days ago      /bin/sh -c set -ex;
apt-get update; apt-ge... 510MB
<missing>          4 weeks ago     /bin/sh -c apt-get
update && apt-get install... 146MB
<missing>          4 weeks ago     /bin/sh -c set -ex; if
! command -v gpg > /... 17.5MB
<missing>          4 weeks ago     /bin/sh -c apt-get
update && apt-get install... 16.5MB
<missing>          4 weeks ago     /bin/sh -c #(nop) CMD
["bash"]            0B
<missing>          4 weeks ago     /bin/sh -c #(nop) ADD
file:4b03b5f551e3fbdf4... 114MB

```

As you can see, each layer is made up of files generated from running a Dockerfile command during image build. All layers in a Docker image are read-only except the top one. The top layer is read/write, and it gives the appearance that you can modify the image, but the read-only layers below actually maintain their and the container's integrity.

Public image repositories provide lots of useful images, but very often you need to create your own either by modifying the existing image or by creating a new one from scratch.

The `docker build` command creates an image from a Dockerfile and a context:

[Click here to view code image](#)

```
docker build -t devnet/sample -f ~/Dockerfile <context>
```

- **-t**: Specifies a repository and tag at which to save the new image if the build succeeds.
- **-f**: A path to the Dockerfile (by default, at the root of the *context*).
- **<context>**: The directory exposed to the Docker daemon. The main use case is to allow copy files from the host system to the Docker image with the COPY instruction.

Note that the build is run by the Docker daemon, not by the command-line interface (CLI) tool, and the first thing a build process does is send the entire context (recursively) to the daemon. The build will be slow if a context is large (don't use Linux "/" root directory as the context!), so in most cases, it is best to start with an empty directory as the context and add only the needed files to it. Another option to control the size of the context and increase performance is to exclude files and directories from it using the **.dockerignore** file.

A *Dockerfile* is a text document that contains the following instructions for assembling an application image:

- **FROM <image>**: A Dockerfile must begin with a FROM instruction. The FROM instruction initializes a new build stage and specifies the base image for subsequent instructions. It may be an existing image from the public repository or a reserved, minimal image named **scratch**.
- **ENV <key> <value>**: Sets the environment variable `<key>` to the value `<value>` within the container.
- **WORKDIR <path>**: Sets the working directory for any subsequent RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.

- **COPY <src> <dst>**: Copies new files or directories from `<src>` on the host to the `<dest>` on the container file system.
- **RUN <command>**: Executes a command in a new layer on top of the current image.
- **EXPOSE <port>**: Does not publish the port but functions as documentation between the person who builds the image and the person who runs the container. To publish the port when running the container, use the `-p` flag with `docker run`.
- **ENTRYPOINT / CMD**: Default arguments for an executing container (see the following example for details).

Here's an example:

[Click here to view code image](#)

```
FROM python:3.7

ENV CONTROLLER 10.1.1.100
ENV USER_NAME dev_admin

WORKDIR /usr/src/app

COPY package.json ./packages
COPY . .

RUN pip install -r requirements.txt

EXPOSE 8080

ENTRYPOINT [ "python3", "-E"]
CMD [ "app.py" ]
```

**Here are the** `ENTRYPOINT` and `CMD` rules:

- At least one of them must be present. Normally you can specify parameters in both and they will add up

(`python3 -E app.py` would be executed in the preceding example).

- There are two forms for both:

- **exec form:** Just run the executable; it will get PID 1 and react to `docker stop`. If `CMD` is present, it will be added to the list of arguments.

```
ENTRYPOINT/CMD ["executable", "param1", "param2"]
```

- **shell form:** Will be run as `/bin/sh -c command param1 param2`, PID 1 will be the shell, so it won't respond gracefully to `docker stop` (for example, `SIGTERM` would be ignored and then followed by the ungraceful `SIGKILL`). The `CMD` parameter is ignored (unless there's no `ENTRYPOINT`). Here's an example:

```
ENTRYPOINT/CMD command param1 param2
```

- The `CMD` parameters are “default,” so if command-line arguments are supplied to `docker run`, they override the `CMD` part.

## Docker Storage

By default, all files created inside a container are stored on a writable container layer. This means the following:

- The data doesn't persist when that container no longer exists, and it can be difficult to get the data out of the container if another process needs it.
- A container's writable layer is tightly coupled to the host machine where the container is running. You can't easily move the data somewhere else.

Docker has two options for containers to store files in the host machine so that the files are persisted even after the container stops—volumes and bind mounts:

- *Volumes* are stored in a part of the host file system that is managed by the Docker (`/var/lib/docker/volumes/` on Linux). Non-Docker processes should not modify this part of the file system. Volumes are the best way to persist data in Docker.
- *Bind mounts* may be anywhere on the host system; they can even be important system files or directories. Both non-Docker processes and a Docker container can modify them at any time.

In both cases, the storage data looks the same from within the container.

## Docker Networking

Docker's networking subsystem is pluggable using these drivers that provide core networking functionality:

- **bridge**: The default driver; usually used when your applications run in standalone containers that need to communicate
- **host**: Used to remove network isolation between the container and the Docker host, and use the host's networking directly
- **overlay**: Used to build a distributed network among multiple Docker daemon hosts
- **macvlan**: Used to route traffic to containers by their MAC addresses (for legacy apps mainly)
- **none**: Used to disable all networking

*Bridge networks* allow containers connected to the same bridge network to communicate while providing isolation from containers that are not connected to that bridge network. The Docker bridge driver automatically installs rules in the host machine so that containers on different

bridge networks cannot communicate directly with each other.

When you start Docker, a default bridge is created automatically, and all new containers connect to it unless otherwise specified. You can also create user-defined custom bridge networks.

User-defined bridge networks have the following benefits compared to the default bridge network:

- Automatic DNS resolution between containers.  
Containers on the default bridge network can only access each other by IP addresses. On a user-defined bridge network, containers can resolve each other by name or alias.
- Better isolation, as only explicitly added containers will be able to communicate with hosts on the user-defined bridge network.
- Containers can be attached and detached from user-defined networks on the fly. To remove a container from the default bridge network, you need to stop the container and re-create it with different network options.

## 4.5 Describe the tenets of the “12-factor app”

The 12-factor app methodology is for building software-as-a-service applications. These best practices are designed to enable applications to be built with portability and resilience when deployed to the Web.

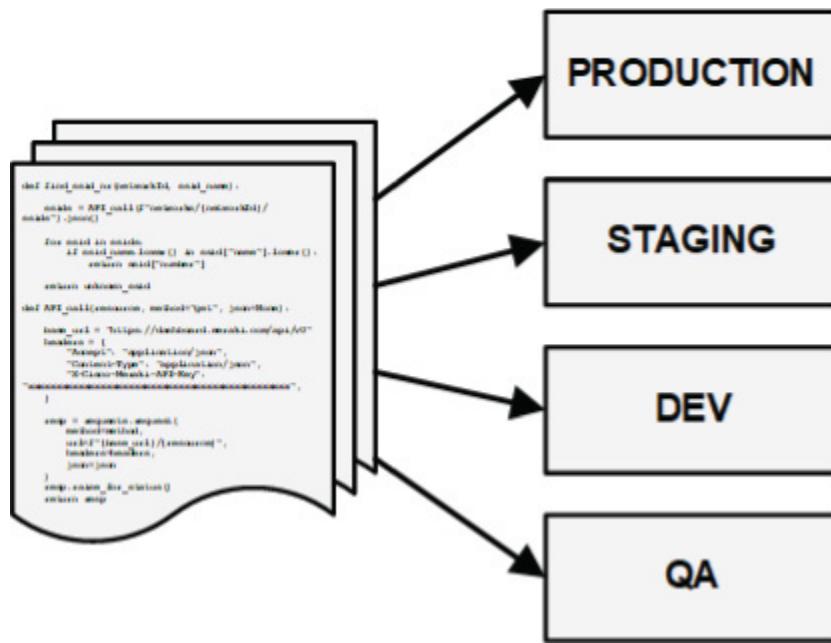
### **Factor I: Codebase**

*One codebase tracked in revision control, many deploys.*

There should be exactly one codebase for a deployed service, with that codebase being used for many deployments.

If there are multiple codebases, it's not an app; it's a distributed system. Then each component is an app, and each can individually comply with the 12-factor methodology.

Multiple apps sharing the same code is a violation of the 12-factor methodology (do libraries instead).



## Factor II: Dependencies

*Explicitly declare and isolate dependencies.*

All dependencies should be explicitly declared, with no implicit reliance on system tools or libraries.

The benefit of the explicit dependency declaration is that it simplifies setup for new developers. The only prerequisites to start development are programming language runtime and dependency manager. The developer checks out the app's codebase onto their development machine and runs a

deterministic build command to set up everything needed to run the app's code.

Example for Python: create a virtual environment, pull the code, and run `pip -r requirements.txt`.

## Factor III: Config

*Store config in the environment.*

Config is everything likely to vary between deploys: credentials, resource handles, and per-deploy values. The 12-factor app requires strict separation of config from code and stores config in *environment variables* (or *env vars*).

Environment variables are easy to change between deploys without changing any code and, unlike config files, there is a low chance of them being checked into the code repo accidentally.

## Factor IV: Backing Services

*Treat backing services as attached resources.*

A backing service is *any* service the app consumes over the network during its normal operation. Examples include data stores (MySQL), SMTP service (Postfix), and message/queue system (RabbitMQ).

The code for a 12-factor app makes no distinction between local and third-party services. To the app, both are *attached resources*, accessed via a URL or other locator/credentials stored in the config. One resource may be replaced with another without any change to the app code (for example, local MySQL to Amazon RDS) at will.

## Factor V: Build, Release, Run

*Strictly separate build and run stages.*

The delivery pipeline should strictly consist of separate build, release, and run stages. No changes are allowed in the running release. If an update is required, apply it to the source code and repeat all the stages.

## **Factor VI: Processes**

*Execute the app as one or more stateless processes.*

Twelve-factor processes are stateless and share nothing. Any data that needs to persist must be stored in a stateful backing service (typically a database).

## **Factor VII: Port Binding**

*Export services via port binding.*

The 12-factor app is completely self-contained and does not rely on runtime injection of a web server to create a web-facing service. The app provides HTTP service by binding to a port and listening to requests coming in on that port (the PHP module for Apache is not a 12-factor app!).

## **Factor VIII: Concurrency**

*Scale out via the process model.*

The 12-factor app is process-oriented. Developers can architect their app to handle diverse workloads by assigning each type of work to a certain process type. The process model allows easy independent scale-out for each individual process type.

(In other words, build your app as a set of concurrent processes rather than some monolith.)

## **Factor IX: Disposability**

*Maximize robustness with fast startup and graceful shutdown.*

The 12-factor app's processes are disposable, meaning they can be started or stopped at a moment's notice.

Processes should strive to minimize startup time to provide more agility and robustness.

Processes shut down gracefully when they receive a `SIGTERM` signal but are architected to handle unexpected, non-graceful terminations as well.

## Factor X: Dev/Prod Parity

*Keep development, staging, and production as similar as possible.*

The 12-factor app is designed for continuous deployment by keeping the gap between development and production small:

- **Make the time gap small:** A developer may write code and have it deployed hours or even just minutes later.
- **Make the personnel gap small:** Developers who wrote the code are closely involved in deploying it and are watching its behavior in production.
- **Make the tools gap small:** Keep development and production as similar as possible.

## Factor XI: Logs

*Treat logs as event streams.*

Logs have no fixed beginning or end but flow continuously as long as the app is operating. A 12-factor app never concerns itself with routing or storage of its output stream. It should not attempt to write to or manage log files. Instead, each running process writes its event stream, unbuffered, to `stdout`.

In staging and production deploys, each process's stream will be captured by the execution environment, collated together with all other streams from the app, and routed to one or more final destinations for viewing and long-term archival. These archival destinations are *not visible* to or configurable by the app.

## Factor XII: Admin Processes

*Run admin/management tasks as one-off processes.*

One-off admin processes should be run in an identical environment as the regular long-running processes of the app. They run against a release, using the same codebase and config as any process run against that release. Admin code must ship with application code to avoid synchronization issues.

## 4.6 Describe an effective logging strategy for an application

Logs are a very efficient component of the application observability framework. They are an important tool used for troubleshooting and finding errors and their root causes. It's very easy to implement logging in a monolithic application, which may be as simple as placing some `print()` statements throughout the code. However, it's better to use an established a logging framework (for example, the "logging" module) to deliver uniform results:

[Click here to view code image](#)

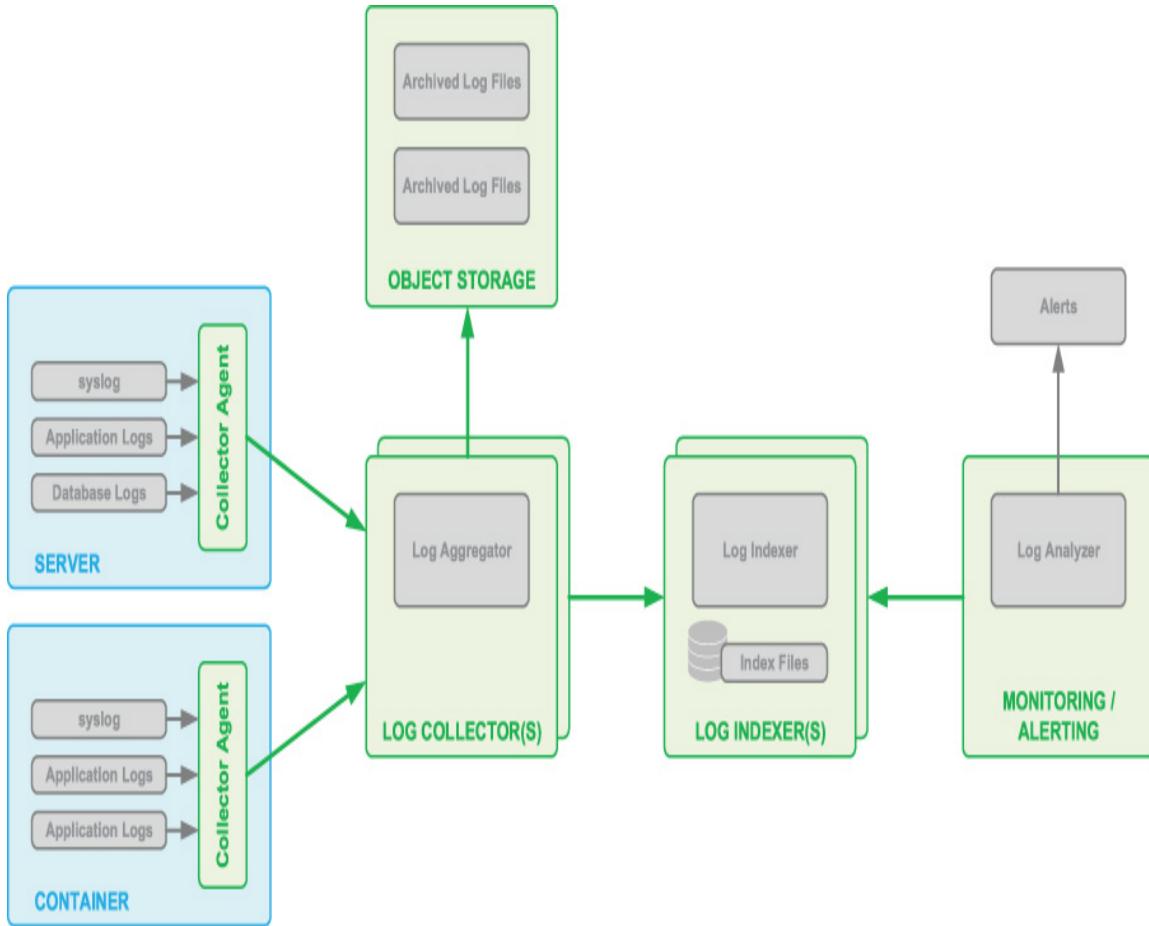
```
import logging
logger = logging.getLogger(app_name)
logger.setLevel(logging.INFO)
logger.info(f'Starting {app_name} in a {run_mode} mode')
```

For logging to be effective, provide as many details about errors and program state in log messages as possible: timestamps, log levels, variable values, and so on.

With distributed applications, efficient logging is not so trivial, as each component and each container instance (which may be very short-lived!) would be producing its own log data. Logs become distributed as well, and now it's unfeasible to connect to each component and manually check each individual log file. The solution to this problem is to send all the generated logs to an external, centralized place where they would be aggregated and stored. This is the goal of the distributed logging: to provide one place where all the logs are easily found.

Distributed logging is divided into five stages—collection, forwarding, storage, indexing, and alerting—as shown in [Figure 34](#) and described in the following list:

*Figure 34: Distributed Logging*



- **Collection:** Collector agent(s) watch the log file for changes, subscribe to the logging system (for example, Syslog), or offer an API for applications to push log entries.
- **Forwarding:** Collected logs forwarded to a log aggregator. The log aggregator may be a standalone application or just another collection agent (for example, Syslog accepting network logs).
- **Storage:** Store logs on any local/remote/shared storage (files, databases, and clusters). Policies applied to filters, granularity, retention time, and so on.
- **Indexing (optional):** Logs are indexed into different data sets to allow for faster log searches. Indexed logs

are then more easily consumed by searching and analytics tools.

- **Alerting (optional):** Checks various thresholds and anomalies and raises alerts, if needed.

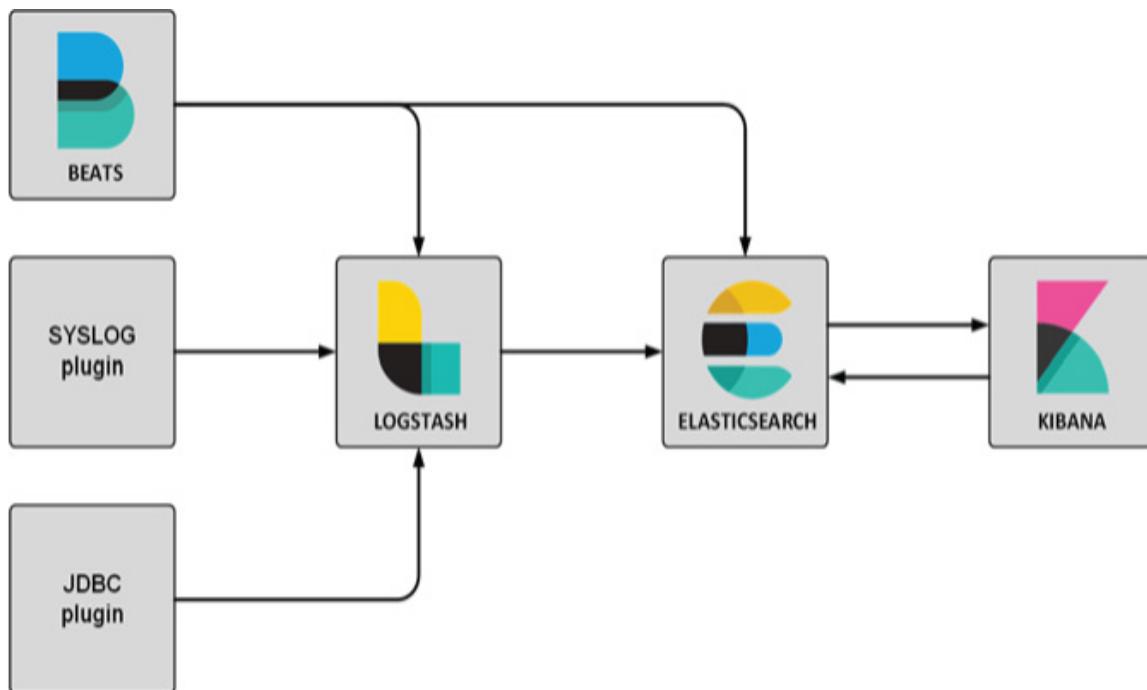
Distributed logging provides best results when best practices are followed:

- Aggregate system, database, event, and other logs as well, as this would provide more visibility during troubleshooting (did the container crash due to low memory or database failure?).
- Tracing request flow through many containers is quite complicated. Generate a *correlation ID* when making the first microservice call and pass the same correlation ID to the downstream services. Log the correlation ID across all the microservice calls. Use the ID to filter logs and quickly trace application flow.
- Different components might have very different logging formats. To allow efficient search and analytics, it's best to preprocess log messages and transform them into some common standard before aggregating.
- Even with the Network Time Protocol (NTP), there will be some time difference between different systems. Pick a single source for the log's time (for example, add the aggregator's time to every processed log message).
- Include details and context in log messages; the more information available, the easier it is to troubleshoot.
- Logging is a software element and could fail. Applications should try to handle this properly (for example, keep a local log file or cache log messages until the logging service is back up).

The ELK (Elasticsearch, Logstash, Kibana) stack, illustrated in [Figure 35](#), is a popular modern solution for the distributed logging:

- Elasticsearch is a distributed search and analytics engine for all types of data.
- Logstash is a data processing pipeline that ingests data from a multitude of sources, transforms it, and then sends it to your favorite “stash,” regardless of format or complexity.
- Kibana lets you visualize your Elasticsearch data. Do anything from tracking query load to understanding the way requests flow through your apps.
- Beats (Filebeat, Metricbeat, Winlogbeat, and so on) are data shipper agents installed on servers to send operational data to Elasticsearch, either directly or via Logstash.

*Figure 35: ELK Stack*



## 4.7 Explain data privacy concerns related to storage and transmission of data

*Data privacy* describes the practices that ensure the data shared by customers is only used for its intended purpose. To protect data privacy means to guarantee confidentiality, integrity, and authenticated access to that data, including personally identifiable information (PII) collection and use considerations.

Data privacy laws govern specific types of data, such as the United States Health Insurance Portability and Accountability Act (HIPAA), Electronic Communications Privacy Act (ECPA), and Children's Online Privacy Protection Act (COPPA). The most significant standard about PII processing regulations is the EU's General Data Protection Regulation (GDPR) because it applies to Europe and to any other country that wants to provide services to individuals within the EU, and as such it extends many of its data privacy safeguards globally.

Companies implement their own standards to ensure the protection of the industry secrets and critical private documents that must not be disclosed.

PII is any data that can be used to clearly identify an individual, or, as GDPR defines it:

- '*[P]ersonal data*' means any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person.

Here are some other important GDPR definitions:

- '*Consent*' of the data subject means any freely given, specific, informed, and unambiguous indication of the data subject's wishes by which he or she, by a statement or by a clear affirmative action, signifies agreement to the processing of personal data relating to him or her.

This means the users must be clearly informed about the use of their personal data, and they must explicitly consent to that usage. Website cookies are PII; that's why these days many websites request user consent to use them.

- '*[P]ersonal data breach*' means a breach of security leading to the accidental or unlawful destruction, loss, alteration, unauthorised disclosure of, or access to, personal data transmitted, stored or otherwise processed.

This means that any time there is any loss, modification, or PII data theft, users have to be informed, despite a potential loss of reputation for the victim of the breach.

GDPR defines six principles relating to the processing of personal data:

- **Lawfulness, fairness, and transparency:** Data is to be used lawfully, fairly, and in a transparent manner.
- **Purpose limitation:** Data is to be collected for specified, explicit, and legitimate purposes.
- **Data minimization:** Data is to be relevant and limited to what is necessary in relation to the purposes for which it is collected. (You cannot collect more data than you need to provide a service.)

- **Accuracy:** Data is to be accurate and, where necessary, kept up to date.
- **Storage limitation:** Data should be kept for no longer than it is necessary.
- **Integrity and confidentiality:** You need to ensure the security of the personal data.

GDPR demands that stored data undergo either an anonymization or a pseudonymization (deidentification) process. In both cases, PII data is removed; however, anonymization irreversibly destroys any way of identifying the data subject, whereas pseudonymization substitutes the identity of the data subject in such a way that with some additional information it is possible to re-identify the data subject. In other words, with pseudonymization, some unique ID is still present, but it's not personally identifiable. Pseudonymization is less secure, as it allows re-identification either as a result of some authority's request or due to advanced analytics (for example, from knowing that same person went to certain places, it is possible to deduce who he/she is).

Encryption, integrity, and nonrepudiation technologies are used to protect data confidentiality and reliability. Data protection can be divided into data-at-rest and data-in-transit protection.

*Data at rest* is stored data that is not actively changing, such as a local file, a file on the network storage, a file on the backup disk/tape, or a file stored in the cloud. Protecting data at rest is mandatory for most public or private organizations and, in general, is critically important as attackers find increasingly innovative ways to compromise systems and steal data. While data at rest is generally considered to be less vulnerable than data in transit, attackers often find data at rest a more valuable target.

To protect data by encryption, three options are available: file, disk, and tape encryption.

Disk encryption is slower but protects all the data on disk. File encryption is more flexible and faster but does not provide 100% coverage (for example, the attacker can see info such as directory structure, filenames, and so on). Tape encryption is a solution to encrypt data backups; encryption is done at the hardware level so it's fast and protects the full volume.

*Data-in-transit* communication security is provided by the Transport Layer Security (TLS) cryptographic protocol. It's often called SSL, which is the name of its predecessor, the Secure Socket Layer protocol (deprecated). TLS guarantees integrity and confidentiality over network communication (more detail can be found in [Section 4.9](#)).

## 4.8 Identify the secret storage approach relevant to a given scenario

Many applications require some user or resource authentication. The most often used (and familiar) is *password* authentication. It's easy to implement and it's cost-effective; however, it has security issues (mainly human factor related): people use short passwords, guessable passwords, the same passwords for different systems, or they never change their passwords. Some of the issues may be alleviated, but as a result, passwords often become too hard to memorize, so people just write them down.

Applications often need credentials to be able to access external systems, resources, and APIs. Since there's no need for human memorization, passwords (also called secrets, passphrases, or keys) can be

- Complex enough to be unguessable
- Changed often
- Different for each resource or environment

When planning protection for the application secrets, consider the following:

- There's no universally correct way to protect credentials because architectures vary.
- If credentials need to be known on more than one system, use a distributed approach.
- If you're using several environments (prod/pre-prod/dev), do not use the same secret.
- Use multifactor authentication (MFA), if feasible.
- Use a single sign-on system when possible to lower the number of authentication transactions and reduce the possibility of password sniffing.

Application secrets need to be known in the code before applications can use them. There are various ways to do that; some are simple and unsecure, and some are secure but more complex to implement.

## Traditional Secrets Storing

The very simplest way to store a secret is to define it within the code, as shown here:

[Click here to view code image](#)

```
ACCESS_USER="admin"
ACCESS_SECRET = "my_own_password"
result1= API_call (url, user=ACCESS_USER,
password=ACCESS_SECRET)
result2= API_call (url, user="admin",
password="my_own_password")
```

The disadvantage of this method is that everybody on the development team has access to the value of the secret, like a network firewall password, which they are not supposed to know. It may also lead to accidental secret exposure, if the source code is shared or published on a public repository, like GitHub.

The better way is to store a secret in an environment variable, separating the value from code, as shown next:

[Click here to view code image](#)

```
ACCESS_USER=os.environ["ENV_ACCESS_USER"]
ACCESS_SECRET=os.environ["ENV_ACCESS_SECRET"]
result= API_call (url, user=ACCESS_USER, password=ACCESS_SECRET)
```

Various secrets can be stored in a single centralized password database, as shown below, where access is protected with its own set of credentials. The main advantage is that now passwords may be managed independently from the code (for example, implement regular password rotation). A disadvantage is that the password problem is not resolved; instead, it's just moved to a different location. Plus, if the database password is compromised, an attacker would have access to all the secrets at once.

[Click here to view code image](#)

```
DEVICE_IP="10.1.1.250"
query=f"SELECT username, password FROM secrets WHERE
deviceIP='{DEVICE_IP}'"

db = mysql.connect(host=db_host, user="dba", passwd="dbpass",
database="my_db")
cursor = db.cursor()
cursor.execute(query)
ACCESS_USER=cursor["username"]
ACCESS_SECRET=cursor["password"]

result= API_call (url, user=ACCESS_USER, password=ACCESS_SECRET)
```

A variation of this approach, with the same drawbacks, is using a cloud API syncing service, where application code would make an authenticated API call and retrieve device credentials to be used. The benefits of this option are an encrypted API connection and ready-to-use password encryption functions.

Password encryption can be used with any previously mentioned method. It may be a simple obfuscation like Base64 encoding, but be aware that despite “U29tZVZlcnIMb25nUGFzc3dvcmQ=” looking way more secure than “SomeVeryLongPassword” string, it may be reversed quite easily.

More advanced security may be implemented, where the decryption key is required to reverse encrypted credentials, but this gets us back to the original issue of securing access to the authentication credential; this time, of the decryption key.

*Secret managers* provide centralized secure storage for passwords, tokens, certificates, encryption keys, and other sensitive data. They provide access to secrets via an API, CLI, or GUI in a tightly controlled manner. Additional security features may be implemented, such as role-based permissions, time-based permissions, password rotation, temporary access, and so on.

An example of a secret manager is HashiCorp Vault.

*Cloud-based password managers* use dedicated cloud services, where the privilege to access the secret storage service is granted to VMs and services. This way, developers can work on their code but don’t have access to the authentication service. This service is easy to use and to integrate with VMs and services in the same cloud environment.

An example of such a cloud service is Identity and Access Management (IAM) for AWS.

Another recent method is to allow the application instances to request access to resources through API gateways that act as an authentication proxy. An API gateway authenticates requests based on identity, makes calls to back-end servers with proper credentials, and sends resources back to the requesting application upon receiving a response.

The main advantage is that secrets are never stored or sent to the application, so it is impossible to recover secrets for developers or anybody possessing the source code. A limitation is that it can be complex to implement, as advanced cloud knowledge is required.

## 4.9 Configure application-specific SSL certificates

When applications allow remote users to connect to them over unencrypted sessions, they expose users to certain security risks. An attacker could record and monitor their traffic and obtain any information the user supplies, like user credentials in Telnet or credit card numbers in a banking app. Furthermore, an attacker may modify traffic and use the application as a platform for further attacks.

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL, now deprecated), are cryptographic protocols designed to provide end-to-end secure network communications. Websites can use TLS to secure all communications between their servers and web browsers.

The TLS protocol provides *privacy* and *data integrity* between communicating applications. When secured by TLS,

connections between a client and a server have one or more of the following properties:

- The connection is secure because of the symmetric cryptography used to encrypt the transmitted data. Keys for this encryption are generated uniquely for each connection, and their negotiation is both secure (the negotiated secret is unavailable to eavesdroppers) and reliable (no attacker can modify the communications during the negotiation without being detected).
- The identity of the communicating parties is authenticated using *public-key cryptography*. This authentication is required for at least one of the parties (typically the server).
- The connection is reliable because each transmitted message includes an integrity check to prevent undetected data loss or alteration during transmission.

Public key cryptography is a cryptographic technique that enables entities to *securely* communicate on an *insecure public network* and reliably verify the identities via digital signatures.

It uses pairs of keys: *public keys* and *private keys*. Effective security only requires keeping the private key private and known only to the owner; the public key can be openly distributed without compromising security. This system has two uses:

- **Asymmetric encryption:** Anybody can encrypt a message using the receiver's public key, and it can only be decrypted by the receiver with its private key
- **Authentication:** The private key owner can create a digital signature of the message by combining the message with the private key when sending it. Anyone

with the sender's public key can verify signature validity, thus ensuring that the message was sent by the legitimate owner of the private key.

A very important issue with this system is confidence/proof that a particular public key is authentic and has not been tampered with or replaced by a malicious third party. One widely used method is a public key infrastructure (PKI), in which one or more third parties, known as certificate authorities (CAs), certify ownership of key pairs.

PKI is a set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption. PKI creates digital certificates that map public keys to entities, securely stores these certificates in a central repository, and revokes them if needed.

PKI is an example of a trusted third-party system. With this method, the client trusts the server's identity (in a form of a digital certificate) when its certificate is signed by the trusted CA.

The basis of this trust is the CA's public key, which is contained in the CA's own identity (also called *root*) certificate. It is self-signed and, as such, its identity must be confirmed explicitly so all the systems that use PKI must have the CA's public key installed and marked as trusted. Often this process is automated; for example, all popular web browsers include a large set of preinstalled public root CA certificates.

To obtain and activate its identity certificate, the system's (website) owner needs to *enroll* with the CA first. Here's the process:

- The resource's RSA key pair (public and private) is generated.

- A *certificate signing request* (CSR) containing identity information and the public key is delivered to a CA. Identity details can include data such as the system name, the organization to which the system belongs, and location information.
- The CA validates the request.
- The CA generates the system's certificate, which contains these (or more) fields, as per the X.509 standard:
  - *Serial Number*, which is used to track revocation status with the CA.
  - *Subject* with the system's identity details (from CSR).
  - *Public Key* of certificate's subject (from CSR).
  - *Key Usage* defines valid cryptographic uses; typically it's *Digital Signature*.
  - *Not Before* and *Not After* define a time range for certificate validity.
  - *Issuer* contains the CA's subject.
  - *Signature Algorithm* is used to sign the public key certificate.
  - *Signature* of the certificate body by the CA's private key.
- The certificate is delivered back and installed on a system.

CSRs and certificates are typically distributed in the PEM format. A *PEM file* is a text file containing one or more items in Base64 ASCII encoding that may be easily copied/pasted, each with plaintext headers and footers (for example, ----- BEGIN CERTIFICATE----- and -----END CERTIFICATE-----). An

alternative is DER, which is a binary format for certificates and private keys.

As an example, here is content of [github.com](https://github.com) certificate with its fields:

[Click here to view code image](#)

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      05:57:c8:0b:28:26:83:a1:7b:0a:11:44:93:29:6b:79
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = US, O = DigiCert Inc, OU = www.digicert.com,
CN = DigiCert SHA2 High Assurance Server CA
    Validity
      Not Before: May  5 00:00:00 2020 GMT
      Not After : May 10 12:00:00 2022 GMT
    Subject: C = US, ST = California, L = San Francisco, O =
"GitHub, Inc.", CN = github.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
        RSA Public-Key: (2048 bit)
          Modulus:
            00:bb:32:b4:d0:d8:9e:9a:9e:8c:79:29:4c:2b:a8:
              [..... removed for brevity .....,]
              18:b5
            Exponent: 65537 (0x10001)
      X509v3 extensions:
        X509v3 Authority Key Identifier:
          keyid:51:68:FF:90:AF:02:07:75:3C:CC:D9:65:64:62:A2:12:B8:59:72:3
          B
        X509v3 Subject Key Identifier:
          63:02:D2:5D:02:5F:F7:8D:D5:5A:12:9E:76:11:36:96:86:2C:8A:48
        X509v3 Subject Alternative Name:
          DNS:github.com, DNS:www.github.com
        X509v3 Key Usage: critical
          Digital Signature, Key Encipherment
        X509v3 Extended Key Usage:
          TLS Web Server Authentication, TLS Web Client
          Authentication
        X509v3 CRL Distribution Points:
```

```
Full Name:  
URI:http://crl3.digicert.com/sha2-ha-server-  
g6.crl  
  
Full Name:  
URI:http://crl4.digicert.com/sha2-ha-server-  
g6.crl  
  
X509v3 Certificate Policies:  
Policy: 2.16.840.1.114412.1.1  
CPS: https://www.digicert.com/CPS  
Policy: 2.23.140.1.2.2  
  
Authority Information Access:  
OCSP - URI:http://ocsp.digicert.com  
CA Issuers -  
URI:http://cacerts.digicert.com/DigiCertSHA2HighAssuranceServerC  
A.crt  
  
X509v3 Basic Constraints: critical  
CA:FALSE  
  
Signature Algorithm: sha256WithRSAEncryption  
86:32:8f:9c:15:b8:af:e8:d1:de:08:3a:44:0e:71:20:24:d6:  
[..... removed for brevity .....]  
27:4a:7a:49
```

If private keys are compromised or business use for a certificate has expired (for example, the person is leaving the company), then certificates may be *revoked*. Certificate revocation is a centralized function, and the two most used methods are certificate revocation lists and the Online Certificate Status Protocol:

- *Certificate revocation lists (CRLs)* are generated and published periodically, often at a defined interval. To prevent attacks, CRLs are usually digitally signed by the corresponding CA. All CRLs have a lifetime during which they are valid; this timeframe is often 24 hours or less.

Since CRLs are generated periodically and need to be downloaded, they are not always up to date, and this allows some window of opportunity to an attacker.

- *Online Certificate Status Protocol (OCSP)* is an alternative to CRLs. It allows for real-time certificate validation checks. An OCSP response contains less data than a CRL, so it puts less load on network and client resources. Messages communicated via OCSP are encoded in ASN.1 and are usually communicated over HTTP.

URIs for both CRL and OCSP are often embedded into the identity certificate.

PKI can be set up hierarchically. An intermediate CA certificate is a subordinate certificate issued by the root CA specifically to issue end-entity server certificates. The result is a trust-chain that begins at the trusted root CA, through the intermediate, and finally ending with the certificate issued to the end user. Such certificates are called *chained root certificates*.

Intermediate CAs provide added flexibility, reliability, and security. Different subordinate CAs may have different policies. As there's no need to issue certificates directly from the CA root certificate, the CA may be kept offline all the time (with rare exceptions of issuing intermediate CAs) to prevent security accidents. In large networks using CRLs in multiple tiers, the CA helps to control their size.

When using intermediate certificates, the client needs to know the whole certificate chain to verify the validity of an SSL certificate. An effective way to inform the client of the whole chain is using certificate bundles that contain an identity certificate along with all the intermediate certificates.

One very common use of PKI is TLS.

TLS uses PKI to authenticate peers and to facilitate the exchange of session keys that are used to encrypt the session. Many applications use TLS to provide authentication and encryption, with HTTPS being the most widely used. SMTP, LDAP, and POP3 were modified to be transported within TLS to address their lack of proper authentication and encryption.

An encrypted session starts with the TLS handshake, where the following steps are taken:

1. **Client hello:** The client sends a “hello” message to the server. It includes the supported TLS version, supported cipher suites, and a string of random bytes known as the “client random.”
2. **Server hello:** In the reply to the client, the server sends a message with the server’s certificate, the chosen cipher suite, and server-generated “server random” string of bytes.
3. **Authentication:** The client verifies the server’s certificate to confirm that the client is interacting with the actual owner of the domain.
4. **Premaster secret:** The client sends one more random string of bytes, known as the “premaster secret.” It is encrypted with the public key from the certificate and can only be decrypted by the server.
5. **Session keys:** Both the client and server generate session keys from the client random, the server random, and the premaster secret. They should arrive at the same results.
6. **Client is ready:** The client sends a “finished” message that is encrypted with a session key.
7. **Server is ready:** The server sends a “finished” message encrypted with a session key.

**8. Secure symmetric encryption achieved:** The handshake is completed, and communication continues using the session keys.

Note: Asymmetric key algorithms are much more computationally intensive than symmetric ones. It is common to generate a session key using a certificate-based key-exchange algorithm, described earlier, and then transmit encrypted data using the session key.

Certificate validation is not always successful, but it's up to the application how to handle it. Many web browsers will just display a security warning and allow a user to proceed with the connection under potentially hazardous conditions. Here are the most common issues with certificates:

- **Hostname/Identity mismatch:** The name specified in the URL does not match the name specified in the server's identity certificate. A benign example would be using an IP address instead of the hostname. A malign example would be an attacker using a domain name that looks very similar to an authentic name to trick a user into bypassing the browser warning.
- **Validity date range:** Specified within the certificates with the *Not Before* and *Not After* dates. If the current date is outside the range, the certificate is considered invalid. It could be an administrator's oversight (certificate expired) but may also reflect a more serious problem and should not be trusted. There's one possible benign cause for that, thought: incorrect local system time. Therefore, that needs to be checked as well.
- **Signature validation error:** If the certificate signature cannot be validated, there is no assurance that the system is authentic. Often, it's a server that uses a self-signed certificate to avoid the complexity or

expense of using PKI. Their usage puts the responsibility of the certificate verification on a user. Another possible cause of a signature verification error is that the certificate has been tampered with; for example, an attacker replaced the server's real public key with his own public key.

## 4.10 Implement mitigation strategies for OWASP threats (such as XSS, CSRF, and SQL injection)

The Open Web Application Security Project (OWASP) is an online community that produces freely available articles, methodologies, documentation, tools, and technologies in the field of web application security. The *OWASP Top Ten* is a list of the 10 current and most dangerous web application security flaws.

The latest OWASP Top Ten was released in 2017 and covers the following risks:

### - A1:2017—Injection

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. An attacker can trick the interpreter into executing unintended commands or accessing data without proper authorization.

**Prevention:** Use automated testing, keeping data separate from commands and queries, and input validation.

### - A2:2017—Broken Authentication

Application functions related to authentication and session management are often implemented incorrectly,

allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

**Prevention:** Use well-tested, current frameworks, use MFA, and do *not* use defaults.

#### - A3:2017—Sensitive Data Exposure

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII data. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised.

Prevention: Encrypt sensitive data at rest and in transit and use secure password storage.

#### - A4:2017—XML External Entities (XXE)

Many older or poorly configured XML processors evaluate External Entity references within XML documents. External Entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial-of-service attacks.

**Prevention:** Disable processing of any External Entities included in the XML document.

#### - A5:2017—Broken Access Control

Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data or access rights, and so on.

**Prevention:** Except for public resources, deny by default; implement and reuse access control mechanisms and log access control failures.

### - A6:2017—Security Misconfiguration

The most commonly seen issue, which is a result of insecure default, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information.

**Prevention:** Use a repeatable automated hardening process; environments (Dev, QA, Prod) should all be configured identically, but with different credentials.

### - A7:2017—Cross-Site Scripting (XSS)

XSS may occur whenever an application includes untrusted data in a web page without proper validation or escaping, or if it updates an existing web page with user-supplied data. XSS allows attackers to execute scripts in the victim's browser, which can hijack user sessions, deface websites, or redirect the user to malicious sites.

**Prevention:** Separate untrusted data from active browser content (escaping).

### - A8:2017—Insecure Deserialization

- *Deserialization* is taking data structured from some format (for example, JSON or XML) and rebuilding it into an object. Insecure deserialization often leads to remote code execution or can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

**Prevention:** Do not accept serialized objects from untrusted sources, perform integrity checks, use well-

tested libraries that implement safe parsing of those formats.

### **- A9:2017—Vulnerable Components**

Components such as libraries, frameworks, and other software modules run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

**Prevention:** Apply a patch management process (keep track of tools, monitor vulnerabilities, and download from the official sources only).

### **- A10:2017—Insufficient Logging & Monitoring**

Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is more than 200 days, typically detected by external parties rather than internal processes or monitoring.

**- Prevention**—Implement logging with enough context for login, access-control, and validation failures, implement monitoring and alerting, keep audit trail for critical transactions

A more detailed overview can be found at  
<https://owasp.org/www-project-top-ten/>.

*Cross-site scripting (XSS)* is a type of code injection attack. Attackers use known vulnerabilities in web-based applications (websites) and exploit them to insert malicious content into content delivered from those websites. When

the resulting content arrives at the client-side web browser, it has all been delivered from the trusted source and is processed accordingly. As a result, an attacker can gain elevated access privileges to sensitive page content, session cookies, and a variety of other information maintained by the browser on behalf of the user.

There are two types of XSS:

- **Non-persistent (reflected)**: When a user entry is immediately displayed back (for example, a search engine typically includes the query with the results). Typically delivered via email or neutral website as a click-bait URL pointing to a trusted site but including the XSS vector. If the site is vulnerable, the script will be executed in the victim's browser.
- **Persistent (stored)**: When data provided by the attacker is saved by the server and then permanently displayed to other users. A classic example is an online forum that allows HTML-formatted user posts.

To mitigate this attack, the easiest way is to escape any user input, both when receiving it from the user and when sending data to other users. For example, replacing the < character with &lt; or &#60; for the web content would disable all HTML tags. In Python, this can be done with the standard `html.escape()` and `html.unescape()` functions:

[Click here to view code image](#)

```
import html

print (html.escape ("<script type='application/javascript'>"))
&lt;script type=&#x27;application/javascript&#x27;&gt;
```

Sometimes some HTML tags are acceptable, so escaping them all is not an option. In this case, *sanitization* is a suitable solution. Sanitization processes untrusted user

input and runs it through the filter, which allows valid content but strips anything that is not permitted. For example, it may allow `<A>`, `<IMG>`, `<B>` HTML tags but strip any other tags.

*Cross-site request forgery (CSRF)* is an attack that forces the user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick users into executing actions of the attacker's choosing. Successful CSRF attacks can force the user to perform state-changing requests like transferring funds, changing their email address, and so forth. In this example of such an exploit for a weak banking website, a user is tricked into transferring money when clicking the link:

[Click here to view code image](#)

```
<a href="http://bank.com/transfer.do?acct=MARIA&amount=1000">My  
Pictures!</a>
```

The attack would be successful if the user currently has an open authenticated session with the bank site, as the existing session cookie would be used for site authentication.

One of the mitigation techniques is to use the CSRF session token: a unique, unpredictable value generated by the server and transmitted to the client. When the user request is made, the token is included in the request, and the server validates the request, rejecting it if the token is invalid.

CSRF tokens make a CSRF attack impossible because an attacker cannot guess the value of a user's token and construct a valid HTTP request that looks like it was created by an unsuspected user.

For CSRF protection, you can write your own CSRF token code or use one of the existing libraries (for example, the

Flask-WTF extension if you work with Flask).

*SQL injection* is a code injection technique, used to attack data-driven applications, in which the attacker inserts malicious SQL code into an entry field. Executing this code grants the attacker complete control over the application's database and allows attackers to spoof identity, tamper with existing data, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

This form of injection occurs when user input is passed into an SQL statement without proper filtering.

The following line of code illustrates this vulnerability:

[Click here to view code image](#)

```
statement = "SELECT * FROM users WHERE name = '" + userName +  
"';"
```

If `userName` is obtained from user input or a URL query parameter and is not validated, it may be exploited. Here are a few examples:

- If `userName` is set to '`' OR '1'='1`', then the query becomes `SELECT * FROM users WHERE name = '' OR '1'='1'`; which would return the complete list of all users in the system.
- If `userName` is set to `a';DROP TABLE users;`, then the query becomes `SELECT * FROM users WHERE name = 'a';DROP TABLE users;` which would delete all the database users.

The following defense methods are available:

- Input validation/sanitization in the form of blacklisting (where certain characters are not accepted) or whitelisting (where only certain characters are approved). Whitelisting is more effective, as complex

obfuscation technologies may allow attackers to bypass blacklisting.

- Parameterized queries and prepared statements. SQL statements are built without user-supplied data and then augmented with the data in such a manner that the structure and intent of the SQL statement cannot be altered.
- Applications should catch and prevent all SQL-generated error messages from being displayed to the end user because SQL errors greatly aid attackers in successfully exploiting SQL injection vulnerabilities. Error messages should be logged to the back end instead.
- Using the “least privilege” principle, users should only have permissions to things they need; for example, an account that is used to read data from the database should not have permissions to drop tables.
- Use the data validation process to assess data against a set of rules (schema) to make sure that the data is structured in the way you expect. For example, an “SKU” field must be an integer between 1000 and 9999. Cerberus is one of the popular libraries for data validation in Python.
- On a network level, an intrusion prevention system (IPS) or a next-generation firewall (NGFW) can detect and prevent SQL injections attacks.

## 4.11 Describe how end-to-end encryption principles apply to APIs

End-to-end encryption (E2EE) is a method of secure communication where only the communicating end users can access the data, while any third-party is prevented from

accessing it while it's transferred from one end system to another.

In E2EE, the data is encrypted on the sender's system or device and only the recipient is able to decrypt it. Nobody in between, be they an Internet service provider, application service provider, or hacker, can read it or tamper with it because encryption keys are known only to the communicating parties.

A direct TLS session between two servers is an example of E2EE, whereas for two users messaging each other in Facebook chat, E2EE is not the case, as Facebook servers decrypt, process, and re-encrypt all the messages, so Facebook has full access to the conversation.

Confidentiality is a regulatory or industry standard requirement for many businesses these days, and that includes network traffic. There are various traffic flow patterns: users to the Internet, on-premises to the cloud, traffic within the data center, and so on, and each of them requires secure communication.

API traffic requires secure treatment as well because it may carry access credentials or sensitive data. Unsecured API calls may result in security breaches that may have serious business consequences, such as losing the right to operate due to lack of compliance, reputation loss due to security breaches, or escalating cloud costs dues to malicious use of resources billable in a pay-as-you-go model.

There are generally two main types of traffic, and API calls should be protected for both of them:

- **North-south:** An internal user to an external resource, such as user to Internet
- **East-west:** Local traffic, such as communication between cluster members at a data center

Because of the multitier design of today's enterprise applications, the east-west direction typically accounts for 70% of all network traffic in the data center. With the adoption of a microservices architecture and container technology, the portion of east-west traffic will only grow larger. As monolithic applications are broken into pieces, local function calls are replaced by packets on the wire.

Traditionally, north-south was given much more attention, with perimeter security devices deployed at the data centers. Inter-data-center traffic is often further secured by IPSec.

At the same time, east-west traffic security within a data center is often neglected. In fact, according to some research reports, about half of the security breaches are caused by insiders within a data center. A compromised device can introduce malware that can spread rapidly without constraints, or sensitive data may be stolen, and so on. This means that protecting east-west communications is essential as well.

To protect north-south API traffic, the following options are available:

- *Traditional TLS or Mutual TLS (MTLS) authentication and encryption*

Traditional TLS is a popular and easy-to-implement option for API call protection. One downside of it is that it provides one-way authentication only (the client authenticates the server but the server has no information about who the client is). This is not always desirable for API calls when access to the critical information needs to be restricted only to certain accounts.

With Mutual TLS, both the server and a client must present a trusted certificate to authenticate the session.

- *TLS or IPSec from the client to an API gateway or NGFW*

An API gateway is a reverse proxy in front of API endpoints that allows access but avoids exposing them directly to the outside world. Typical features of an API gateway include the ability to authenticate requests, enforce security policies, inject additional details, load-balance between back-end services, and throttle them if necessary.

The main advantage of using an API gateway is reduced complexity in client and server code, a more manageable way to enforce access and impose limits, and a reduction in the overall network latency in some cases.

- *Cloud-based VPN service*

In this case, a shared IPSec VPN is established between an on-premises router and a VPN service or VPN gateway at the cloud provider. This VPN is used by all clients. This method does not provide full EE2E, as the traffic between a client and the local router is not encrypted, so it is recommended to use TLS between the endpoints where possible.

- *Dedicated cloud connections*

With this method, the connection to the cloud is a dedicated private circuit instead of a path over the Internet. Despite being called “private,” third parties (service providers) still have visibility into the data sent over this circuit, so it’s best to protect this path with IPSec and/or TLS, as in the previous case.

To protect the east-west API traffic, the following options are available:

- *TLS or MTLS between services, like for north-south traffic*
- *Service mesh segmentation service*

Microservices-based architecture creates a new challenge in service-to-service communication. Service mesh is an infrastructure layer that manages communication by abstracting application networking and providing a set of L7 controls over traffic routing, policy enforcement, and strong identity (authentication and authorization) and security (encryption and mTLS).

With service mesh, all communication between the application services are facilitated through the proxies rather than making direct container-to-container connections. Popular service mesh technologies include Istio, Linkerd, and AWS App Mesh.

- *Dedicated container firewall*

Due to the dynamic nature of containers, it's challenging to protect them on a network level. One way to address this is to integrate a firewall within a container. They usually miss NGFW functionality like L7 application awareness, IPS features, identity-based firewalling, but they can deliver whitelisting, connection inspection and encryption, and more at the container level.

## 4.12 Chapter 4 Review Questions

**Q1: Which statement is true about CI/CD pipeline failures?**

- A) Job failure in a CI/CD stage means that the whole pipeline fails, and it needs to be restarted from scratch after the problems are fixed.
- B) Test failure should be addressed by the source code review and correction.

- C) Missing dependencies and wrong library versions result in the continuous delivery phase failure.
- D) Running a complete CI/CD pipeline is the most efficient way to test all the source code changes.

#### Q2: Which statement is true about Kubernetes?

- A) Applications should be converted to the Kubernetes format before their deployment.
- B) Kubernetes orchestrates the build and deployment of applications.
- C) Kubernetes provides storage and cache services to applications.
- D) Kubernetes ensures the desired state for the deployed application by creating, removing, and restarting containers as needed.

#### Q3: Which is *not* a Kubernetes element?

- A) Kubernetes pod
- B) Kubernetes cluster
- C) Kubernetes container engine
- D) Kubernetes worker node

#### Q4: Which statement is true about Kubernetes objects?

- A) Kubernetes objects status describes the desired state of the cluster.
- B) Kubernetes Deployment object defines the application and its network parameters.
- C) When Kubernetes detects a mismatch between the specs and the status of the object, it responds by automatically making the required correction.
- D) At the end of the CI/CD pipeline that updates the application to a newer version, Kubernetes objects are

also updated, which requires a simultaneous restart of all the application instances on all nodes.

**Q5: Which statement is true about continuous testing in a CI pipeline?**

- A) Continuous testing is a recurring process that restarts as soon as the previous iteration is complete.
- B) It reduces business risks.
- C) It improves quality but slows down the overall software delivery.
- D) It's a slow process, but it is done in parallel with the software development, so it does not have much of an impact on the delivery timelines.

**Q6: Which statement is *not* true about static code analysis in a CI pipeline?**

- A) It runs a set of test cases against the application code elements.
- B) It can detect security vulnerabilities in the application code.
- C) It can detect bugs in the application code.
- D) It can analyze source syntax and enforce coding standards.

**Q7: Which statement is true about Docker containers?**

- A) Containers share process table and network interfaces with the operating system of the host.
- B) The main components of the Docker Engine are the server, client, and REST API.
- C) The OS kernel may be included in a container to allow it to run on a host with a different kernel.
- D) Microservices applications may be easily and effectively managed from the CLI with the “docker” client.

**Q8:** Which statement is true about building custom Docker images?

- A) Docker images are monolithic.
- B) The “FROM” directive may be omitted from the Dockerfile, and “FROM scratch” is assumed then.
- C) The “RUN” directive executes a command in a new layer on top of the current image, with the resulting image used for the next steps.
- D) The “EXPOSE” Dockerfile directive defines the container’s ports that will be open for external connectivity when the container is executed.

**Q9:** A container was created and executed as shown next. What would be the result?

[Click here to view code image](#)

```
$ cat Dockerfile
FROM python
ENTRYPOINT ["echo","Blue"]
CMD ["Green"]
$ docker build -t colors . >/dev/null
$ docker run -it colors "Red"
```

- A) The latest Python interpreter is started.
- B) “Blue” is printed.
- C) “Blue Green” is printed.
- D) “Blue Red” is printed.

**Q10:** Which statement is true about Docker storage and networking?

- A) By default, container storage is available (read-only) to the other processes on the host machine.
- B) Both non-Docker processes and a Docker container can modify Docker volumes.

- C) The host network driver that uses the host's networking directly is a default driver.
- D) On a user-defined bridge network, containers can resolve each other by name or alias.

**Q11: Which principle is true about applications built according to the "12-factor app" methodology (1)?**

- A) Each deploy (DEV, UAT, and PROD) has its own codebase.
- B) All dependencies (tools and libraries) should be explicitly declared.
- C) Per-deployment variables and credentials may be stored in the environment variables or in the source code.
- D) To be able to use MySQL in the Dev environment and Amazon RDS in production, the database selection logic should be updated within the application.

**Q12: Which principle is true about applications built according to the "12-factor app" methodology (2)?**

- A) Minor bugfixes may be implemented directly in the production environment.
- B) An application may rely on the external web service to process HTTP requests.
- C) A sudden ungraceful shutdown of a process within an app may cause application instability.
- D) The application does not manage log files. It just writes an unbuffered log stream to stdout.

**Q13: Which statement is true about the effective logging strategy for an application?**

- A) Distributed logging always includes log collection, forwarding, and indexing.

- B) Each event stream is typically stored in its own separate space.
- C) Using the same correlation ID in all related calls and logging it greatly simplifies troubleshooting.
- D) Logs should be stored as-is.

**Q14: Which statement is true about data privacy?**

- A) Visiting a website implicitly permits site owners to collect user information.
- B) Pseudonymization irreversibly destroys any way of identifying the data subject.
- C) Anonymization irreversibly destroys any way of identifying the data subject.
- D) Data in transit is more secure because it is encrypted, whereas data at rest is not.

**Q15: Which statement is true about safely storing resource credentials and other secrets?**

- A) Storing credentials directly in the application code is acceptable.
- B) Storing credentials directly in the application code is acceptable as soon as they are obfuscated.
- C) Storing credentials in the environment variables is acceptable.
- D) Using API gateways is an easy and efficient way to provide secured access.

**Q16: Which is the correct sequence for the website's identity certificate activation?**

- A) The CA generates the system's certificate.
- B) A certificate signing request (CSR) with the identity information and the public key is delivered to a CA.
- C) The certificate is delivered and installed on a system.

- D) The CA validates the request.
- E) The RSA key pair (public and private) is generated.

Q17: Which statement is true about PKI (public key infrastructure)?

- A) Remote identity is trusted as soon as the presented certificate is signed by a CA (certificate authority).
- B) If the certificate gets compromised, it may be immediately revoked via the CRL (Certificate Revocation List) process.
- C) Certificate keys are used to encrypt data in the TLS-protected session.
- D) TLS provides encryption not only to HTTP but to other upper-level protocols as well.

Q18: Which statement is true about OWASP threats?

- A) Cross-site scripting (XSS) protection requires all HTML tags to be escaped/removed.
- B) Cross-site request forgery (CSRF) attacks may be mitigated by using session tokens.
- C) An SQL injection attack is a unique type of attack because it exploits specific database system vulnerabilities.
- D) Input blacklisting is the most efficient protection from SQL injection attacks.

Q19: Which statement is true about end-to-end encryption (E2EE)?

- A) Users communicating with each over a secure connection to an intermediate server is one example of E2EE.
- B) API traffic typically carries non-business-critical data, so it's preferred but not required to have it encrypted.

- C) API traffic within a data center is secure and does not require encryption.
- D) Mutual TLS is more preferred for API calls over regular TLS.

---

## *5. Infrastructure and Automation*

---

### **5.1 Explain considerations of model-driven telemetry (including data consumption and data storage)**

Traditional monitoring with the periodic fetching of data (for example, SNMP) is not an adequate solution for applications requiring frequent or prompt updates of remote object state. Polling-based solutions impose a load on networks, devices, and applications.

Telemetry is an alternative automated process by which measurements and other data are collected at remote points and transmitted to the receivers for monitoring. Model-driven telemetry (MDT) provides a mechanism to stream *YANG-modeled* data to a data collector.

The following roles are defined when using telemetry:

- **Subscriber:** Makes requests for a set (or sets) of YANG object data and creates subscriptions.
- **Publisher:** Network element that sends the telemetry data.
- **Receiver (or collector):** Receives the telemetry data. Typically, the receiver and subscriber will be the same entity.

A *subscription* is a contract between a publisher and a subscriber that stipulates the data to be pushed and the associated terms (for example, is data to be published

continuously or upon change? how the data is to be formatted?).

There are two types of subscriptions used in telemetry on Cisco IOS XE systems: dynamic and configured subscriptions. In *dynamic* subscriptions, clients subscribe to specific data items they need, by using standard-based YANG data models over the NETCONF, RESTCONF, or gRPC Network Management Interface (gNMI) protocol. *Configured* (or *static*) subscriptions are created by using CLIs.

NETCONF, gRPC, and gNMI transport protocols are supported (as of IOS XE 17.3.3) with the XML, protobuf (Google Protocol Buffers), and JSON\_IETF encodings. Refer to a current Cisco IOS XE Programmability Configuration Guide to confirm compatible combinations.

The two flavors of model-driven telemetry are *dial-in* and *dial-out*, as detailed in the following table.

<b>Dial-In</b>	<b>Dial-Out</b>
Dynamic subscription.	Configured subscription.
Telemetry updates are sent to the initiator/subscriber.	Telemetry updates are sent to the configured receiver/collector

<b>Dial-In</b>	<b>Dial-Out</b>
The life of the subscription is tied to the connection/session that created it. No change in the running configuration is observed.	Subscription is created as part of the running configuration; it remains the device configuration until the configuration is removed.
Dial-in subscriptions need to be re-initiated after a reload because established connections or sessions are dropped.	Dial-out subscriptions automatically reconnect to the receiver after a reload because they are a part of the device configuration.
Subscription ID is dynamically generated upon successful establishment of a subscription.	Subscription ID is fixed and configured on the device as part of the configuration.
NETCONF, gNMI are used as transport.	gRPC is used as transport.

The amount of data that is collected may be very extensive in large networks to the point it may impact overall network performance; therefore, it is important to apply proper filters (as specific as possible) and review the frequency of periodic updates.

Here's a configuration example:

[Click here to view code image](#)

```
MDT#show running-config | sec tele
telemetry ietf subscription 100
  encoding encode-kvvpb
  filter xpath /process-cpu-ios-xe-oper:cpu-usage/cpu-
utilization/five-seconds
  source-address 192.168.2.100
  source-vrf Mgmt-vrf
  stream yang-push
  update-policy periodic 500
  receiver ip address 172.16.10.10 57000 protocol grpc-tcp
```

Full implementation typically includes a receiver to consume data, a database to store data, and a visualization/alerting tool. One example would be a combination of Telegraf, InfluxDB, and Grafana, and another would be Kafka + ELK stack.

## 5.2 Utilize RESTCONF to configure a network device, including interfaces, static routes, and VLANs (IOS XE only)

*RESTCONF* is another protocol developed to manage network devices in a standard way. It is similar to NETCONF but uses HTTP/HTTPS as a transport (instead of SSH over port 830) and uses HTTP verbs (GET/POST/PUT/PATCH/DELETE) to define the desired operation. RESTCONF also supports both JSON and XML messages, versus just XML for the NETCONF.

Protocols seem to look very similar, but the more important difference is that RESTCONF is missing several crucial NETCONF components, like multiple data stores, commits or rollbacks, and configuration locking.

Data structures exchanged in NETCONF/RESTCONF messages are described by the *YANG data modeling language* (similarly to how SMIv2 describes SNMP MIBs). YANG does not define the actual data but rather the data element types and allowed values, their structures, and the relationships between them.

YANG models the hierarchical organization of data as a tree in which each node has a name and either a value or a set of child nodes. YANG provides clear and concise descriptions of the nodes as well as the interaction between those nodes.

YANG structures data models into modules and submodules. A module can *import* definitions from other external modules and can *include* definitions from submodules. The hierarchy can be *augmented*, allowing one module to add data nodes to the hierarchy defined in another module.

YANG data models can describe constraints to be enforced on the data, restricting the presence or value of nodes based on the presence or value of other nodes in the hierarchy.

When building the YANG tree, you use a module as the base unit. It contains a collection of related definitions and consists of three types of statements: module header statements, “revision” statements, and definition statements.

YANG defines four main types of data nodes for data modeling used in the definition statements:

- **Leaf nodes:** Contain simple data like an integer or a string. A leaf node has exactly one value of a particular type and no child nodes.

Here's a YANG example:

[Click here to view code image](#)

```
leaf host-name {  
    type string;  
    description  
        "Hostname for this system.";  
}
```

And here's an instance example (XML encoded):

[Click here to view code image](#)

```
<host-name>my.example.com</host-name>
```

- **Leaf-list nodes:** Define a sequence (list) of values *of a particular type*.

Here's a YANG example:

[Click here to view code image](#)

```
leaf-list domain-search {  
    type string;  
    description  
        "List of domain names to search.";  
}
```

And here's an instance example:

[Click here to view code image](#)

```
<domain-search>ns1.example.com</domain-search>  
<domain-search>ns2.example.com</domain-search>  
<domain-search>ns3.example.com</domain-search>
```

- **Container nodes:** Used to group related nodes in a subtree. A container has only child nodes and no value. A container may contain any number of child nodes of any type (leafs, lists, containers, leaf-lists, actions, and notifications).

Here's a YANG example:

[Click here to view code image](#)

```
container system {  
    container login {
```

```
    leaf message {
        type string;
        description
            "Message given at start of login session.";
    }
}
```

And here's an instance example:

[Click here to view code image](#)

```
<system>
<login>
    <message>Good morning</message>
</login>
</system>
```

- **List nodes:** Define a sequence of list entries. If a list node has any *key leafs* defined, their values must be unique. A list can define multiple key leafs and may contain any number of child nodes of *any type* (including leafs, lists, containers, and so on).

Here's a YANG example:

[Click here to view code image](#)

```
list user {
    key "name";
    leaf name {
        type string;
    }
    leaf full-name {
        type string;
    }
    leaf class {
        type string;
    }
}
```

And here's an instance example (note that "name" is a key, so its value must be unique):

[Click here to view code image](#)

```

<user>
  <name>glocks</name>
  <full-name>Goldie Locks</full-name>
  <class>intruder</class>
</user>
<user>
  <name>snowey</name>
  <full-name>Snow White</full-name>
  <class>free-loader</class>
</user>
<user>
  <name>rzell</name>
  <full-name>Rapun Zell</full-name>
  <class>tower</class>
</user>

```

Similar to programming languages, YANG uses *types* for the data, and they are grouped as follows:

- Built-in data types. Here's an example:

```
leaf name { type string; }
```

- Common data types `ietf-yang-types` and `ietf-inet-types` (defined in RFC 6021). Here's an example:

[Click here to view code image](#)

```
import ietf-yang-types { prefix yang; }
leaf birthday { type yang:date-and-time; }
```

- Derived data types. YANG can define these from base types using the `typedef` statement. A base type can be either a built-in type or a derived type, allowing for a hierarchy of derived types. Here's an example:

[Click here to view code image](#)

```

typedef percent {
  type uint8 {
    range "0 .. 100";
  }
}

leaf completed {
```

```
    type percent;  
}
```

Let's put this theory into practice with the IOS XE RESTCONF implementation.

First, it needs to be enabled on a device:

[Click here to view code image](#)

```
CAT9K(config)#aaa new-model  
CAT9K(config)#aaa authentication login default local  
CAT9K(config)#aaa authorization exec default local  
CAT9K(config)#username admin privilege 15 secret cisco  
CAT9K(config)#ip http secure-server  
CAT9K(config)#ip http authentication local  
CAT9K(config)#restconf  
  
CAT9K#show platform software yang-management process  
confd          : Running  
nesd           : Running  
syncfd         : Running  
ncsshd         : Not Running  
dmiauthd       : Running  
nginx          : Running  
ndbmand        : Running  
pubd           : Running  
gnmib          : Not Running
```

Cisco IOS XE devices support native, IETF, OpenConfig, and tail-f YANG models. Cisco's vendor-specific native model will be used in the following examples to work with the interface, VLAN, and static routing configuration. Resource URIs used in examples are long and may seem confusing. For an overview of the process of figuring them out, refer to [Appendix A: RESTCONF URI Demystified \(IOS XE\)](#).

The following basic code will be used to perform RESTCONF calls. Functions provide wrappers for five RESTCONF operations, printing the resulting HTTP code and, in case of HTTP error, decoding and printing the error cause. RESTCONF URIs for IOS XE always start at `/restconf/data`, so this will be included in the base URL.

In the following code examples in this section, executable code is shown as a bold text and corresponding screen output is added as a blue text following each RESTCONF interaction:

[Click here to view code image](#)

```
import requests
import json

requests.packages.urllib3.disable_warnings()

host="192.168.2.4"
port="443"
username="admin"
password="cisco"

base = f"https://{{host}}:{{port}}/restconf/data"

auth = requests.auth.HTTPBasicAuth(username, password)
headers = {
    'Content-Type': 'application/yang-data+json',
    'Accept': 'application/yang-data+json',
}

def restconf_read (url):
    r = requests.request("GET", url, auth=auth, headers=headers,
verify=False)
    print(f"GET result code: {{r.status_code}}")
    if r.status_code >= 400:
        print(f">>Error: {{r.json()['errors']['error'][0]['error-
message']}}")
    else:
        print(r.text)

def restconf_write (url, method, write_data):
    r = requests.request(method, url,
                         auth=auth, headers=headers, data=json.dumps(write_data),
verify=False)
    print(f"Write {{method}} result code: {{r.status_code}}")
    if r.status_code >= 400:
        print(f">>Error: {{r.json()['errors']['error'][0]['error-
message']}}")

def restconf_delete (url):
    r =
```

```
    requests.request("DELETE",url,auth=auth,headers=headers,verify=False)
        print(f"DELETE result code: {r.status_code}")
        if r.status_code >= 400:
            print(f">>Error: {r.json()['errors']['error'][0]['error-message']}")
```

## Interface Configuration

The starting URI for all interface configuration data is <https://host/restconf/data/Cisco-IOS-XE-native:native/interface/>, and the URI to work with Vlan10 SVI specifically is <https://host/restconf/data/Cisco-IOS-XE-native:native/interface/Vlan=10/> (refer to [Appendix A](#) for more details).

The following code demonstrates how to create, read, update, rewrite, and delete the routed SVI interface configuration using RESTCONF:

[Click here to view code image](#)

```
# predefine URLs to avoid repetition

# parent leaf for all interfaces configuration
url_interfaces = f"{base}/Cisco-IOS-XE-native:native/interface/"

# specific leaf for the Vlan10 configuration
url_vlan10 = f"{base}/Cisco-IOS-XE-
native:native/interface/Vlan=10/"

# specific leaf for the Vlan10 IP address configuration
url_vlan10_ip = f"{base}/Cisco-IOS-XE-
native:native/interface/Vlan=10/" \
                f"ip/address/primary/"

# Vlan10 definition
vlan10_data={

    "Vlan": {
        "name": "10",
        "description": "DevNet Vlan SVI interface",
        "ip": {
            "address": {
                "primary": {
                    "address": "10.10.0.1",
                    "mask": "255.255.255.0"
                }
            }
        }
    }
}
```

```

        "mask": "255.255.255.0"
    }
}
}

# new IPs for the interface
vlan10_ip_1 = {
    "primary" : {
        "address": "10.10.0.10",
        "mask": "255.255.255.0"
    }
}

vlan10_ip_2 = {
    "primary" : {
        "address": "10.10.0.20",
        "mask": "255.255.255.0"
    }
}

# first, DELETE Vlan10 interface for a fresh start. Errors may
be ignored
restconf_delete (url_vlan10)
DELETE result code: 204

# Create and configure the interface in a single call with POST
call
restconf_write (url_interfaces, "POST", vlan10_data)
Write (POST) result code: 201

# POST is not an idempotent operation! Repeat it and it will
fail
restconf_write (url_interfaces, "POST", vlan10_data)
Write (POST) result code: 409
>>Error: object already exists:
/ios:native/ios:interface/ios:Vlan[ios:name='10']

# Read interface configuration to confirm it was successfully
created
restconf_read (url_vlan10)
GET result code: 200
{
    "Cisco-IOS-XE-native:Vlan": {
        "name": 10,
        "description": "DevNet Vlan SVI interface",
        "ip": {

```

```

    "address": {
      "primary": {
        "address": "10.10.0.1",
        "mask": "255.255.255.0"
      }
    }
}

# Now change the IP address on the interface with the PATCH call
restconf_write (url_vlan10_ip, "PATCH", vlan10_ip_1)
Write (PATCH) result code: 204

# Note the changed IP
restconf_read (url_vlan10_ip)
GET result code: 200
{
  "Cisco-IOS-XE-native:primary": {
    "address": "10.10.0.10",
    "mask": "255.255.255.0"
  }
}

# Change the IP address on the interface with the PUT call.
restconf_write (url_vlan10_ip, "PUT", vlan10_ip_2)
Write (PUT) result code: 204

restconf_read (url_vlan10_ip)
GET result code: 200
{
  "Cisco-IOS-XE-native:primary": {
    "address": "10.10.0.20",
    "mask": "255.255.255.0"
  }
}

```

Let's review another example. This time it is a regular switched L2 interface:

[Click here to view code image](#)

```

interface GigabitEthernet1/0/2
  switchport access vlan 10
  switchport mode access
  switchport nonegotiate

```

```
channel-group 10 mode active
lacp rate fast
```

In this example, you can see how the “Cisco-IOS-XE-native” YANG module is augmented with the “Cisco-IOS-XE-switch” and “Cisco-IOS-XE-ethernet” modules (note that the “/” in the interface name needs to be encoded as %2F to avoid HTTP errors):

[Click here to view code image](#)

```
url_interface = f"{base}/Cisco-IOS-XE-native:native/interface/"
\
        f"GigabitEthernet=1%2F0%2F2"
restconf_read(url_interface)

GET result code: 200
{
    "Cisco-IOS-XE-native:GigabitEthernet": {
        "name": "1/0/2",
        "switchport": {
            "Cisco-IOS-XE-switch:access": {
                "vlan": {
                    "vlan": 10
                }
            },
            "Cisco-IOS-XE-switch:mode": {
                "access": {}
            },
            "Cisco-IOS-XE-switch:nonegotiate": [null]
        },
        "Cisco-IOS-XE-ethernet:channel-group": {
            "number": 10,
            "mode": "active"
        },
        "Cisco-IOS-XE-ethernet:lacp": {
            "rate": "fast"
        }
    }
}
```

## VLAN Configuration

The starting URI for all VLAN definition data is  
<https://host/restconf/data/Cisco-IOS-XE-native:native/vlan/>.

The following code demonstrates how to create, read, update, rewrite, and delete a switched VLAN on a switch using RESTCONF:

[Click here to view code image](#)

```
# redefine URLs to avoid repetition

# parent leaf for all VLAN configuration
url_vlan=f"{base}/Cisco-IOS-XE-native:vlan/"

# specific leaf for the Vlan10 name configuration
url_vlan10_name=f"{base}/Cisco-IOS-XE-native:vlan/vlan-
list=10/name/"

# specific leaf for the Vlan11 configuration
url_vlan11=f"{base}/Cisco-IOS-XE-native:vlan/vlan-
list=11/"

# initial set of VLANs
base_vlans = {
    "Cisco-IOS-XE-native:vlan": {
        "Cisco-IOS-XE-vlan:vlan-list": [
            {
                "id": 10,
                "name": "DEVNET-10"
            },
            {
                "id": 11,
                "name": "DEVNET-11"
            },
        ]
    }
}

# additional VLAN to be created
more_vlans = {
    "Cisco-IOS-XE-vlan:vlan-list": [
        {
            "id": 99,
            "name": "DEVNET-99"
        },
    ]
}
```

```
}

# new name for VLAN10
vlan10_newname = { "Cisco-IOS-XE-vlan:name":  
"RESTCONF_Patched_VLAN" }

# fresh start
# PUT command will replace any existing VLAN database with a new
one
restconf_write (url_vlan, "PUT", base_vlans)
Write (PUT) result code: 204

# Read VLAN configuration to confirm it was successfully created
restconf_read (url_vlan)
GET result code: 200
{
  "Cisco-IOS-XE-native:vlan": {
    "Cisco-IOS-XE-vlan:vlan-list": [
      {
        "id": 10,
        "name": "DEVNET-10"
      },
      {
        "id": 11,
        "name": "DEVNET-11"
      }
    ]
  }
}

# add one more VLAN with the POST command
restconf_write (url_vlan, "POST", more_vlans)
Write (POST) result code: 201

# Read VLAN configuration to confirm
restconf_read (url_vlan)
GET result code: 200
{
  "Cisco-IOS-XE-native:vlan": {
    "Cisco-IOS-XE-vlan:vlan-list": [
      {
        "id": 10,
        "name": "DEVNET-10"
      },
      {
        "id": 11,
        "name": "DEVNET-11"
      },
      {
        "id": 12,
        "name": "DEVNET-12"
      }
    ]
  }
}
```

```

        {
            "id": 99,
            "name": "DEVNET-99"
        }
    ]
}

# delete one VLAN we don't need
restconf_delete (url_vlan11)
DELETE result code: 204

# Read VLAN configuration to confirm
restconf_read (url_vlan)
GET result code: 200
{
    "Cisco-IOS-XE-native:vlan": {
        "Cisco-IOS-XE-vlan:vlan-list": [
            {
                "id": 10,
                "name": "DEVNET-10"
            },
            {
                "id": 99,
                "name": "DEVNET-99"
            }
        ]
    }
}

# change the name of VLAN 10
restconf_write (url_vlan10_name, "PATCH", vlan10_newname)
Write (PATCH) result code: 204

restconf_read (url_vlan)
GET result code: 200
{
    "Cisco-IOS-XE-native:vlan": {
        "Cisco-IOS-XE-vlan:vlan-list": [
            {
                "id": 10,
                "name": "RESTCONF-Patched_VLAN"
            },
            {
                "id": 99,
                "name": "DEVNET-99"
            }
        ]
    }
}

```

```
}
```

## Static Routes Configuration

The RESTCONF data tree may be quite deep, and RESTCONF URIs may become very long and hard to read, as you'll see in this section. The starting URI for the static IP routing configuration data is <https://host/restconf/data/Cisco-IOS-XE-native:native/ip/route/>. The same methodology as before was used to research the YANG data tree and find out more specific URIs.

The following code demonstrates how to create, read, update, rewrite, and delete static routes using RESTCONF.

[Click here to view code image](#)

```
# predefine URLs to avoid repetition

# parent leaf for all VLAN configuration
url_route=f"{base}/Cisco-IOS-XE-native:native/ip/route/"

# read only prefix and route name
url_route_filtered=f"{base}/Cisco-IOS-XE-
native:native/ip/route/" \
    "ip-route-interface-forwarding-list?
fields=prefix;fwd-list/name"

# specific leaf for the route name configuration
url_route10_name = f"{base}/Cisco-IOS-XE-native:native/" \
    "ip/route/ip-route-interface-forwarding-
list=10.10.0.0,255.255.0.0/" \
    "fwd-list=192.168.2.1/name/"

# specific leaf for the route configuration
url_route11 = f"{base}/Cisco-IOS-XE-native:native/" \
    "ip/route/ip-route-interface-forwarding-
list=10.11.0.0,255.255.0.0/"

# initial set of routes
base_routes = {
    "Cisco-IOS-XE-native:route": {
        "ip-route-interface-forwarding-list": [
            {
```

```

        "prefix": "10.10.0.0",
        "mask": "255.255.0.0",
        "fwd-list": [
            {
                "fwd": "192.168.2.1",
                "name": "test",
            }
        ]
    },
    {
        "prefix": "10.11.0.0",
        "mask": "255.255.0.0",
        "fwd-list": [
            {
                "fwd": "192.168.2.1"
            }
        ]
    }
]
}
}

# additional route to be created
more_routes = {
    "Cisco-IOS-XE-native:ip-route-interface-forwarding-list": [
        {
            "prefix": "10.20.0.0",
            "mask": "255.255.255.0",
            "fwd-list": [
                {
                    "fwd": "192.168.2.254"
                }
            ]
        }
    ]
}

# new name for route
route10_newname = { "name": "DevNet_route" }

# fresh start
# PUT command will REPLACE any existing static routes with new ones
restconf_write (url_route, "PUT", base_routes)
Write (PUT) result code: 204

# Read static routes to confirm, filter data
restconf_read (url_route_filtered)

```

```
GET result code: 200
{
  "Cisco-IOS-XE-native:ip-route-interface-forwarding-list": [
    {
      "prefix": "10.10.0.0",
      "fwd-list": [
        {
          "name": "test"
        }
      ]
    },
    {
      "prefix": "10.11.0.0"
    }
  ]
}

# add one more route with the POST command
restconf_write (url_route, "POST", more_routes)
Write (POST) result code: 201

# Read static routes to confirm
restconf_read (url_route_filtered)
GET result code: 200
{
  "Cisco-IOS-XE-native:ip-route-interface-forwarding-list": [
    {
      "prefix": "10.10.0.0",
      "fwd-list": [
        {
          "name": "test"
        }
      ]
    },
    {
      "prefix": "10.11.0.0"
    },
    {
      "prefix": "10.20.0.0"
    }
  ]
}

# delete one route we don't need
restconf_delete (url_route11)
DELETE result code: 204

# Read static routes to confirm
```

```
restconf_read (url_route_filtered)
GET result code: 200
{
  "Cisco-IOS-XE-native:ip-route-interface-forwarding-list": [
    {
      "prefix": "10.10.0.0",
      "fwd-list": [
        {
          "name": "test"
        }
      ]
    },
    {
      "prefix": "10.20.0.0"
    }
  ]
}

# change the name of the 10.10.0.0 route
restconf_write (url_route10_name, "PATCH", route10_newname)
Write (PATCH) result code: 204

# Read full static routes information
restconf_read (url_route)
GET result code: 200
{
  "Cisco-IOS-XE-native:route": {
    "ip-route-interface-forwarding-list": [
      {
        "prefix": "10.10.0.0",
        "mask": "255.255.0.0",
        "fwd-list": [
          {
            "fwd": "192.168.2.1",
            "name": "DevNet_route"
          }
        ]
      },
      {
        "prefix": "10.20.0.0",
        "mask": "255.255.255.0",
        "fwd-list": [
          {
            "fwd": "192.168.2.254"
          }
        ]
      }
    ]
  }
}
```

```
    }
```

As a final note on the RESTCONF, we've used the "native" YANG model in this section, but IOS XE supports other models as well (notably, IETF and OpenConfig). IETF is a standards body that builds models through the same process that brings the other IETF standards. OpenConfig is an informal working group of network operators who joined forces to build a set of YANG models that meet their requirements.

They all target the same network features on network devices, but there are some differences between them, such as available configuration elements and data representation formats. The following example reads the IPv4 static routing configuration for the routes configured in the previous example. As you can see, some information, like the "name" for the route, is not available; however, you might find it easier to perform some tasks with this model.

[Click here to view code image](#)

```
url_route = \
    f'{base}/ietf-routing:routing/routing-
instance=default/routing-protocols/" \
    f"routing-protocol=static,1/static-routes/ipv4"

restconf_read (url_route)
GET result code: 200
{
    "ietf-ipv4-unicast-routing:ipv4": {
        "route": [
            {
                "destination-prefix": "10.10.0.0/16",
                "next-hop": {
                    "next-hop-address": "192.168.2.1"
                }
            },
            {
                "destination-prefix": "10.20.0.0/24",
                "next-hop": {
                    "next-hop-address": "192.168.2.254"
                }
            }
        ]
    }
}
```

```
        }
    ]
}
```

## 5.3 Construct a workflow to configure network parameters with:

### 5.3.A Ansible playbook

There are a number of Ansible modules specially designed to work with Cisco IOS devices, as summarized at <https://docs.ansible.com/ansible/latest/collections/cisco/ios/>. Here's a list of some of these modules:

#### - **ios\_command**

([https://docs.ansible.com/ansible/2.9/modules/ios\\_command\\_module.html](https://docs.ansible.com/ansible/2.9/modules/ios_command_module.html))

Used to run commands on remote devices running Cisco IOS. Here's an example:

[Click here to view code image](#)

```
tasks:
  - name: run multiple commands on remote nodes
    ios_command:
      commands:
        - show version
        - show interfaces
```

#### - **ios\_config**

([https://docs.ansible.com/ansible/2.9/modules/ios\\_config\\_module.html](https://docs.ansible.com/ansible/2.9/modules/ios_config_module.html))

Used to manage Cisco IOS configuration. Here's an example:

[Click here to view code image](#)

```

tasks:
  - name: configure top level configuration
    ios_config:
      lines: hostname {{ inventory_hostname }}

  - name: configure interface settings
    ios_config:
      lines:
        - description test interface
        - ip address 172.31.1.1 255.255.255.0
      parents: interface Ethernet1

  - name: load new acl into device
    ios_config:
      parents: ip access-list extended ACL-IN
      lines:
        - 10 permit ip host 1.1.1.1 any log
        - 20 permit ip host 2.2.2.2 any log
        - 30 permit ip host 3.3.3.3 any log
      before:
        - interface GigabitEthernet0/1
        - no ip access-group ACL-IN in
        - no ip access-list extended ACL-IN
      after:
        - interface GigabitEthernet0/1
        - ip access-group ACL-IN in
      match: exact
      replace: block

  - name: save running to startup when modified
    ios_config:
      save_when: modified

```

## - **ios\_static\_route**

([https://docs.ansible.com/ansible/2.9/modules/ios\\_static\\_route\\_module.html](https://docs.ansible.com/ansible/2.9/modules/ios_static_route_module.html))

Used to manage static IP routes on Cisco IOS network devices. Here's an example:

[Click here to view code image](#)

```

tasks:
  - name: configure black hole in vrf blue depending on
    tracked item 10
    ios_static_route:

```

```
prefix: 192.168.2.0
mask: 255.255.255.0
vrf: blue
interface: null0
track: 10

- name: remove route
  ios_static_route:
    prefix: 192.168.2.0
    mask: 255.255.255.0
    next_hop: 10.0.0.1
    state: absent
```

## - **ios\_vlans**

([https://docs.ansible.com/ansible/2.9/modules/ios\\_vlan\\_s\\_module.html](https://docs.ansible.com/ansible/2.9/modules/ios_vlan_s_module.html))

Used to manage VLANs on IOS network devices. Here's an example:

[Click here to view code image](#)

```
tasks:
  - name: Merge provided configuration with device
    configuration
    ios_vlans:
      config:
        - name: Vlan_10
          vlan_id: 10
          state: active
          shutdown: disabled
        - name: Vlan_20
          vlan_id: 20
          state: active
          shutdown: enabled
      state: merged

  - name: Override all VLANs with the provided configuration
    ios_vlans:
      config:
        - name: Vlan_10
          vlan_id: 10
      state: overridden
```

## - **ios\_vrf**

([https://docs.ansible.com/ansible/2.9/modules/ios\\_vrf\\_.html](https://docs.ansible.com/ansible/2.9/modules/ios_vrf_.html))

[module.html](#))

Used to manage the collection of VRF definitions on Cisco IOS devices. Here's an example:

[Click here to view code image](#)

```
tasks:
  - name: configure a vrf named management
    ios_vrf:
      name: management
      description: oob mgmt vrf
      interfaces:
        - Management1

  - name: remove a vrf named test
    ios_vrf:
      name: test
      state: absent

  - name: configure a set of VRFs and purge any others
    ios_vrf:
      vrfs:
        - red
        - blue
        - green
      purge: yes
```

Note: As discussed in [Section 5.4](#), some modules are imperative (for example, `ios_static_route`) and some are declarative (`ios_vlans`, `ios_interfaces` with `state: overridden`, and `ios_vrf` with `purge: yes`).

A sample playbook is shown next; its goal is to help document the existing network by assigning interface descriptions based on CDP/LLDP neighbor information:

[Click here to view code image](#)

```
$ cat hosts.yml
[cisco]
cat9k ansible_host=192.168.2.4

[cisco:vars]
ansible_user=admin
```

```

ansible_password=cisco
ansible_connection=network_cli
ansible_network_os=ios

$ cat cdp-descr.yml
---
- name: Sample Ansible play
  hosts: cisco
  gather_facts: no

  tasks:
    - name: Obtain Cisco device information
      ios_facts:
        gather_subset:
          - interfaces

    - name: Display provisioned device name
      debug:
        msg: "Working on: {{ ansible_net_hostname }}"

    - name: Update interface description based on CDP/LLDP info
      ios_config:
        lines:
          - description Link to {{ item.value[0].port }} on
            {{item.value[0].host }}
            parents: interface {{ item.key }}
            save_when: changed
        with_dict: "{{ansible_facts.net_neighbors }}"

    - name: save running to startup when modified
      ios_config:
        save_when: modified

$ ansible-playbook cdp-descr.yml

PLAY [Sample Ansible play]
*****
****

TASK [Obtain Cisco device information]
*****
***  

ok: [cat9k]

TASK [Display provisioned device name]
*****
***  

ok: [cat9k] => {

```

```

    "msg": "Working on: CAT9K-01"
}

TASK [Update interface description based on CDP/LLDP info]
*****
changed: [cat9k] => (item={'key': 'GigabitEthernet1/0/23',
'value': [{'host': 'CAT9K-04', 'platform': 'cisco C9200-24P',
'port': 'GigabitEthernet1/0/24'}]})

changed: [cat9k] => (item={'key': 'GigabitEthernet1/0/24',
'value': [{'host': 'CAT9K-05', 'platform': 'cisco C9200-24P',
'port': 'GigabitEthernet1/0/24'}]})

changed: [cat9k] => (item={'key': 'GigabitEthernet1/0/21',
'value': [{'host': 'CAT9K-02', 'platform': 'cisco C9200-24P',
'port': 'GigabitEthernet1/0/24'}]})

changed: [cat9k] => (item={'key': 'GigabitEthernet1/0/22',
'value': [{'host': 'CAT9K-03', 'platform': 'cisco C9200-24P',
'port': 'GigabitEthernet1/0/24'}]})

[WARNING]: To ensure idempotency and correct diff the input
configuration lines should be similar to how they appear if
present in the running configuration on device

TASK [save running to startup when modified]
*****
changed: [cat9k]

PLAY RECAP
*****
*****
cat9k : ok=4    changed=2    unreachable=0
failed=0   skipped=0    rescued=0    ignored=0

```

Verification of the device is shown next:

[Click here to view code image](#)

```

CAT9K-01#show run | b net1/0/21
interface GigabitEthernet1/0/21
  description Link to GigabitEthernet1/0/24 on CAT9K-02
!
interface GigabitEthernet1/0/22
  description Link to GigabitEthernet1/0/24 on CAT9K-03
!
interface GigabitEthernet1/0/23
  description Link to GigabitEthernet1/0/24 on CAT9K-04
!
interface GigabitEthernet1/0/24
  description Link to GigabitEthernet1/0/24 on CAT9K-05

```

## 5.3.B Puppet manifest

There are several Puppet modules to work with Cisco IOS devices, as detailed here:

- **cisco\_ios** (by puppetlabs,  
[https://forge.puppet.com/puppetlabs/cisco\\_ios](https://forge.puppet.com/puppetlabs/cisco_ios))

Used to manage the configuration of Cisco Catalyst devices running IOS and IOS-XE. Here's a sample manifest:

[Click here to view code image](#)

```
ntp_server { '1.2.3.4':  
    ensure => 'present',  
    key => 94,  
    prefer => true,  
    minpoll => 4,  
    maxpoll => 14,  
    source_interface => 'Vlan 42',  
}
```

- **ciscopuppet** (by puppetlabs,  
<https://forge.puppet.com/puppetlabs/ciscopuppet>)

Used to manage the configuration of Cisco Nexus devices. Here's a sample manifest:

[Click here to view code image](#)

```
# create "devnet" OSPF process  
cisco_ospf {"devnet":  
    ensure => present,  
}  
  
# default VRF (global) settings for the "devnet" OSPF  
process  
cisco_ospf_vrf {"devnet default":  
    ensure => 'present',  
    default_metric => '5',  
    auto_cost => '46000',  
}
```

```
# interface settings for the "devnet" OSPF process
cisco_interface_ospf {"Ethernet1/2 devnet":
  ensure => present,
  area => 200,
  cost => "200",
}
```

### - **ciscoyang**

(<https://forge.puppet.com/modules/ciscoeng/ciscoyang>)

Used to manage the configuration of Cisco IOS-XR devices through YANG-models in JSON/XML format; it bundles the **cisco\_yang** and **cisco\_yang\_netconf**.

The following sample manifest configures IOS-XR using YANG models in JSON format via gRPC:

[Click here to view code image](#)

```
node 'default' {
  cisco_yang { 'my-config':
    ensure => present,
    target => '{"Cisco-IOS-XR-infra-rsi-cfg:vrfs": [null]}',
    source => '{"Cisco-IOS-XR-infra-rsi-cfg:vrfs": [
      "vrf": [
        {
          "vrf-name": "VOIP",
          "description": "Voice over IP",
          "vpn-id": {
            "vpn-oui": 875,
            "vpn-index": 3
          },
          "create": [
            null
          ]
        }
      ]
    }',
  }
}
```

The following sample manifest configures IOS-XR using YANG models in XML format via NETCONF:

[Click here to view code image](#)

```
node 'default' {
  cisco_yang_netconf { 'my-config':
    target => '<vrfs xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg"/>',
    source => '<vrfs xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg">
      <vrf>
        <vrf-name>INTERNET</vrf-name>
        <create/>
        <description>Generic external
traffic</description>
        <vpn-id>
          <vpn-oui>875</vpn-oui>
          <vpn-index>22</vpn-index>
        </vpn-id>
      </vrf>
    </vrfs>',
    mode => replace,
    force => false,
  }
}
```

## 5.4 Identify a configuration management solution to achieve technical/business requirements

*Configuration management* is the practice of handling changes systematically so that systems maintain their integrity over time. Configuration management tools manage and evaluate proposed changes, track the status of changes, and maintain an inventory as systems change.

Configuration management is an important element of the *infrastructure as code (IaC)* concept. Per Wikipedia, “[I]nfrastructure as code is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.” Or, simply put, IaC means that IT infrastructure is managed using

configuration files rather than manual configuration changes. As with any other code, DevOps practices like version control, testing, and continuous monitoring apply to the infrastructure configuration.

IaC adoption brings the following technical and business benefits:

- **Lower costs:** Both direct (less hardware and fewer people to operate it) and indirect (engineers now have more time and are able to refocus their efforts toward other enterprise tasks).
- **Faster execution:** Complete development (or production) environment may be set up very quickly, allowing other teams across the enterprise to work quickly and more efficiently.
- **Lower risks:** Decreased downtime and increased reliability by making config files the single source of truth. It guarantees that the same configurations would be deployed over and over, without human errors and discrepancies.

Many options are available for IaC implementation, and it is important to be aware of the differences, as detailed in the following sections.

## **Orchestration versus Configuration Management**

Configuration management tools were created to manage existing server instances (for example, to install packages, start services, install scripts and configuration files) in an automated manner. They may execute simple or complex scripted configuration changes on network devices. Most importantly, configuration management tools make sure users don't do manual configuration.

Orchestrators create instances themselves as well as services. They use a policy-driven approach to automation to coordinate hardware and software components (that may span multiple domains) involved in turning up service.

These two categories are not mutually exclusive, as most configuration management tools can do some degree of orchestration, and most orchestration tools can do some degree of configuration management. However, the focus on configuration management or orchestration means that some of the tools are a better fit for certain types of tasks.

## **Declarative (Functional) versus Imperative (Procedural) Approach**

The following table summarizes features of the declarative and imperative approaches to IaC implementation:

	<b>Declarative</b>	<b>Imperative</b>
<b>Directive</b>	“What”	“How”
<b>Defines</b>	What the eventual target configuration should be	How the infrastructure is to be changed to meet this

	<b>Declarative</b>	<b>Imperative</b>
<b>How to get there?</b>	Define <i>the desired state</i> , and the system executes what needs to happen to achieve that desired state	Define <i>specific commands</i> that need to be executed in the appropriate order to reach the desired state

For example, if you have two web servers running and you need five temporarily, with the declarative system you just change the configuration setting to five active servers and then change it back to two. With the imperative, you first create a task to add three more servers (to total five) and then later create another task to remove those three servers.

## **Push/Pull Model**

In a pull model, the *agent* on a server to be configured will *pull* its configuration from the controlling server. In the *push* method, the controlling server *pushes* the configuration to the *agentless* destination system. Let's review the features of some popular solutions.

*Terraform* is one of the most popular orchestrators. It uses a declarative approach and agentless push model. It has some configuration management functionality as well (for example, allowing for VMs to be customized with the standard *cloud-init* and proprietary *Packer* tools).

*Cisco NSO* is a service-provider-grade *network orchestrator* that can automate everything from simple device turn-up, to

cross-domain automation, to sophisticated full lifecycle service management. NSO offers a series of unique capabilities:

- A rich and diverse set of northbound APIs and software interfaces that allow straightforward integration into existing business systems and operational tool chains
- A *multivendor* device abstraction layer that uses *network element drivers (NEDs)* to provide access to both Cisco and more than 150 other-party physical and virtual devices
- A globally scalable, highly available data store for both configuration and state information

*Puppet* is a declarative configuration management tool. It typically uses a pull model and requires agents to be installed on each managed node (agentless management for network device is supported, with the agent on a server acting like a proxy). Agents perform the following tasks:

- They gather *facts* and send them to a master node.
- They check a *catalog* received from a master node and apply the required changes as described.
- They respond to a master node with a report.

*Ansible* is an agentless configuration management tool. It uses *modules* to work with the configuration, and modules may be written in both declarative and imperative ways, making Ansible a hybrid tool.

## 5.5 Describe how to host an application on a network device (including Catalyst 9000 and Cisco IOx-enabled devices)

The move to virtual environments has given rise to the need to build applications that are reusable, portable, and scalable. Application hosting gives administrators a platform for leveraging their own tools and utilities. An application, hosted on a network device, can serve a variety of purposes, such as automation, configuration management, monitoring, and integration with existing tool chains.

Cisco IOx (IOS + Linux) is an application framework that provides application-hosting capabilities for different Cisco network platforms, with Docker as a containerization platform. Administrators can use prepackaged Docker containers or easily create and deploy their own.

Application hosting is generally supported on Catalyst 9000, ISR 4K, CSR1000v, and ASR1K platforms; however, hardware requirements and functionality may differ between platform and IOS XE versions, so make sure to check the documentation when starting your projects.

Application hosting is designed in a way to prevent performance and security impact on the main system:

- Memory and CPU usage for applications is bounded using control groups (cgroups).
- Process and file access is isolated and restricted (using user namespace).
- Disk usage is isolated using separate storage.

There are two major requirements to enable application hosting on Catalyst 9000:

- External storage. Two options are available, depending on the platform: USB 3.0 or SATA SSD drives, with AES-256 encryption (as a sidenote, starting Cisco IOS XE 17.3.3, application hosting is supported on the internal bootflash, but only for Cisco-signed applications).

- DNA-Advantage licensing.

## Application Hosting Lifecycle

Application hosting only supports Docker containers. Applications need to be packaged as TAR files and are created from container images by the Docker with the “save” command (for more details on how to work with containers, refer to [Section 4.4: Utilize Docker to containerize an application](#)).

The following steps summarize the application hosting lifecycle:

- Create a TAR file from the Docker container (on a Linux box):

[Click here to view code image](#)

```
$ docker save iox_app -o iox_app.tar
```

- Enable the IOx subsystem on a network device:

[Click here to view code image](#)

```
Device(config)#iox
*Apr  6 15:28:22: %IM-6-IOX_ENABLEMENT: Switch 1 R0/0:
ioxman: IOX is ready.

Device# show iox-service

IOx Infrastructure Summary:
-----
IOx service (CAF) 1.11.0.4      : Running
IOx service (HA)                 : Running
IOx service (IOxman)              : Running
Libvирtd 1.3.4                  : Running
```

- Tune the container settings (networking, resources, Docker options) in the “app-hosting appid” configuration section (refer to the IOS XE Programmability Configuration Guide).

- Install the application container image (resources are not committed to the application yet):

[Click here to view code image](#)

```
Device# app-hosting install appid iox_app package  
usbflash1:iox_app.tar
```

- Activate the application (required resources will be committed to the application):

[Click here to view code image](#)

```
Device# app-hosting activate appid iox_app
```

- Start the application:

[Click here to view code image](#)

```
Device# app-hosting start appid iox_app
```

- (When completed) stop the application:

[Click here to view code image](#)

```
Device# app-hosting stop appid iox_app
```

- Deactivate all the resources allocated for the application:

[Click here to view code image](#)

```
Device# app-hosting deactivate appid iox_app
```

- Uninstall the application:

[Click here to view code image](#)

```
Device# app-hosting uninstall appid iox_app
```

## Guest Shell

The Guest Shell is a special container bundled with the system image, designed to run Linux applications, including

Python for automated control and management of Cisco devices.

Users can access Guest Shell and update scripts and software packages within the container; however, they cannot modify the host file system and processes within the Guest Shell.

The following example demonstrates how to run the Guest Shell and how to interact with the host device's configuration from it using built-in tools:

[Click here to view code image](#)

```
CAT9K(config)#app-hosting appid guestshell
CAT9K(config-app-hosting)# app-vnic management guest-interface 0

CAT9K# guestshell enable
Guestshell enabled successfully

CAT9K# guestshell run bash
[guestshell@guestshell ~]$ dohost "show system mtu"
Global Ethernet MTU is 1500 bytes.

[guestshell@guestshell ~]$ dohost "conf t ; hostname CAT9K-bash"
[guestshell@guestshell ~]$ exit

CAT9K-bash# guestshell run python3
Python 3.6.8 (default, Nov 21 2019, 22:10:21)
[GCC 8.3.1 20190507 (Red Hat 8.3.1-4)] on linux
>>> import cli
>>> cli.cli("show system mtu")
'Global Ethernet MTU is 1500 bytes.\n'
>>> cli.configurep("hostname CAT9K-python")
'Line 1 SUCCESS:  hostname CAT9K-python\n'
>>> quit()

CAT9K-python#
```

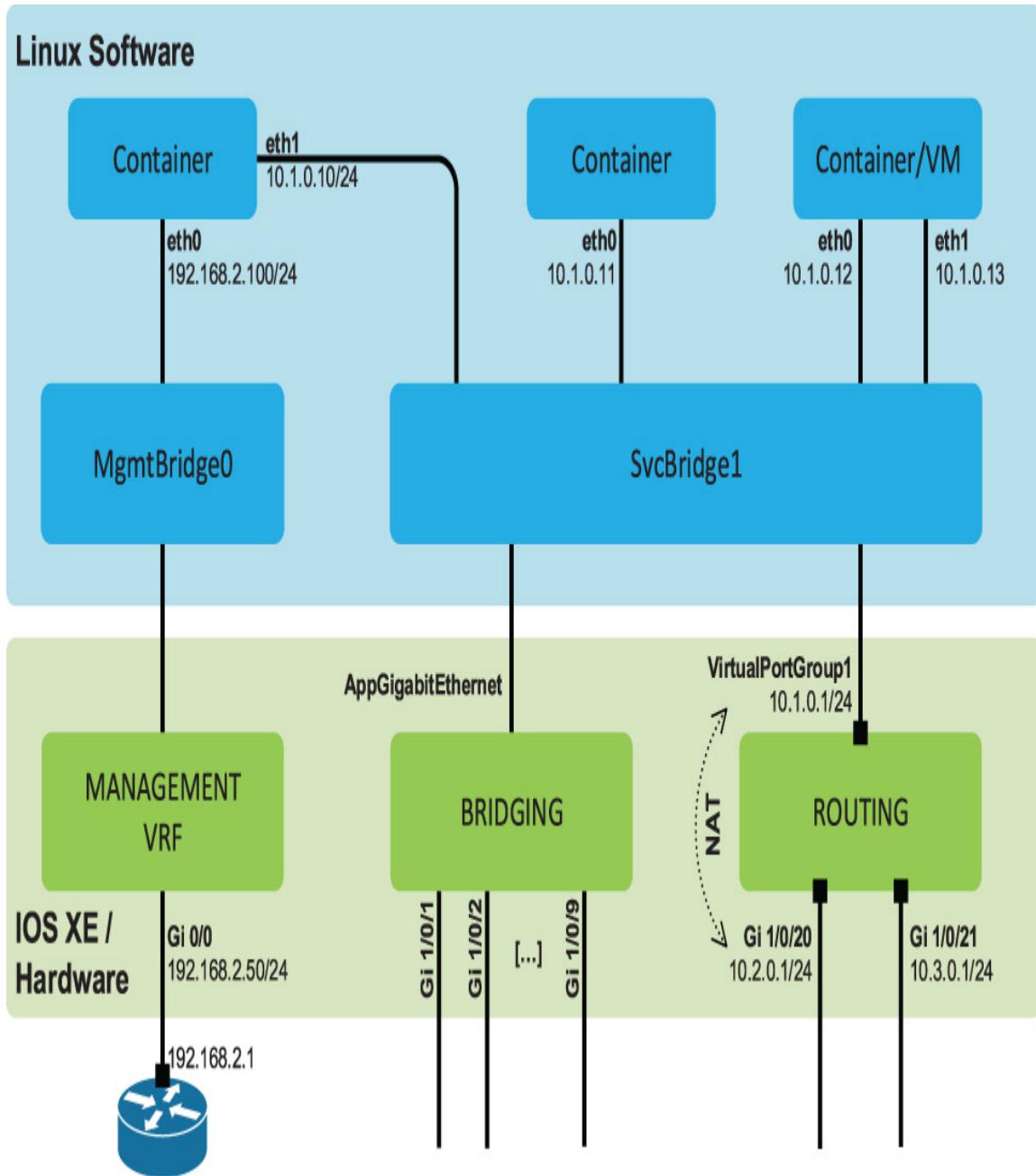
The Guest Shell is used in the Zero-Touch Provisioning (ZTP) process, where it downloads and executes the bootstrap Python script that configures the device.

## Application Hosting Networking

To provide network connectivity to containers, at least one virtual Ethernet pair is created for each connection. One interface of this pair, called the *virtual network interface card (vNIC)*, is part of the application container. Cisco IOx is responsible for assigning the IP address and unique MAC address to a vNIC in a container and for connecting it to the platform's data plane. Inside a container, vNICs look like a standard ethX Ethernet interface.

[Figure 36](#) summarizes all network connectivity options for applications, but be aware that not all options are available on all platforms.

*Figure 36: IOS XE Application Hosting Networking*



There are three ways to connect containers to the external networks: via the Management interface, via L2 data ports, and via L3 routed interface.

Note: “guest-interface x” maps to the container’s vNIC and the Linux *eth* interface with the same number.

Note: IP information may be assigned to containers statically (in IOS XE config), manually with Linux commands, or with DHCP. The following examples use static configuration.

## Connectivity via Management Interface

The container's vNIC is connected via the L2 bridge to the device's management interface. As such, the container's IP address should be assigned to the same IP subnet. Here's an example:

[Click here to view code image](#)

```
interface GigabitEthernet 0/0
    vrf forwarding Mgmt-vrf
    ip address 192.168.2.50 255.255.255.0

app-hosting appid iox_app
    app-vnic management guest-interface 0
        guest-ipaddress 192.168.2.100 netmask 255.255.255.0
    !
    name-server0 8.8.8.8
    app-default-gateway 192.168.2.1 guest-interface 0
```

## Connectivity via Front Panel (Regular) L2 Interfaces

The container's vNIC is connected via the L2 bridge to the special AppGigabitEthernet internal hardware interface. AppGigabitEthernet connects applications to the hardware switch fabric and the front panel data ports. Containers can connect to AppGigabitEthernet in two modes: VLAN (where a single untagged VLAN is mapped to a container's interface) and Trunk (where traffic for all the allowed VLANs is sent as Native or VLAN-tagged frames). Here's an example:

[Click here to view code image](#)

```

interface GigabitEthernet1/0/1
  switchport trunk allowed vlan 10-12
  switchport mode trunk
!
interface GigabitEthernet1/0/2
  switchport mode access
  switchport access vlan 20
!
interface AppGigabitEthernet
  switchport trunk allowed vlan 10-12,20
  switchport mode trunk
!
app-hosting appid iox_app_vlan
  app-vnic AppGigabitEthernet trunk
    vlan 10 guest-interface 0
    guest-ipaddress 192.168.0.1 netmask 255.255.255.0
!
app-hosting appid iox_app_trunk
  app-vnic AppGigabitEthernet trunk
    guest-interface 0

```

## Connectivity via L3 Routed Interface

In such a configuration, the device itself will be a default gateway for the container. For the external connectivity, NAT might be required and may be performed on the device itself (with the VirtualPortGroup interface as “inside” and outgoing routing interface as “outside”). Here’s an example:

[Click here to view code image](#)

```

interface GigabitEthernet1/0/20
  ip address dhcp
  ip nat outside
!
interface VirtualPortGroup1
  ip address 10.1.0.1 255.255.255.0
  ip nat inside
!
ip nat inside source list NAT_ACL interface
GigabitEthernet1/0/20 overload
ip access-list standard NAT_ACL
  permit 10.1.0.0 0.0.0.255

app-hosting appid iox_app

```

```
app-vnic gateway1 virtualportgroup 1 guest-interface 0 guest-
ipaddress 10.1.0.10 netmask 255.255.255.0
app-default-gateway 10.1.0.1 guest-interface 0
```

## 5.6 Chapter 5 Review Questions

**Q1:** Which statement is true about model-driven telemetry (MDT)?

- A) MDT is similar to the traditional SNMP-based monitoring, just the encoding format is different.
- B) Data streamed from publishers to data collectors is YANG-modeled.
- C) All MDT subscriptions must be added to the running configuration on network devices.
- D) Dial-in MDT sessions automatically reconnect to the receiver after a reload.

**Q2:** Which statement is true about YANG data modeling?

- A) YANG structures data models into modules and submodules.
- B) YANG defines four types of data nodes: leaf nodes, leaf-list nodes, instance nodes, and list nodes.
- C) YANG defines data elements types, their structures, and the actual data.
- D) A container node may contain child nodes of any type except container to avoid recursion.

**Q3:** Which statement is true about RESTCONF?

- A) RESTCONF functionally is very similar to NETCONF, but differences are in the transports and formats.
- B) RESTCONF operations (read/write/delete) are specified in the request's body.
- C) The resource target of a RESTCONF operation is specified in the request's body.

D) Network devices may support more than one YANG model.

**Q4: Which is the correct RESTCONF base URL for a Layer 2 VLAN 10 configuration on an IOS XE device?**

- A) `https://host:port/restconf/data/Cisco-IOS-XE-native:native/interface/Vlan=10/`
- B) `https://host:port/restconf/interface/Cisco-IOS-XE-native:native/data/Vlan=10/`
- C) `https://host:port/restconf/data/Cisco-IOS-XE-native:native/vlan/vlan-list=10/`
- D) `https://host:port/restconf/vlan/Cisco-IOS-XE-native:native/data/vlan-list=10/`

**Q5: Which statement is true about configuring network parameters with the Ansible?**

- A) The “state: overridden” parameter, where supported, may be used to make sure only the Ansible-provided configuration is present on a device.
- B) “ios\_config” is the most comprehensive module to configure Cisco devices, so the other modules are redundant and considered legacy.
- C) The “ios\_command” module allows “show” commands and one-line configuration.
- D) The “state: absent” parameter, where supported, may be used to apply configuration only if it is not already present on a device.

**Q6: Which statement is true about configuring network parameters with Puppet?**

- A) The “mode” parameter determines whether or not the config should be present on a device.
- B) The “ensure” parameter indicates whether the provided config is a merge or a replacement of the

- existing config on a device.
- C) The “target” parameter of the ciscoyang module indicates the RESTCONF URL for the POST operation.
  - D) The “source” property of the ciscoyang module contains the model data in YANG JSON or YANG XML NETCONF format or a reference to a local file containing the model data.

**Q7: Which statement is true about configuration management solutions?**

- A) The imperative approach defines what the eventual target configuration should be.
- B) Configuration management enables the infrastructure as code (IaC) concept.
- C) Orchestration automates the configuration of service elements.
- D) Terraform agents pull their settings from the central server and translate them into the appropriate local configuration.

**Q8: Which statement is true about application hosting on Cisco IOx-enabled network devices?**

- A) CPU and resources are shared with the main system, so application hosting needs to be used with care.
- B) DNA-Essentials licensing is required for the application hosting.
- C) Extra storage (SSD or external USB) is required for the application hosting.
- D) Only Cisco-signed applications may be hosted on Cisco network devices.

**Q9: Which is *not* a step in the application hosting lifecycle on a Cisco IOS XE device?**

- A) Enable IOx subsystem: `Device(config)# iox`

- B) Copy application container image: Device# copy  
usbflash1:my\_iox\_app.tar flash:
- C) Activate the application: Device# app-hosting activate  
appid iox\_app
- D) Start the application: Device# app-hosting start appid  
iox\_app

**Q10: Which statement is *not* true about application hosting networking?**

- A) IP information may only be assigned to the hosted container via IOS XE config.
- B) When connecting externally via the Management interface, the container should be assigned an IP address from the same IP range as the Management interface.
- C) When connecting externally via front panel interfaces, the internal AppGigabitEthernet interface connects applications to the hardware switch fabric and the front panel data ports.
- D) When using the VirtualPortGroup interface, the hosting device becomes the default gateway for the container and routes its traffic.

---

## *6. Appendix A: RESTCONF URI Demystified (IOS XE)*

---

The RESTCONF protocol uses secure HTTP methods to provide CREATE, READ, UPDATE, and DELETE (CRUD) operations on a device's YANG-modelled data store. For those operations, HTTP URI acts as a location identifier within the hierarchical data tree. The following example shows how to read the complete configuration for the SVI Vlan10 interface using Cisco's "native" model:

[Click here to view code image](#)

```
$ curl -k -u admin:cisco -H 'Accept: application/yang-data+json'  
\  
https://192.168.2.4/restconf/data/Cisco-IOS-XE-  
native:native/interface/Vlan=10  
  
{  
  
  "Cisco-IOS-XE-native:Vlan": {  
  
    "name": 10,  
  
    "description": "DevNet Vlan SVI interface",  
  
    "ip": {  
  
      "address": {
```

```
        "primary": {  
  
            "address": "10.10.0.1",  
  
            "mask": "255.255.255.0"  
  
        }  
  
    }  
  
}  
  
}  
  
}
```

The payload is easy to understand and work with, but how do you find a correct URI? Some examples are included in Cisco documentation, but the goal of this appendix is to demonstrate how to do your own research from scratch. Curl is used as an HTTP tool from the CLI in this activity, but any other program may be used (for example, Postman).

A YANG-modeled data store is structured as a hierarchical data tree. To traverse it, first, the root of the tree needs to be identified. RESTCONF RFC 8040 specifies the standard `/well-known/host-meta` endpoint for it:

[Click here to view code image](#)

```
$ curl -k -u admin:cisco \  
  
https://192.168.2.4/.well-known/host-meta  
  
<XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
```

```
<Link rel='restconf' href='/restconf' />

</XRD>
```

It shows that `/restconf` is a root. Let's query it:

[Click here to view code image](#)

```
$ curl -k -u admin:cisco \
https://192.168.2.4/restconf/

<restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">

<data/>

<operations/>

<yang-library-version>2016-06-21</yang-library-version>

</restconf>
```

The root has a few elements, including the `/restconf/data/` branch, which represents all the configuration and state resources (mandatory as per RFC 8040). Resources are defined in the specific YANG modules under this, but their list and definitions (schemas) are needed to be able to use them.

Using the standard mechanism (RFC 7895), the list of the supported YANG modules may be retrieved by reading the “modules-state” container of the “ietf-yang-library” module:

[Click here to view code image](#)

```
$ curl -k -u admin:cisco -H 'Accept: application/yang-data+json'
\
```

```

https://192.168.2.4/restconf/data/modules-state/

{

    "ietf-yang-library:modules-state": {

        "module-set-id": "349ef7f26dc8311bd6c10c0640b98636",

        "module": [

            {

                "name": "BGP4-MIB",

                "revision": "1994-05-05",

                "schema": "https://192.168.2.4:443/restconf/tailf/modules/BGP4-MIB/1994-05-05",

                "namespace": "urn:ietf:params:xml:ns:yang:smiv2:BGP4-MIB",

                "conformance-type": "implement"

            },

            [... rest skipped ...]
    }
}

```

The list is long (several hundred modules), and each module contains an on-device link to its YANG model schema in the “schema” element. This information is also available directly from the public YANG repository, which contains many models, and particularly for Cisco IOS XE at <https://github.com/YangModels/yang/tree/master/vendor/cisco>

`co/xe`. On-device discovery is typically used in automated processing, but for manual research, like in this case, it is more convenient to download all the modules at once from the repository.

YANG structures are oriented for machine consumption and are not very human friendly, so to navigate them, it's best to use specialized tools like the CLI-oriented pyang (<https://github.com/mbj4668/pyang>) or the GUI-based ANX (<https://github.com/cisco-ie/anx>). The following examples use pyang to visualize YANG tree structures (check out `pyang --tree-help` to better understand its output):

[Click here to view code image](#)

```
$ python3 -m venv pyang

$ source pyang/bin/activate

(pyang)$ cd pyang/

(pyang)$ pip install pyang

[...]

Successfully installed lxml-4.6.3 pyang-2.4.0

(pyang)$ git clone https://github.com/YangModels/yang.git

[...]

Checking out files: 100% (44132/44132), done.

(pyang)$ cd yang/vendor/cisco/xe/1731/
```

```
(pyang)$ pyang Cisco-IOS-XE-native.yang -f tree --ignore-errors  
--tree-depth=1
```

```
module: Cisco-IOS-XE-native
```

```
    +-+ rw native
```

The following arguments are used when running pyang:

- `Cisco-IOS-XE-native.yang`: The name of the YANG module for the Cisco native model.
- `-f tree`: Indicates the *tree* output format.
- `--tree-path`: The starting branch within a tree.
- `--tree-depth`: The levels of the tree to print. The amount of output is extensive on the top levels, so this parameter limits it.
- `--ignore-errors`: Do not display error messages.

The “Cisco-IOS-XE-native” module has a single element, called “native.” The corresponding RESTCONF path for it is `/restconf/data/Cisco-IOS-XE-native:native` (and `/restconf/data/native` works, too, but it’s best to explicitly indicate the module name to avoid conflicts).

Here are the sub-elements of the “native” element (there’s lots of output, but we’re interested in the “interface” section):

[Click here to view code image](#)

```
(pyang)$ pyang Cisco-IOS-XE-native.yang -f tree --ignore-errors  
\
```

```
--tree-path=native --tree-depth=2
```

```
module: Cisco-IOS-XE-native
```

```
    +-+ rw native
```

```
        +-+ rw default
```

```
        |      ...
```

```
        +-+ rw bfd
```

```
[.....]
```

```
        +-+ rw interface
```

```
[.....]
```

Now let's explore sub-elements of the "interface" element (lots of interface type, but we're interested in the "Vlan" section). Notice how the tree path starts to build up:

[Click here to view code image](#)

```
(pyang)$ pyang Cisco-IOS-XE-native.yang -f tree --ignore-errors\n
```

```
--tree-path=native/interface --tree-depth=3
```

```
module: Cisco-IOS-XE-native
```

```
+--rw native  
  
    +-rw interface  
  
        +-rw AppNav-Compress* [name]  
  
[.....]  
  
        +-rw GigabitEthernet* [name]  
  
[.....]  
  
        +-rw Vlan* [name]
```

And one step further is a section for “Vlan”-type interfaces definition:

[Click here to view code image](#)

```
(pyang)$ pyang Cisco-IOS-XE-native.yang -f tree --ignore-errors  
\  
--tree-path=native/interface/Vlan  
  
module: Cisco-IOS-XE-native  
  
    +-rw native  
  
        +-rw interface  
  
            +-rw Vlan* [name]  
  
                +-rw name          uint16
```

```
+--rw description?          string  
  
+--rw switchport-conf  
  
|   +--rw switchport?    boolean  
  
[... rest skipped ...]
```

The resulting path within a tree for “Vlan” interfaces is “native/interface/Vlan,” which maps into the

/restconf/data/Cisco-IOS-XE-native:native/interface/Vlan  
RESTCONF path:

[Click here to view code image](#)

```
$ curl -k -u admin:cisco -H 'Accept: application/yang-data+json'  
\  
https://192.168.2.4/restconf/data/Cisco-IOS-XE-native:native/interface/Vlan  
  
{  
  
  "Cisco-IOS-XE-native:Vlan": [  
  
    {  
  
      "name": 1,  
  
      "ip": {  
  
        "address": {  
  
          "primary": {
```

```
        "address": "192.168.2.4",  
  
        "mask": "255.255.255.0"  
  
    }  
  
}  
  
}  
  
,  
  
{  
  
    "name": 10,  
  
    "description": "DevNet Vlan SVI interface",  
  
    "ip": {  
  
        "address": {  
  
            "primary": {  
  
                "address": "10.10.0.1",  
  
                "mask": "255.255.255.0"  
  
            }  
  
        }  
  
    }  
  
}
```

```
    }  
  
]  
  
}
```

But how does one address a specific element (for example, Vlan10)?

Note that in pyang output, the “Vlan” element shows as `+--  
rw Vlan* [name]`, where `*` indicates that this is a list/leaf-list, and `[name]` indicates that the “name” element is a key for this list. Using RESTCONF syntax to refer to the element with the name “10,” the final URI is the following (the same as at the beginning of this section):

[Click here to view code image](#)

```
$ curl -k -u admin:cisco -H 'Accept: application/yang-data+json'  
\  
  
https://192.168.2.4/restconf/data/Cisco-IOS-XE-  
native:interface/Vlan=10  
  
{  
  
  "Cisco-IOS-XE-native:Vlan": {  
  
    "name": 10,  
  
    "description": "DevNet Vlan SVI interface",  
  
    "ip": {  
  
      "address": {  
  
        "primary": {
```

```
        "address": "10.10.0.1",

        "mask": "255.255.255.0"

    }

}

}

}

}
```

Now for a few more notes for your reference.

Sometimes keys are composite, consisting of several elements. In that case, all of them should be provided, separated by commas (for example, for the routes, the keys are subnet and mask):

[Click here to view code image](#)

```
(pyang)$ pyang Cisco-IOS-XE-native.yang -f tree --ignore-errors \
--tree-path=native/ip/route/ip-route-interface-forwarding-list \
-tree-depth=4

module: Cisco-IOS-XE-native

    +-+ rw native

        +-+ rw ip
```

```
+--rw route

    +-rw ip-route-interface-forwarding-list* [prefix
mask]

$ curl -k -u admin:cisco -H 'Accept: application/yang-data+json'
\

https://192.168.2.4/restconf/data/Cisco-IOS-XE-
native:native/ip/route/ \
ip-route-interface-forwarding-list=0.0.0.0,0.0.0.0

{

"Cisco-IOS-XE-native:ip-route-interface-forwarding-list": {

    "prefix": "0.0.0.0",

    "mask": "0.0.0.0",

    "fwd-list": [

        {

            "fwd": "192.168.2.1"

        }

    ]

}

}
```

```
}
```

Queries may be amended with some additional parameters to impact what kind of information is returned. For example, adding a field parameter would result in returning only specified fields, not the whole subtree (compare with the previous result):

[Click here to view code image](#)

```
$ curl -k -u admin:cisco -H 'Accept: application/yang-data+json'  
\  
"https://192.168.2.4/restconf/data/Cisco-IOS-XE-native:native/"\  
  
"interface/Vlan=10?fields=name;description"  
  
{  
  
"Cisco-IOS-XE-native:Vlan": {  
  
    "name": 10,  
  
    "description": "DevNet Vlan SVI interface"  
  
}  
  
}
```

The full list of the options supported by the device may be obtained at the `/restconf/data/ietf-restconf-monitoring:restconf-state/capabilities` URL.

---

## *7. Appendix B: Answers to Chapter Review Questions*

---

### **7.1 Answers for Chapter 1: Software Development and Design**

**Q1: Which statement is true about the front end and back end in the distributed applications design?**

- A) The front end is a part of an application that executes the application's business logic.
- B) Some of the back-end functionality may be offloaded to the front end.
- C) The front end may be written in HTML, JavaScript, or Python.
- D) Load balancers can only distribute the load equally among back-end servers.

**Correct Answer: B (Sec 1.1)**

- A: The front end is everything a user sees and interacts with. The back end is the part of the application that executes business logic.
- C: The most common front-end languages are HTML, CSS, and JavaScript. Back-end development languages are Java, C++, Python, and PHP.
- D: Load may be distributed across servers equally or based on their utilization, response time, number of connections, and so on.

**Q2: Which statement is true about modular application design?**

- A) Modules contain everything necessary to execute a certain aspect of the application functionality and therefore are hard to replace with another module.
- B) A module may be easily replaced with another module as soon as their “implementation” sections are similar.
- C) Project work may be broken down into smaller independent projects.
- D) Modular code is more complicated to read and maintain compared to nonmodular.

**Correct Answer: C (Sec 1.2)**

- A: Modules are interchangeable. This is one of the benefits of the modular design.
- B: A module’s “implementation” section is hidden from the other parts of the code. You can easily replace one module with another as soon as their *interface* is the same.
- D: Software logic is expressed more clearly in a modular design, and as a result, code is easier to understand and maintain.

**Q3: Which statement is true about scalable application design?**

- A) Scaling up is a simple way to increase the reliability of a system.
- B) Monolithic applications may be scaled out as easily as microservices.
- C) Implementing load balancing is one of the methods to scale applications horizontally.

D) Performance scalability is the only way to achieve application scalability.

**Correct Answer: C (Sec 1.1, 1.2)**

- A: Scaling up resolves performance issues but does not address reliability, as a system is still a single point of failure.
- B: Scaling out monolithic applications typically requires some redesign, whereas with distributed applications, like microservices, new instances of components may be easily added.
- C: Scalability applies to several dimensions: administrative, functional, load, heterogeneous, and so on.

**Q4: Which statement is true about high availability in the application design?**

- A) Clustered applications continue to provide service even when some of their components fail.
- B) Availability is expressed as percentage of downtime over a period of time.
- C) Stateful data flows help achieve higher availability.
- D) As soon as the service is operational, there's no need to worry about its internal components.

**Correct Answer: A (Sec 1.3)**

- B: Availability is expressed as percentage of *uptime* over a period of time.
- C: Statefulness creates single points of failure in software designs, which decreases the availability.
- D: Internal failures may be transparent to users but should be detected and acted upon.

**Q5: Which statement is true about resilience in the application design?**

- A) Using timeouts helps improve resilience, with higher timeout resulting in better resiliency.
- B) Parameter checking in functions and objects improves application resilience.
- C) Resilience tries to detect and work around the failure by replacing a broken component
- D) If a request fails on a first attempt, the application should fail fast to prevent any further waste of system resources

**Correct Answer: B (Sec 1.3)**

- A: Timeouts should be high enough to allow slower responses but low enough to stop waiting for a response that is never going to arrive, so some tuning is required.
- C: Resilience tries to live through the failure using error handling, while high availability's goal is to work around the failure by replacing a broken component.
- D: “Fail fast” skips costly actions when failures are expected, which is not the case after just one call. Retrying the same operation a few times in the hope to work around a temporary problem is a standard method to make an application more resilient.

**Q6: Which statement is true about latency in the application design?**

- A) Latency is not a concern for the distributed microservices applications for which components are running within the same data center.
- B) End-user latency is based on the physical limitations (distance, speed of light) and therefore cannot be improved.

- C) For a better user experience, it is more optimal to load data in small blocks instead of doing it in one big chunk.
- D) Using UDP instead of TCP for video-conferencing data may result in unpredictable conversation latency and a bad user experience.

**Correct Answer: C (Sec 1.4)**

- A: Latency between the distributed application components is very low and close to zero but is not zero, as cabling distance latency is just one element of it. Others are application latency, OS latency, and NIC latency.
- B: Latency may be reduced by bringing content closer to the end user; for example, using content delivery networks (CDNs).
- D: In interactive applications, it is better to ignore lost/delayed packets because their retransmission would only introduce latency and result in a poorer user experience. UDP is a better choice for such uses because it allows lost packets to be ignored, whereas TCP has a built-in retransmission mechanism.

**Q7: Which statement is true about rate-limiting?**

- A) In REST API calls, an HTTP response with code 329 and a redirect URL indicates that the API is being overused.
- B) A rate-limiting mechanism may be used in processes other than REST API calls.
- C) An exponential token is a rate-limiting mechanism where the number of tokens in the bucket is doubled after each successful attempt.
- D) Disconnecting a user after an unsuccessful login attempt is an example of rate-limiting.

**Correct Answer: B (Sec 1.4)**

- A: The HTTP “Too Many Requests” error code is 429. A response may include the retry time interval, but not the URL: it will be the same as in the original request.
- C: There’s no *exponential token* rate-limiting mechanism. The popular ones are *token bucket* and *exponential backoff*.
- D: This is an example of access control. An example of rate-limiting would be to allow only three login attempts per minute.

**Q8: Which statement is true about maintainability in the application design and implementation?**

- A) Automated testing helps achieve better software code.
- B) High-quality code requires more effort and therefore is harder to maintain.
- C) Nonmodular code is better documented and is easier to maintain.
- D) Modules are independent of each other, so it is a good idea to use the most suitable tools, libraries, and naming conventions when working on each of them.

**Correct Answer: A (Sec 1.5)**

- B: Writing quality code may require more effort, but it would require less maintenance work later on.
- C: Modularity allows easier changes in the source code and therefore makes it more maintainable.
- D: While this statement is true in general, from the code maintainability perspective, it is best if the same naming conventions, libraries, and tools are used throughout the whole project.

**Q9: Which statement is true about observability in the application design?**

- A) Logs, metrics, and release packaging are the three pillars of observability.
- B) Distributed tracing brings visibility into the lifetime of a request across several systems.
- C) Excessively detailed logging puts an extra load on the application, so it should be avoided.
- D) Observability is just another name for traditional monitoring.

**Correct Answer: B (Sec 1.6)**

- A: Logs, metrics, and *distributed tracing* are the three pillars of observability.
- C: Detailed event and state logging help achieve better application observability.
- D: Observability is a superset of monitoring: while providing the same operating details, observability also supplies details on *why* an application behaves in a certain way.

**Q10: Which statement is true about using event logs to diagnose problems with an application?**

- A) If an application is having problems but the event logs indicate there were no crashes, then the problem is external to the application and should be addressed by other teams.
- B) The downside of Syslog is that it does not include a timestamp in logs. It is resolved in Syslog-*ng*.
- C) It is easier to troubleshoot problems when logs contain enough context around events.
- D) If a Python application crashes with an exception, adding the “try / except” wrapping around failed

operation is enough to address the problem.

**Correct Answer: C (Sec 1.7)**

- A: Application problems do not necessarily result in a crash; it could be a fault in the application logic.
- B: Syslog format does include a timestamp.
- D: “Try / except” is a valid technique to avoid crashes, but the surrounding context needs to be reviewed as well. For example, a crash could’ve been a result of a function returning the unexpected value.

**Q11: Which is *not* a primary factor to be considered when selecting a database system for the application?**

- A) The kinds of data that will be stored
- B) Simplicity of the installation process
- C) The ability to perform complex queries
- D) Data model flexibility

**Correct Answer: B (Sec 1.8)**

- B: Database selection should be driven by the application requirements, not operational processes.

**Q12: Which statement is *not* true about relational databases?**

- A) Transactions always result in a valid state of the database.
- B) All operations in a transaction succeed, or every operation is rolled back.
- C) Relational databases focus on the consistency and availability of data.
- D) A relational database is not a good choice for complex querying.

**Correct Answer: D (Sec 1.8)**

D: Relational databases are very well suited for complex querying and non-real-time analytics applications.

**Q13:** A manufacturing plant is equipped with various types of specialized sensors that send their readings to a custom application. What type of database is most suitable to store sensor data and allow real-time and historical data analytics?

- A) Key-value database
- B) Relational database
- C) Graph databases
- D) Time-series database

**Correct Answer:** D (Sec 1.8)

- A: Readings are “key”:“value” (“sensor”:“reading”) records, but the analytics would require several keys (timestamps, sensor, data type, and so on), and key-value databases are not efficient enough for such use.
- B: Relational databases are less efficient and less flexible for such application.
- C: Sensors are unrelated to each other, so there’s no benefit in using graphs.

**Q14:** Which statement is true about the monolithic architectural pattern?

- A) Monolithic applications are self-contained and have no dependencies on other applications.
- B) In monolithic applications, the user interface, business logic, and database are combined into a single executable.
- C) Monolithic applications are hard to scale both vertically and horizontally.
- D) Monolithic applications are a thing of the past, and creating them is a sign of a poor project planning.

**Correct Answer: A (Sec 1.9)**

- B: The database is not a part of the monolith application, but all the database logic is, along with the user interface and business logic.
- C: Vertical scaling (while it can be done) is relatively easy for monolith applications.
- D: Monolith applications are still widely used and are the right choice as soon as they deliver the required functionality.

**Q15: Which statement is true about the service-oriented architecture (SOA) pattern?**

- A) Services are reusable blocks of code that are called directly from within the application code.
- B) SOA data models, protocols, and technologies are standardized, which allows services to call other services with ease.
- C) Services can be defined in business rather than technical SOA terms, which improves collaboration between business and IT.
- D) Services are stateful to allow visibility into the service state for the consumers.

**Correct Answer: C (Sec 1.9)**

- A: In SOA, application components provide services to other elements through a communication protocol over a network. Functions and objects are examples of reusable blocks of code, not services.
- B: While some common principles are established for SOA, there are no industry standards for it. Enterprise Services Bus is SOA's component that performs integration between services.
- D: Services are stateless and abstracted, and their inner logic is hidden from the consumers.

**Q16: Which statement is true about the microservices architectural pattern?**

- A) Microservices are much smaller but more complicated compared to SOA.
- B) Microservices applications are easier to develop, maintain, and operate.
- C) Microservices are best developed when all teams use the same programming language and tools.
- D) With microservices, there is significant communication happening between different components of the system, which adds overhead and latency.

**Correct Answer: D (Sec 1.9)**

- A: Microservices are smaller and less complicated, as they are focused on doing just one thing.
- B: Because the number of services is higher, all these tasks are more complicated, and DevOps practices such as CI/CD are necessary to effectively perform them.
- C: Microservices are independent of each other, so development teams can use any available tools that fit best.

**Q17: Which statement is true about the event-driven architecture (EDA)?**

- A) Reactions to events are immediate.
- B) The event broker topology is better suited for complex situations than the mediator one.
- C) EDA is the most scalable but at the cost of higher implementation and maintenance complexity.
- D) The event broker provides details about event creators to event processors in case they need to

obtain more information about events.

**Correct Answer: C (Sec 1.9)**

- A: Events are asynchronous and may trigger when resources are not available. Events will be stored in an event queue until resources become available.
- B: The mediator topology is better suited for more complex situations where coordination or orchestration of the event processing is required.
- D: EDA is extremely loosely coupled: the event creator does not know by whom it will be processed, and event processors are not aware who created these events.

**Q18: Which statement is true about Git branch merging?**

- A) HEAD should be pointing to the correct receiving branch when you issue the `merge` command.
- B) Content of the staging area is added to the receiving branch before the merge.
- C) Rebase operation retains all changes to and history of the merged branch.
- D) If the “fast-forward” merge is not possible, Git requires a manual intervention to resolve a conflict.

**Correct Answer: A (Sec 1.10a)**

- B: The working and staging areas need to be clean; otherwise, `merge` would fail with an error.
- C: Rebase adds merged commits at the end of the target branch while the changes and the history of the original branch are lost.
- D: If the “fast-forward” merge is not possible, then Git tries to perform a *three-way merge*.

**Q19:** Which statement is true about the Git merge conflict resolution?

- A) The `git merge status` command shows extensive details about the merge conflict.
- B) If the merge completed successfully, then no further verification is needed.
- C) A merge conflict can either be resolved manually in a text editor or the merge may be rolled back.
- D) Binary files cannot be edited in a text editor, so the only way to resolve a merge conflict for them is to roll back.

**Correct Answer:** C (Sec 1.10b)

- A: Use the `git status` command to find out what went wrong.
- B: Changes in branches may be compatible from Git's perspective but still break the code.
- C: `git checkout --ours` and `git checkout --theirs` allow to handpick a version of the file.

**Q20:** Which statement is true about the `git reset` command?

- A) `git reset` is a safe command; it is always possible to roll back.
- B) `git reset` can be used to clean up commit history and "squash" several commits into one.
- C) `git reset HEAD^2` and `git reset HEAD~2` do the same action.
- D) `git reset --hard filename.txt` may be used to restore a previous version of a single file.

**Correct Answer:** B (Sec 1.10, 1.10c)

- A: The `git reset --hard` command wipes the working area, so use it with care!

C: `HEAD^2` refers to the second parent of the commit, and `HEAD~2` refers to the first parent of the first parent, so they are different. `HEAD^^` and `HEAD~~` are the same, though.

D: `git reset` does not allow per-file operations when used with the `--hard` and `--soft` options.

**Q21: Which statement is true about the `git checkout` command?**

- A) `git checkout` is a safe command; it is always possible to roll back.
- B) Commits can be made in the “Detached HEAD” state, but they would be eventually lost since they won’t belong to any branch.
- C) `git checkout` is a safe command, as it only moves the HEAD pointer but does not change the content of the working area.
- D) `git checkout --hard filename.txt` may be used to restore a previous version of a single file.

**Correct Answer: A (Sec 1.10d)**

- B: Work can be saved by creating a new branch and switching to it with the `git checkout` command, thus returning to the “attached” state.
- C: The working area is updated to match the content of the specified snapshot. Note that this is still considered safe, as the working area content needs to be saved before executing the `git checkout` command, and therefore it can be easily reverted to.
- D: `git checkout` allows per-file operations, but the syntax is `git checkout [branch|commit] <file>`.

**Q22: Which statement is true about the `git revert` command?**

- A) The `git revert HEAD` command erases the latest commit as if it never happened.
- B) Executing `git revert HEAD` twice or `git reset --hard HEAD~2` once produces the same content in the working area.
- C) Unlike `git reset`, the `git revert` command preserves the full commit history of the project.
- D) `git revert filename.txt` may be used to restore a previous version of a single file.

Correct Answer: C (Sec 1.10e)

- A: `git revert` undoes changes made in the latest commit, but they are still present in a history log.
- B: The second `git revert` undoes the effect of the first one, so the content of the working area will be unchanged after two `git revert` commands. `git revert` rolls back two commits.
- D: `git revert` does not allow per-file operations; only the whole commit may be reversed.

Q23: The Python code works on a developer's machine but fails in the client's newly created virtual environment. What could be a reason?

- A) The source code is distributed as a package and installed with pip.
- B) The developer forgot to create a list of library dependencies and include it with the code.
- C) The code is distributed as a text, not a compiled binary.
- D) Python was not copied into a fresh virtual environment when it was created.

Correct Answer: B (Sec 1.11)

- A: Packaging source code and installing it with pip is a standard way of distributing Python source code.

- C: Python is an interpreted language, and its code does not require compilation.
- D: Python executable and standard libraries are automatically copied into a new environment on its creation.

#### Q24: Which statement is true about sequence diagrams?

- A) The horizontal axis shows time progression.
- B) When participants interact with each other, their lifelines intersect.
- C) Synchronous messages are calls to other participants, and a participant's call to itself is called asynchronous.
- D) API calls are visualized as HTTP Request/Response sequences.

#### Correct Answer: D (Sec 1.12)

- A: The vertical axis represents time progression; horizontal lines show message exchanges.
- B: Lifelines are parallel lines; participants use messages, depicted as arrows, for interactions.
- C: Participant's call to itself is called self-message; both synchronous and asynchronous messages are calls to other participants.

## 7.2 Answers for Chapter 2: Using API

#### Q1: Which statement is true about REST API timeout errors?

- A) Timeout handling is only needed for external connections with high latency.
- B) Timeout indicates that the server is down, so it's best to exclude it from subsequent calls.

- C) The backoff timer method uses decreasing (backing off) timeouts between calls.
- D) Setting the timeout value both too low and too high has a negative impact.

**Correct Answer: D (Sec 1.1)**

- A: A timeout may happen with any request over the network, no matter how high the network reliability is or how low is the latency.
- B: The server may be down, but it just may be busy, so the best approach is to retry the connection after some pause.
- C: The backoff timer uses increasing intervals between calls to achieve the best balance between user experience and load on the network.

**Q2: Which statement is true about REST API rate-limiting server response codes?**

- A) The server responds with the HTTP 229 status code to indicate that the request was executed but the next call should be paused.
- B) The server responds with the HTTP 329 status code to indicate a redirect to a new URL that needs to be called to confirm it's not a denial of service.
- C) The server responds with the HTTP 429 status code to indicate to a client that it's making too many requests.
- D) The server responds with the HTTP 529 status code to indicate the server failed to execute a request and it should be retried.

**Correct Answer: C (Sec 1.1)**

- A: The 2xx response codes indicate the success of the operation. The operation that was rate-limited was

- not executed and should be retried, so the 2xx code is not applicable.
- B: The 3xx response codes indicate redirect to a new URL and are not used in the rate-limiting logic.
- D: The 5xx response codes indicate internal server failures, which is not the case with rate-limiting.

**Q3: Which statement is true about timeout and rate-limiting errors handling with REST API?**

- A) Any API call may be safely repeated after a small pause following a timeout or rate-limit error.
- B) Rate-limiting response indicates how long to wait before making the next call. If no such indication is included, a retry can be made immediately.
- C) Using rate-limiting requests from within the application is a proper solution to avoid errors when API restrictions are known and documented.
- D) Requests can be retried as many times as needed until a successful response is received.

**Correct Answer: C (Sec 2.1)**

- A: Calls that create new objects require extra care when repeated, as initial calls could've been executed properly but it was a confirmation response that was lost.
- B: There are no established standards on how to indicate a delay in the response. If no indication is included, some default delay should be assumed (for example, 1 second).
- D: While it's reasonable to retry some requests, it is not advisable to do it too many times and block the application flow. If no response is received after a few attempts, an application should consider the operation as failed and move on.

**Q4: Which statement is true about REST API unrecoverable errors?**

- A) Although not desired, unrecoverable errors are expected and should be properly treated.
- B) Upon receiving an unrecoverable REST API error, program execution should be immediately stopped.
- C) The HTTP 403 “Forbidden” error code indicates the authentication token has expired and the user needs to re-authenticate.
- D) Both recoverable and unrecoverable errors may be handled without user interaction.

**Correct Answer: A (Sec 2.2)**

- B: Although the REST API request has failed, it does not imply the program should terminate. When such errors happen, diagnostic information should be collected and stored, and program execution may continue depending on the application logic.
- C: The HTTP 403 Forbidden code indicates that the user is authenticated and authorized but does not have permission to access the resource. It is an unrecoverable error.
- D: Recoverable errors may be handled by the application logic. Unrecoverable errors typically require user interaction for continuation.

**Q5: Which statement is true about HTTP Cache controls in API calls?**

- A) A stored fresh response can be reused only after validation at the origin server.
- B) If a response is not fresh, it's unusable and should be removed from the cache.
- C) ETag is one of the headers used for cache content freshness validation.

- D) A freshness timer is a faster and more reliable mechanism than content validation.

**Correct Answer: C (Sec 2.3)**

- A: If the stored response is fresh, it may be reused without a validation.
- B: If a cached response is not fresh, it might still be reused after validation, or if the origin is unavailable.
- C: A freshness timer is a faster mechanism because there's no need for the additional requests, but it *assumes* data is still valid, so it's less reliable than content validation, which ensures that the cache is up to date.

**Q6: Which statement is true about HTTP Cache-Control headers?**

- A) Both strong and weak ETags may be used in the “If-None-Match” header in HTTP GET operations.
- B) Both strong and weak ETags may be used in the “If-Match” header in HTTP PUT operations.
- C) The “If-Unmodified-Since” header is used in HTTP GET operations.
- D) The “no-cache” value of the Cache-Control HTTP header disables caching of the returned response.

**Correct Answer: A (Sec 2.3)**

- B: HTTP PUT succeeds only when a strong ETag is used in the “If-Match” header and the resource hasn’t changed (ETags match). If a resource has changed, the server responds with the HTTP status code 412 “Precondition Failed.”
- C: The “If-Modified-Since” header is used in the HTTP GET requests to check resource freshness (the logic is

- similar to “If-None-Match”). “If-Unmodified-Since” is used in HTTP PUT requests (similar to the “If-Match”).
- D: The “no-store” value disables caching; “no-cache” allows caching but indicates that it should always be validated (that is, it’s immediately marked as “not fresh”).

**Q7: Which statement is true about pagination support by REST APIs?**

- A) Cursor-based pagination is a simple and robust way of doing pagination.
- B) Offset/page-based pagination is a simple and robust way of doing pagination.
- C) Pagination improves the user experience at the cost of increased resource utilization.
- D) Pagination links are typically constructed by the application that uses the REST API.

**Correct Answer: A (Sec 2.4)**

- B: It is simple, but it’s prone to page drifts that may cause confusion.
- C: Smaller “paginated” responses consume much less compute and network resources.
- D: APIs provide direct links to pages (for example, “next” or “previous”) in HTTP headers or the response body.

**Q8: Which is *not* a valid actor in the OAuth2 authorization?**

- A) Resource owner
- B) Resource server
- C) Authorization application
- D) Authorization server

**Correct Answer: C (Sec 2.5)**

C: The actors in OAuth flows are the resource owner (the actual user), the resource server (hosts the protected content), the client (the application that tries to access the resource), and the authorization server (verifies identity and issues tokens).

Q9: Which parameter is fixed and always the same in the first (Authorization) client request of the three-legged OAuth2 flow?

- A) response\_type
- B) client\_id
- C) state
- D) redirect\_uri

Correct Answer: A (Sec 2.5). response\_type is always set to “code.”

- B: “client\_id” is the client identifier issued by the authorization server during client registration.
- C: “state” is an arbitrary value used by the client to maintain state between the request and callback.
- D: “redirect\_uri” is a URL on a client where the authorization server will send the user-agent back to once access is granted.

Q10: Which statement is true about the second (Access Token) client request of the three-legged OAuth2 flow?

- A) The “grant\_type” parameter is set to “client\_credentials.”
- B) The client requests an access token by providing the Authorization code received in the previous step.
- C) “redirect\_uri” will be used to supply the newly obtained access token.
- D) The response to this request includes an access token that is valid until the user logs out.

**Correct Answer: B (Sec 2.5)**

- A: “grant\_type” is set to “authorization\_code” for the three-legged OAuth2 authentication.
- C: “redirect\_uri” in this call is used for verification purposes, and it must match “redirect\_uri” from the initial request.
- D: The response includes the access token, its expiration (in seconds), token type (typically, “bearer”), and a refresh token.

## 7.3 Answers for Chapter 3: Cisco Platforms

**Q1: Which are the elements of the Cisco Webex ChatOps application workflow? (Choose three.)**

- A) Create a Cisco Webex chatbot and obtain its access token.
- B) Register a webhook via Cisco Webex API.
- C) Create an application to handle a webhook call from the Cisco Webex API.
- D) Type a message in the Cisco Webex client.

**Correct Answers: A, B, C (Sec 3.1)**

**D: User interactions are not a part of the ChatOps application.**

**Q2: Which statement is true about ChatOps implementation with Cisco Webex API?**

- A) The ChatOps bot is notified about all the messages in all the rooms it belongs to.
- B) The POST method handler is only needed if the ChatOps bot intends to post messages to Webex rooms.
- C) Webhook notification is JSON-formatted.

- D) The webhook call contains full details about the posted message.

**Correct Answer: C (Sec 3.1)**

- A: Webhooks are only called when the bot is specifically mentioned in a room (with the @).
- B: Webhook notification data is sent via HTTP POST, and a response is required to confirm notification delivery; otherwise, the Webex API would eventually disable such a webhook.
- C: Webhook calls only notify that an event has occurred; they do not provide the message content. The webhook handler has to make an additional call to the Webex REST API to get full details.

**Q3: Which statement is true about API interactions with Cisco Firepower Device Manager?**

- A) Username and password are included in headers of every API call for authentication.
- B) When creating objects, the “type” field of the object is reflected in the target URL. For example, the URL for the “networkobject” type would be <https://host/api/fdm/latest/object/networks>.
- C) Configuration changes made via the API are immediately activated, so they should be performed with caution.
- D) The FDM API supports both JSON and XML data structures.

**Correct Answer: B (Sec 3.2)**

- A: The authentication token needs to be obtained first and then included in the “Authorization” header for all the subsequent API calls.

- C: Like changes made via the GUI, API configuration changes need to be “deployed” to be activated.
- D: The FDM API uses JavaScript Object Notation (JSON) format to represent objects.

**Q4: Which action is *not* a part of the SSID management workflow with the Meraki Dashboard API?**

- A) Read a list of accounts and find a correct AccountId.
- B) Read a list of organizations and find a correct OrganizationId.
- C) Read a list of the networks and find a correct NetworkId.
- D) Read a list of the SSIDs and find a correct SSID.

**Correct Answer: A (Sec 3.3A)**

- A: With Meraki, user accounts own organizations. Organizations consist of networks, which then contain their specific settings (for example, SSIDs). There's no such action as “read list of accounts” in the API.

**Q5: Which statement is true about SSID management with the Meraki Dashboard API?**

- A) Username/password authentication is required to obtain an API token that is used for API calls.
- B) SSIDs are created and updated with the HTTP POST operation.
- C) Reading SSID settings with the HTTP GET, changing needed values only (like “name”, “enabled”, and “authMode”), and sending them back with the HTTP PUT operation is a safe way to use Meraki API.
- D) Setting the “Accept” header to the “application/xml” value directs Meraki API to produce XML data structures.

### Correct Answer: C (Sec 3.3A)

- A: The API token is created in the GUI per user account. There's no username/password authentication with the Meraki API.
- B: A predefined number of SSIDs is created by the system, and they can be updated with the HTTP PUT operation.
- C: The Meraki Dashboard API uses JSON format to represent objects.

### Q6: Which statement is true about the Meraki Location API?

- A) In the Scanning API settings, "Secret" is a string that is used by the Meraki cloud to validate the receiving HTTP server.
- B) The API reports both the client's IP and physical MAC address for simplified user tracking.
- C) The API reports both geo-location (latitude and longitude) and map (X/Y) coordinates.
- D) The API needs to be polled at least once a minute per access point to avoid loss of statistics.

### Correct Answer: C (Sec 3.3B)

- A: On the first connection, the Meraki cloud checks that the server returns the "Validator" string as a response to verify the organization's identity. "Secret" is used by your HTTP server to validate posts coming from the Meraki cloud.
- B: For security reasons, the Meraki cloud stores only hashed, salted, and truncated versions of the MAC addresses so that they are not identifiable.
- C: Location information is exported from the Meraki cloud via an HTTP POST of JSON data to a configured destination server, and there's no polling.

**Q7: Which Intersight API query returns a list of compute resources with the ‘Site’ tag set to ‘Rome’?**

- A) GET /api/v1/compute/RackUnits?\$filter=Site eq ‘Rome’
- B) GET /api/v1/compute/RackUnits?  
    \$filter=Tags/any(t:t/Key eq ‘Site’ and t/Value eq  
        ‘Rome’)
- C) GET /api/v1/compute/RackUnits?\$filter=Tags eq ‘Site’  
    and TagValue eq ‘Rome’
- D) GET /api/v1/RackUnits?\$filter=contains(‘Site’,’Rome’)

**Correct Answer: B (Sec 3.4)**

B: Tags are organized as a set of “key”/“value” pairs, so advanced filtering with a Lambda function is the best approach.

**Q8: Which statement is true about API interactions with Cisco UCS?**

- A) Username and password are included in headers of every API call for authentication.
- B) The “configConfMo” configuration method can be used to configure one or many managed objects.
- C) API operations are transactional and rolled back completely in case of any error.
- D) The UCS API supports both JSON and XML data structures.

**Correct Answer: C (Sec 3.5)**

- A: An authentication cookie needs to be obtained first with the username and password, and then it needs to be included in the XML body of subsequent requests as a “cookie” parameter.
- B: configConfMo works with a single MO, whereas configConfMos and configConfMoGroup work with

multiple managed objects.

D: The UCS API works with XML only.

**Q9: Which statement is true about API interactions with Cisco DNA Center?**

- A) Username and password are included in headers of every API call for authentication.
- B) The “/dna/intent/api/v1/site-health?site=HQ” API call returns health information for the HQ site.
- C) The “/dna/intent/api/v1/network-health” API call returns health information about access switches and devices connected to them.
- D) The “/dna/intent/api/v1/client-health” API call returns client health information per client category.

**Correct Answer: D (Sec 3.6)**

- A: Users authenticate with the username and password *before* making API calls to receive the authentication token. The token needs to be included in every request as the “X-Auth-Token” header.
- B: The “site-health” API call returns the overall health information for *all* sites.
- C: The “network-health” API call returns health information by device category (Access, Distribution, Core, Router, or Wireless) and does not include client information.

**Q10: Which statement is true about the capabilities of AppDynamics?**

- A) The AppDynamics architecture consists of agents, controllers, and applications.
- B) AppDynamics automatically discovers business transactions for the application and establishes baselines for them.

- C) The controller oversees agents and sends them details about which parts of the applications to monitor.
- D) The Analytics Events API is a part of the agent-side API group.

**Correct Answer:** B (Sec 3.7)

- A: The AppDynamics architecture consists of agents and a controller.
- C: Agents monitor every line of the code and update the controller in real time.
- D: Controller APIs and the Analytics Events API belong to platform-side APIs.

## 7.4 Answers for Chapter 4: Application Deployment and Security

**Q1:** Which statement is true about CI/CD pipeline failures?

- A) Job failure in a CI/CD stage means that the whole pipeline fails, and it needs to be restarted from scratch after fixing problems.
- B) Test failure should be addressed by the source code review and correction.
- C) Missing dependencies and wrong library versions result in the continuous delivery phase failure.
- D) Running a complete CI/CD pipeline is the most efficient way to test all the source code changes.

**Correct Answer:** A (Sec 4.1)

- B: Test failure often is a result of some error in the application logic; however, it may be caused by the incorrectly written test as well.

- C: Missing dependency, incompatible versions, and failed unit tests break the continuous integration phase.
- D: Developers first should build and test their code locally in an environment as close to the CI/CD environment as possible before committing code to the main repository.

## Q2: Which statement is true about Kubernetes?

- A) Applications should be converted to the Kubernetes format before their deployment.
- B) Kubernetes orchestrates the build and deployment of applications.
- C) Kubernetes provides storage and cache services to applications.
- D) Kubernetes ensures the desired state for the deployed application by creating, removing, and restarting containers as needed.

## Correct Answer: D (Sec 4.2)

- A: There's no such thing as "Kubernetes format." Kubernetes runs containerized applications.
- B: Kubernetes does not deploy source code and does not build applications.
- C: Kubernetes does not provide application-level services (for example, a database), but such components can run *on* Kubernetes.

## Q3: Which is *not* a Kubernetes element?

- A) Kubernetes pod
- B) Kubernetes cluster
- C) Kubernetes container engine
- D) Kubernetes worker node

**Correct Answer: C (Sec 4.2)**

C: Software that runs containers is required for Kubernetes, but it's not a part of it. It could be Docker or any other engine.

**Q4: Which statement is true about Kubernetes objects?**

- A) Kubernetes objects status describes the desired state of the cluster.
- B) Kubernetes Deployment object defines the application and its network parameters.
- C) When Kubernetes detects a mismatch between the specs and the status of the object, it responds by automatically making a required correction.
- D) At the end of the CI/CD pipeline that updates the application to a newer version, Kubernetes objects are also updated, which requires a simultaneous restart of all the application instances on all nodes.

**Correct Answer: C (Sec 4.2)**

- A: Kubernetes *object specs* describe the desired state of the cluster. *Object status* describes the current state of the object.
- B: Network parameters of the application (for example, external IP) are defined in the Service object.
- D: Kubernetes performs *rolling updates*, which incrementally update pod instances one by one with zero downtime.

**Q5: Which statement is true about continuous testing in a CI pipeline?**

- A) Continuous testing is a recurring process that restarts as soon as the previous iteration is complete.
- B) It reduces business risks.

- C) It improves quality but slows down the overall software delivery.
- D) It's a slow process, but it is done in parallel with the software development, so it does not have much of an impact on the delivery timelines.

**Correct Answer: B (Sec 4.3)**

- A: Continuous testing is a part of the CI/CD process and runs when a new pipeline is started.
- C: Continuous testing accelerates software delivery due to the continuous feedback mechanism.
- D: Continuous testing uses automated tests running in parallel, so developers receive feedback very quickly, thus speeding up the delivery.

**Q6: Which statement is *not* true about static code analysis in a CI pipeline?**

- A) It runs a set of test cases against the application code elements.
- B) It can detect security vulnerabilities in the application code.
- C) It can detect bugs in the application code.
- D) It can analyze source syntax and enforce coding standards.

**Correct Answer: A (Sec 4.3)**

- A: Executing tests against units of code (“unit testing”) is done during the dynamic testing. No application code is executed for the static code analysis.

**Q7: Which statement is true about Docker containers?**

- A) Containers share process table and network interfaces with the operating system of the host.

- B) The main components of the Docker Engine are the server, client, and REST API.
- C) OS kernel may be included in a container to allow it to run on a host with a different kernel.
- D) Microservices applications may be easily and effectively managed from the CLI with the “docker” client.

**Correct Answer:** B (Sec 1.9, 4.2, 4.4)

- A: Containers inherit a virtual copy of the process table and network interfaces from the host’s OS.
- C: The kernel of the host’s OS is shared across all the containers that are running on it, so containers do not include it.
- D: The “docker” client provides an easy interface for manual interaction with containers, but it’s not a very scalable method, so some external orchestrator, like Kubernetes, is more preferred in case of complex deployments.

**Q8: Which statement is true about building custom Docker images?**

- A) Docker images are monolithic.
- B) The “FROM” directive may be omitted from the Dockerfile, and “FROM scratch” is assumed then.
- C) The “RUN” directive executes a command in a new layer on top of the current image, with the resulting image used for the next steps.
- D) The “EXPOSE” Dockerfile directive defines the container’s ports that will be open for external connectivity when the container is executed.

**Correct Answer:** C (Sec 4.4)

- A: Docker images consist of several layers that are generated when the commands in the Dockerfile are executed during the Docker image build. Layers may be reused.
- B: A Dockerfile must begin with the FROM instruction. A base image with no parents is created with the explicit “FROM scratch” directive.
- D: “EXPOSE” functions as documented. The `-p <port>` flag with `docker run` actually exposes a port when running the container.

Q9: A container was created and executed as shown next. What would be the result?

[Click here to view code image](#)

```
$ cat Dockerfile
FROM python
ENTRYPOINT ["echo","Blue"]
CMD ["Green"]
$ docker build -t colors . >/dev/null
$ docker run -it colors "Red"
```

- A) The latest Python interpreter is started.
- B) “Blue” is printed.
- C) “Blue Green” is printed.
- D) “Blue Red” is printed.

**Correct Answer: D (Sec 4.4)**

- A: Based on the Python image, the “test” image defines “echo” as a starting program.
- B: This would be correct if no CMD directives were present.
- C: This would be correct if the container was executed without the “Red” command-line argument.

**Q10: Which statement is true about Docker storage and networking?**

- A) By default, container storage is available (read-only) to the other processes on the host machine.
- B) Both non-Docker processes and a Docker container can modify Docker volumes.
- C) The host network driver that uses the host's networking directly is a default driver.
- D) On a user-defined bridge network, containers can resolve each other by name or alias.

**Correct Answer: D (Sec 4.4)**

- A: By default, all data is isolated within a container and is not available outside of it.
- B: This is true for the Bind mounts. Volumes are managed by the Docker, and non-Docker processes should not modify them.
- C: The default driver is the Bridge, which provides isolation between containers connected to different bridges.

**Q11: Which principle is true about applications built according to the “12-factor app” methodology (1)?**

- A) Each deploy (DEV, UAT, and PROD) has its own codebase.
- B) All dependencies (tools and libraries) should be explicitly declared.
- C) Per-deployment variables and credentials may be stored in the environment variables or in the source code.
- D) To be able to use MySQL in the Dev environment and Amazon RDS in production, the database selection logic should be updated within the application.

### Correct Answer: B (Sec 4.5)

- A: Factor I: “One codebase tracked in revision control, many deploys.”
- C: Factor III requires strict separation of config from code as well as storage of the config in environment variables.
- D: Factor IV: Backing services (for example, a database) are treated as attached resources, so one may be replaced with another by changing resource URLs in the config without modifying the app code.

### Q12: Which principle is true about applications built according to the “12-factor app” methodology (2)?

- A) Minor bugfixes may be implemented directly in the production environment.
- B) An application may rely on the external web service to process HTTP requests.
- C) A sudden ungraceful shutdown of a process within an app may cause application instability.
- D) The application does not manage log files. It just writes an unbuffered log stream to stdout.

### Correct Answer: D (Sec 4.5)

- A: Factor V: Strictly separate build and run stages. When an update is required, it’s applied to the source code, and complete delivery is executed.
- B: Factor VIII stipulates that the app should bind to a port and listen for web requests itself.
- C: Factor IX: Maximize robustness with fast startup and graceful shutdown. Processes are disposable and may be stopped at any moment, so they are designed to handle both graceful and ungraceful shutdowns.

**Q13: Which statement is true about the effective logging strategy for an application?**

- A) Distributed logging always includes log collection, forwarding, and indexing.
- B) Each event stream is typically stored in its own separate space.
- C) Using the same correlation ID in all related calls and logging it greatly simplifies troubleshooting.
- D) Logs should be stored as-is.

**Correct Answer: C (Sec 4.6)**

- A: Distributed logging typically has up to five stages: collection, forwarding, storage, and optionally indexing and alerting.
- B: It is best to aggregate the system, event, and other logs to get more context visibility during troubleshooting.
- D: It is best to pre-process log messages and transform them into some common standard before aggregating because different components might have very different logging formats.

**Q14: Which statement is true about data privacy?**

- A) Visiting a website implicitly permits site owners to collect user information.
- B) Pseudonymization irreversibly destroys any way of identifying the data subject.
- C) Anonymization irreversibly destroys any way of identifying the data subject.
- D) Data in transit is more secure because it is encrypted, whereas data at rest is not.

**Correct Answer: C (Sec 4.7)**

- A: As per GDPR, explicit consent is required to collect and use personally identifiable information (PII).
- B: Pseudonymization substitutes one identity with another and allows re-identification (with some additional information).
- D: Both data in transit and data at rest may be encrypted and secured.

**Q15:** Which statement is true about safely storing resource credentials and other secrets?

- A) Storing credentials directly in the application code is acceptable.
- B) Storing credentials directly in the application code is acceptable as soon as they are obfuscated.
- C) Storing credentials in environment variables is acceptable.
- D) Using API gateways is an easy and efficient way to provide secured access.

**Correct Answer:** C (Sec 4.8)

- A: Storing credentials in the application code is insecure because they get exposed to the development team and, potentially, to the public if a project is committed to a public repository.
- B: Obfuscation is reversible, so it's not a very efficient security method.
- D: API gateways are an efficient but complex-to-implement way to secure access to resources.

**Q16:** Which is the correct sequence for the website's identity certificate activation?

- A) The CA generates the system's certificate.
- B) A certificate signing request (CSR) with the identity information and the public key is delivered to a CA.

- C) The certificate is delivered and installed on a system.
- D) The CA validates the request.
- E) The RSA key pair (public and private) is generated.

Correct Answers: E-B-D-A-C (Sec 4.9)

Q17: Which statement is true about PKI (public key infrastructure)?

- A) Remote identity is trusted as soon as the presented certificate is signed by a CA (certificate authority).
- B) If the certificate gets compromised, it may be immediately revoked via the CRL (Certificate Revocation List) process
- C) Certificate keys are used to encrypt data in the TLS-protected session.
- D) TLS provides encryption not only to HTTP but to other upper-level protocols as well.

Correct Answer: D (Sec 4.9)

- A: The certificate should be signed by a *trusted* CA.
- B: CRLs are generated periodically, so they are not always up to date, thus allowing some window of opportunity to an attacker.
- C: Certificate-based asymmetric encryption algorithms are CPU-intensive, so they are used to securely negotiate a symmetric session key, which is then used for data encryption in a TLS session.

Q18: Which statement is true about OWASP threats?

- A) Cross-site scripting (XSS) protection requires all HTML tags to be escaped/removed.
- B) Cross-site request forgery (CSRF) attacks may be mitigated by using session tokens.

- C) An SQL injection attack is a unique type of attack because it exploits specific database system vulnerabilities.
- D) Input blacklisting is the most efficient protection from SQL injection attacks.

**Correct Answer: B (Sec 4.10)**

- A: Sometimes, if some HTML tags are acceptable, then data may be sanitized to allow such tags and strip the rest of them.
- C: SQL injection is one example of an injection attack type, which occurs when untrusted data (for example, user input) is sent to an interpreter as part of a command or query.
- D: Whitelisting is more effective, as obfuscation may allow attackers to bypass blacklisting. Parameterized queries and prepared statements are other efficient SQL injection mitigation techniques

**Q19: Which statement is true about end-to-end encryption (E2EE)?**

- A) Users communicating with each other via a secure connection to an intermediate server is one example of E2EE.
- B) API traffic typically carries non-business-critical data, so it's preferred but not required to have it encrypted.
- C) API traffic within a data center is secure and does not require encryption.
- D) Mutual TLS is more preferred for API calls over regular TLS.

**Correct Answer: D (Sec 4.11)**

- A: An intermediate server decrypts, processes, and encrypts user messages to relay them back, so it has

- access to the user content, and therefore end-to-end encryption is not happening.
- B: API traffic requires secure treatment because it might carry access credentials or sensitive data.
- C: According to some researchers, about half of all security breaches happen within a data center, so traffic encryption within a DC is a valid concern.

## 7.5 Answers for Chapter 5: Infrastructure and Automation

**Q1: Which statement is true about model-driven telemetry (MDT)?**

- A) MDT is similar to the traditional SNMP-based monitoring, just the encoding format is different.
- B) Data streamed from publishers to data collectors is YANG-modeled.
- C) All MDT subscriptions must be added to the running configuration on network devices.
- D) Dial-In MDT sessions automatically reconnect to the receiver after a reload.

**Correct Answer: B (Sec 5.1)**

- A: With MDT, measurements are automatically collected at remote points and transmitted to the receivers for monitoring, and no regular polling, like with SNMP, is required.
- C: Static dial-out subscriptions require an explicit configuration in the device config, but dynamic dial-in do not.
- D: Dial-In sessions are not stored in the device configuration, so all the subscription information is

lost after a reload. Subscriptions need to be re-initiated by the collectors.

## Q2: Which statement is true about YANG data modeling?

- A) YANG structures data models into modules and submodules.
- B) YANG defines four types of data nodes: leaf nodes, leaf-list nodes, instance nodes, and list nodes.
- C) YANG defines data elements types, their structures, and the actual data.
- D) A container node may contain child nodes of any type except container to avoid recursion.

### Correct Answer: A (Sec 5.2)

- B: There are no “instance” nodes; the fourth type is container nodes.
- C: YANG does not define the actual data.
- D: A container node may have child nodes of any type (leafs, lists, containers, leaf-lists, and so on).

## Q3: Which statement is true about RESTCONF?

- A) RESTCONF functionally is very similar to NETCONF, but differences are in the transports and formats.
- B) RESTCONF operations (read/write/delete) are specified in the request’s body.
- C) The resource target of a RESTCONF operation is specified in the request’s body.
- D) Network devices may support more than one YANG model.

### Correct Answer: D (Sec 5.2)

- A: RESTCONF is missing some crucial NETCONF components, such as multiple data stores, commits, rollbacks, and configuration locking.

- B: RESTCONF uses HTTP verbs (GET/POST/PUT/PATCH/DELETE) to define the desired operation.
- C: A resource is specified in the URL when performing a RESTCONF operation.

Q4: Which is the correct RESTCONF base URL for a Layer 2 VLAN 10 configuration on an IOS XE device?

- A) `https://host:port/restconf/data/Cisco-IOS-XE-native:native/interface/Vlan=10/`
- B) `https://host:port/restconf/interface/Cisco-IOS-XE-native:native/data/Vlan=10/`
- C) `https://host:port/restconf/data/Cisco-IOS-XE-native:native/vlan/vlan-list=10/`
- D) `https://host:port/restconf/vlan/Cisco-IOS-XE-native:native/data/vlan-list=10/`

Correct Answer: C (Sec 5.2, Appendix A)

- A: This is the correct URL for the Layer 3 SVI interface (“interface Vlan10”), but not the L2 VLAN definition.
- B: A RESTCONF URL starts with “`https://host:port/restconf/data/`” in IOS XE.
- D: A RESTCONF URL starts with “`https://host:port/restconf/data/`” in IOS XE.

Q5: Which statement is true about configuring network parameters with Ansible?

- A) The “state: overridden” parameter, where supported, may be used to make sure only the Ansible-provided configuration is present on a device.
- B) “ios\_config” is the most comprehensive module to configure Cisco devices, so the other modules are redundant and considered legacy.

- C) The “ios\_command” module allows “show” commands and one-line configuration.
- D) The “state: absent” parameter, where supported, may be used to apply configuration only if it is not already present on a device.

**Correct Answer: A (Sec 5.3a)**

- B: Other modules may provide simpler and/or more effective control over specific configuration parts, so in no way is “ios\_config” the only module to use.
- C: The “ios\_command” module does not support running commands in “configuration” mode.
- D: “state: absent” is used to delete entries from the configuration.

**Q6: Which statement is true about configuring network parameters with Puppet?**

- A) The “mode” parameter determines whether or not the config should be present on a device
- B) The “ensure” parameter indicates whether the provided config is a merge or a replacement of the existing config on a device.
- C) The “target” parameter of the ciscoyang module indicates the RESTCONF URL for the POST operation.
- D) The “source” property of the ciscoyang module contains the model data in YANG JSON or YANG XML NETCONF format or a reference to a local file containing the model data.

**Correct Answer: D (Sec 5.3b)**

- A: The “mode” parameter of the ciscoyang module indicates whether the provided config is a merge or a replacement of the existing config on a device.

- B: The “ensure” parameter determines whether or not the config should be present on a device.
- C: The “target” parameter indicates the model path on a target device or reference to a local file with the same content.

**Q7: Which statement is true about configuration management solutions?**

- A) The imperative approach defines what the eventual target configuration should be.
- B) Configuration management enables the infrastructure as code (IaC) concept.
- C) Orchestration automates the configuration of service elements.
- D) Terraform agents pull their settings from the central server and translate them into the appropriate local configuration.

**Correct Answer: B (Sec 5.4)**

- A: The declarative approach defines what the eventual target configuration should be. The imperative approach defines how the infrastructure is to be changed (specific commands that need to be executed) to meet requirements.
- C: The orchestrator’s function is to arrange and organize the components involved in turning up service. When service is up, it may be configured by the management elements (for example, Ansible or Puppet).
- D: Terraform uses a declarative approach and agentless push model.

**Q8: Which statement is true about application hosting on Cisco IOx-enabled network devices?**

- A) CPU and resources are shared with the main system, so application hosting needs to be used with care.
- B) DNA-Essentials licensing is required for the application hosting.
- C) Extra storage (SSD or external USB) is required for the application hosting.
- D) Only Cisco-signed applications may be hosted on Cisco network devices.

**Correct Answer: C (Sec 5.4)**

- A: Memory, CPU, files access, and disk utilization are restricted and isolated from the main system.
- B: DNA-Advantage licensing is required for the application hosting.
- C: Application hosting is not restricted, as its goal is to provide users a platform for leveraging their own tools and utilities.

**Q9: Which is *not* a step in the application hosting lifecycle on a Cisco IOS XE device?**

- A) Enable IOx subsystem: `Device(config)# iox`
- B) Copy application container image: `Device# copy  
usbflash1:my_iqx_app.tar flash:`
- C) Activate the application: `Device# app-hosting activate  
appid iox_app`
- D) Start the application: `Device# app-hosting start appid  
iox_app`

**Correct Answer: B (Sec 5.5)**

- B: The correct step is to install the application container image: `Device# app-hosting install appid iox_app package  
usbflash1:my_iqx_app.tar`

**Q10: Which statement is *not* true about application hosting networking?**

- A) IP information may only be assigned to the hosted container via IOS XE config.
- B) When connecting externally via the Management interface, the container should be assigned an IP address from the same IP range as the Management interface.
- C) When connecting externally via front panel interfaces, the internal AppGigabitEthernet interface connects applications to the hardware switch fabric and the front panel data ports.
- D) When using the VirtualPortGroup interface, the hosting device becomes the default gateway for the container and routes its traffic.

**Correct Answer: A (Sec 5.5)**

- A: IP information may be assigned via IOS XE config, manually with Linux commands, or via DHCP.

## **Code Snippets**

Many titles include programming code or configuration examples. To optimize the presentation of these elements, view the eBook in single-column, landscape mode and adjust the font size to the smallest setting. In addition to presenting code and configurations in the reflowable text format, we have included images of the code that mimic the presentation found in the print book; therefore, where the reflowable format may compromise the presentation of the code listing, you will see a “Click here to view code image” link. Click the link to view the print-fidelity code image. To return to the previous page viewed, click the Back button on your device or app.

```
$ ls -1 .git/refs/heads/*
.git/refs/heads/bugfix
.git/refs/heads/master

$ cat .git/refs/heads/master
76362436c76d9a40458c67cf3d840263fc924097

$ cat .git/refs/heads/bugfix
2d3d27828c03967384f6b636f87b0db33621abac

$ cat .git/HEAD
ref: refs/heads/master
```

```
$ git merge bugfix
error: Your local changes to the following files would be overwritten by merge:
      file.txt
Please commit your changes or stash them before you merge.
Aborting
```

```
git init
echo -n >file.txt
git add file.txt

echo "Commit #1: common" > file.txt
git commit -am "Common"

git checkout -b bugfix
echo "Commit #2: bugfix branch" >> file.txt
git commit -am "Bugfix branch"

git checkout master
echo "Commit #3: master branch" >> file.txt
git commit -am "Master branch"
```

```
$ git merge bugfix
Auto-merging file.txt
```

CONFLICT (content): Merge conflict in file.txt  
Automatic merge failed; fix conflicts and then commit the result.

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:  file.txt
```

```
$ cat file.txt
Commit #1: common
<<<<< HEAD
Commit #3: master branch
=====
Commit #2: bugfix branch
>>>>> bugfix
```

```
$ cat file.txt
Commit #1: common
Commit #2: merged bugfix branch
Commit #3: merged master branch

$ git add file.txt
$ git commit -m "Merge with the resolved conflict"
[master a452e0d] Merge with the resolved conflict

$ git log --oneline --graph
*   a452e0d (HEAD -> master) Merge with the resolved conflict
|\ \
| * 2d75a4b (bugfix) Bugfix branch
* | 53f53ac Master branch
| /
* 2c0fd4a Common
```

```
$ git checkout --ours file.txt
$ cat file.txt
Commit #1: common
Commit #3: master branch
```

```
$ git checkout --theirs file.txt
$ cat file.txt
Commit #1: common
Commit #2: bugfix branch
```

```
$ git checkout bugfix
error: Your local changes to the following files would be overwritten by
checkout:
      file.txt
Please commit your changes or stash them before you switch branches.
Aborting
```

```
$ echo "One line" >> file.txt
$ git commit -am "First"
$ echo "Second line added" >> file.txt
$ git commit -am "Second"
$ echo "Third line added" >> file.txt
```

```
$ git commit -am "Third"

$ git log --oneline
f656373 Third
fb49f98 Second
a02e20e First

$ cat file.txt
One line
Second line added
Third line added

$ git revert HEAD
[master eb2397f] Revert "Third"
1 file changed, 1 deletion(-)

$ git log --oneline
eb2397f (HEAD -> master) Revert "Third"
f656373 Third
fb49f98 Second
a02e20e First

$ cat file.txt
One line
Second line added
```

```
$ git revert fb49f98
[master 05d4e61] Revert "Second"
 1 file changed, 1 deletion(-)

$ git log --oneline
05d4e61 (HEAD -> master) Revert "Second"
eb2397f Revert "Third"
f656373 Third
fb49f98 Second
a02e20e First

$ cat file.txt
One line
```

```
~/my_project$ python -V
Python 2.7.17
~/my_project$ python3 -m venv devnet
~/my_project$ source devnet/bin/activate
(devnet) ~/my_project$ python -V
Python 3.6.9
(devnet) ~/my_project$ deactivate
~/my_project$
```

```
def API_Read_Safe (url, timeout = 5, attempts = 3):
    """ Read API URL, handle HTTP 429 and timeout errors """

    tries = 0
    while (tries < attempts):
        tries += 1
        try:
            # read target URL
            response = requests.get(url, headers=headers, timeout = timeout)

            # if successful, return the response
            if response.ok:
                print ("Completed successfully")
                return response

            # If response is not "ok", handle error code 429:
            # - wait "Retry-After" seconds or 1 sec, if not present
            # - retry
            if response.status_code == 429:
                try:
                    retry_after = int(response.headers.get('Retry-After'))
                except Exception:
                    retry_after = 1

                print (f'Retry after {retry_after} seconds')
                sleep(retry_after)
                continue

            # if not OK and not 429, then it's something unrecoverable
            print(f'HTTP error: {response.status_code}')
            response.raise_for_status()

        ## if timeout error, retry. If any other error, leave it unhandled
        except requests.exceptions.ConnectTimeout:
            print ("Timeout, retry")
            continue

    # return "None" if unsuccessful after all attempts
    print ("Unable to get any response")
    return None
```

```
Link: <https://example.com/events?offset=100&limit=20>; rel="self",
<https://example.com/events?offset=80&limit=20>; rel="prev",
<https://example.com/events?offset=120&limit=20>; rel="next",
<https://example.com/events?offset=0&limit=20>; rel="first",
<https://example.com/events?offset=220&limit=20>; rel="last"
```

```
import requests

# Use your own access token
bearer_token = 'NjVkMmIx...cae0e10f'

# Use your favorite room ID.
# Find it by calling https://webexapis.com/v1/rooms with proper authentication
room_id="Y2lzY29z...YTI5Zjg5"

messages_per_page = 10
base_url = 'https://webexapis.com/v1'
headers = {
    "Accept": "application/json",
    "Content-Type": "application/json",
    "Authorization": f"Bearer {bearer_token}",
}

def main ():
```

```
#URL to read first messages in the room
url = f'{base_url}/messages?roomId={room_id}&max={messages_per_page}'

while True:
    #read next batch of messages with primitive error checking
    response = requests.get(url, headers=headers, timeout=3)
    response.raise_for_status()

    #iterate through the received messages and print content
    for msg in response.json().get("items"):
        msg_text = msg.get("text") if msg.get("text") else "<image>"
        print (f'Posted by {msg["personEmail"][:20]}: {msg_text[0:60]}')

    # check the content of the "Link" header.
    # if present, use it's value as URL for the next iteration
    # if not, break the loop (header won't be present in the last batch)
    if response.links:
        url=response.links["next"]["url"]
    else:
        break;

if __name__ == "__main__":
    main()
```

```
{
  "links": {
    "self": "https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=6&limit=2"
  },
  "items": [
    {
      "name": "database",
      "id": "32",
      "links": {
        "self": "https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories/32"
      },
      "type": "ApplicationCategory"
    },
    {
      "name": "download manager",
      "id": "45",
      "links": {
        "self": "https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories/45"
      },
      "type": "ApplicationCategory"
    }
  ],
  "paging": {
    "offset": 6,
```

```
    "limit": 2,
    "count": 42,
    "prev": [
        "https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=4&limit=2",
        "https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=2&limit=2",
        "https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=0&limit=2"
    ],
    "next": [
        "https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=8&limit=2",
        "https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=10&limit=2",
        "https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=12&limit=2",
        "https://fmcrestapisandbox.cisco.com/api/fmc_config/v1/domain/e276abec-e0f2-11e3-8169-6d9ed49b625f/object/applicationcategories?offset=14&limit=2"
    ],
    "pages": 21
}
}
```

```
import requests
from auth_token import get_token

page_size = 3

def main():

    # Using FirePower Threat Defense REST API sandbox
    base_url = "https://10.10.20.65/api/fdm/latest"
    token = get_token(base_url)

    headers = {
        "Accept": "application/json",
        "Authorization": f"Bearer {token}",
    }

    #URL to read first object data
    url=f"{base_url}/object/networks"
    while True:
        #read next batch of messages with primitive error checking
        response = requests.get(
            url, headers=headers, params={"limit": page_size}, verify=False
```

```
)  
response.raise_for_status()  
  
body = response.json()  
  
#iterate through the received data and print content  
for net in body["items"]:  
    print(f"Network object: '{net['name']}'"  
          f"type: {net['subType']}, value: {net['value']}")  
  
# check content of the ["paging"] metadata (the "next" entry)  
# if list is present, use its 1st element as URL for the next iteration  
# if not, job is done ("next" entry will be empty in the last batch)  
if body["paging"]["next"]:  
    url=body["paging"]["next"][0]  
else:  
    break  
  
if __name__ == "__main__":  
    main()
```

```
POST /token HTTP/1.1
Host: authorization-server.com

grant_type=client_credentials&
client_id=iuhfvsdz78fs9dujtn35wigybv&
client_secret=859403985utjfk98u389oijrfu89i
```

```
HTTP/1.1 200 OK
{
  "access_token": "2YotnFZFEjr1zCsicMWpAA",
  "token_type": "example",
  "expires_in": 3600,
  "example_parameter": "example_value"
}
```

```
GET https://accounts.google.com/o/oauth2/v2/auth?  
response_type=code&  
scope=https%3A//www.googleapis.com/auth/books?
```

```
client_id=123456789.apps.googleusercontent.com&  
state=random_custom_string&  
redirect_uri=https://oauth.simplereads.com/cb
```

[https://oauth.simplereads.com/cb?code=Spx1OBxzQQYbYS6WxSbIA&state=random\\_custom\\_string&scope=https%3A//www.googleapis.com/auth/books](https://oauth.simplereads.com/cb?code=Spx1OBxzQQYbYS6WxSbIA&state=random_custom_string&scope=https%3A//www.googleapis.com/auth/books)

```
https://oauth.simplereads.com/cb?error=access_denied&  
state=random_custom_string
```

```
POST /token HTTP/1.1
Host: oauth2.googleapis.com

grant_type=authorization_code&
client_id=123456789.apps.googleusercontent.com&
client_secret=your_client_secret&
code=Spx1OBeZQQYbYS6WxSbIA&
redirect_uri=https://oauth.simplereads.com/cb
```

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8

{
  "access_token": "1/fFAGDNJrz1FTz75BzhT3Zn",
  "token_type": "Bearer",
  "expires_in": 3600,
  "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA",
}
```

```
bearer_token = 'ZTZhZmJi...d65305c2ce82'
base_url = 'https://webexapis.com/v1/'
headers = {
    "Authorization": f"Bearer {bearer_token}",
    "Content-Type": "application/json"
}
result = requests.get(f'{base_url}/<function>', headers=headers)
```

```
import requests

base_url = 'https://webexapis.com/v1'
bearer_token = 'ZTZhZmJi..d65305c2ce82'

headers = {
    "Accept": "application/json",
    "Content-Type": "application/json",
    "Authorization": f"Bearer {bearer_token}",
}

webhook = {
    "name": "My Awesome Webhook",
    "targetUrl": "http://377cf7274b2d.ngrok.io/webhook",
    "resource": "messages",
    "event": "created",
}

#read current webhooks and delete them as a cleanup
response = requests.get(f'{base_url}/webhooks', headers=headers, json=webhook)
response.raise_for_status()

#delete them one by one
for item in response.json()["items"]:
    print (f'Deleting webhook \'{item["name"]}\'...')
    del = requests.delete(f'{base_url}/webhooks/{item["id"]}', headers=headers)
    del.raise_for_status()
    print (del.status_code)

#create a new one
response = requests.post(f'{base_url}/webhooks', headers=headers, json=webhook)
response.raise_for_status()

webhook_id = response.json()["id"]
print(f"Webhook for {webhook['targetUrl']} added with ID\n{n{webhook_id}}")
```

```
from flask import Flask, request, json
import requests

app = Flask(__name__)
port = 5010

bearer_token = 'ZTZhZmJi..d65305c2ce82'
base_url = 'https://webexapis.com/v1/'

class Messenger():
    def __init__(self, base_url=base_url, bearer_token=bearer_token):
```

```

        self.base_url = base_url
        self.bearer_token = bearer_token
        self.headers = {
            "Accept": "application/json",
            "Content-Type": "application/json",
            "Authorization": f"Bearer {bearer_token}",
        }

    def get_message(self, message_id):
        received_message_url = f'{self.base_url}/messages/{message_id}'
        message_text = requests.get(received_message_url,
                                     headers=self.headers).json().get('text')
        return (message_text)

    def post_message(self, room_id, message):
        data = {
            "roomId": room_id,
            "text": message,
        }
        post_message_url = f'{self.base_url}/messages'
        post_message = requests.post(post_message_url, headers=self.headers,
                                      data=json.dumps(data))

msg = Messenger()

@app.route('/webhook', methods=['POST'])
def webhook():

    if 'application/json' in request.headers.get('Content-Type'):
        data = request.get_json()

        room_id = data.get('data').get('roomId')
        message_id = data.get('data').get('id')
        message = msg.get_message(message_id)

        reply = f'Bot received message "{message}"'
        msg.post_message(room_id, reply)

        return data
    else:
        return ('Wrong data format', 400)

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=port, debug=True)

```

```
import requests

base_url = "https://10.10.20.65/api/fdm/latest"
auth_url = f"{base_url}/fdm/token"

def get_token():

    headers = {
        "Content-Type": "application/json",
        "Accept": "application/json",
    }

    data = {
        "grant_type": "password",
        "username": "admin",
        "password": "Cisco1234",
    }

    response = requests.post(auth_url, headers=headers, json=data, verify=False)
    response.raise_for_status()

    access_token = response.json()["access_token"]

    return access_token
```

```
{  
    "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE2MDE1...553sTnbJUm329A",  
    "expires_in": 1800,  
    "token_type": "Bearer",  
    "refresh_token": "ehbGciOiJIUzI1NiJSI6kIafAZxUopvfm0I_...CyfwqwYbNEgIlg",  
    "refresh_expires_in": 2400  
}
```

```
import requests

new_net_object = {
    "name": "GOOGLE-DNS",
    "description": "Google public DNS server",
    "subType": "HOST",
    "value": "8.8.8.8",
    "type": "networkobject"
}

class FDM_API:

    def __init__(self, host, username, password):
```

```

""" Initialize FDM/FTD API object """

self.base_url = f"https://{{host}}/api/fdm/latest"
self.username = username
self.password = password
self.headers = {
    "Content-Type": "application/json",
    "Accept": "application/json",
}
self.get_token()

def do_api_call (self, action, url, fdm_object = None):
    """
    Wrapper for API calls to avoid repeating code. Parameters:
    - action: HTTP verb: GET/POST/DELETE etc.
    - url: API-specific part of URL (prefixed with $base_url)
    - fdm_object: data for the POST calls
    Return: dict with the response results
    """

    api_call = requests.request(
        action,
        f"{{self.base_url}}/{{url}}",
        headers = self.headers,
        json = fdm_object,
        verify = False,
        timeout = 3,
    )
    api_call.raise_for_status()

    # debug output
    print (f" >> {action} to {api_call.url} / HTTP {api_call.status_code}")

    # HTTP 204 responses return empty output, .json() would fail with that
    if api_call.text:
        return api_call.json()

def get_token(self):
    """
    Obtain authentication token and use it in future calls """
    # full URL: "https://{{host}}/api/fdm/latest/fdm/token"
    api_path = "fdm/token"
    data = {
        "grant_type": "password",
        "username": self.username,
        "password": self.password,
    }

    response = self.do_api_call ("POST", api_path, data)
    access_token = response.get("access_token")
    self.headers['Authorization'] = f'Bearer {access_token}'

def deploy (self):
    """
    Call "https://{{host}}/api/fdm/latest/operational/deploy"
    to deploy prepared configuration changes
    """

    response = self.do_api_call("POST", "operational/deploy")
    return True

```

```
        return True

def find_object_by_name (obj_name, obj_list):
    """
```

```
Helper function
Find if the object with a given "obj_name" exists in the "obj_list" list
Return: ID of the existing object
"""

for obj in obj_list:
    if obj['name'] == obj['name']:
        return obj['id']
return None

def main():
    # Disable certificate warnings
    requests.packages.urllib3.disable_warnings()

    # Settings for the "Firepower Threat Defense REST API" DevNet sandbox
    fdm=FDM_API("10.10.20.65", "admin", "Cisco1234")

    # read and print current network objects
    fdm_net_objs = fdm.do_api_call("GET", "object/networks")
    for net_obj in fdm_net_objs['items']:
        print(f"Existing {net_obj['name']} object, ID:{net_obj['id']}")

    # Check if object with the same name already exists
    # if so, delete it first to avoid errors
    obj_id = find_object_by_name (new_net_object, fdm_net_objs['items'])
    if obj_id:
        print(f"{new_net_object['name']} object already exists, deleting...")
        fdm.do_api_call ("DELETE", f"object/networks/{obj_id}")

    # Create a new network object
    new_object = fdm.do_api_call ("POST", "object/networks", new_net_object)
    print(f"Created {new_object['name']} object, ID:{new_object['id']}")

    # Configuration changes need to be deployed to be activated
    # Note it will fail in the DevNet sandbox when it's Read-Only
    fdm.deploy()

if __name__ == "__main__":
    main()
```

```
>>> POST to https://10.10.20.65/api/fdm/latest/fdm/token / HTTP 200
>>> GET to https://10.10.20.65/api/fdm/latest/object/networks / HTTP 200
Existing GOOGLE-DNS object, ID:41eb558b-447b-11eb-b176-39fc4b8b65b3
Existing any-ipv4 object, ID:12e56f78-eea2-11ea-baa0-0f39a2dff5dc
Existing any-ipv6 object, ID:12ffd549-eea2-11ea-baa0-8dfd05c8eadf
GOOGLE-DNS object already exists, deleting...
>>> DELETE to https://10.10.20.65/api/fdm/latest/object/networks/41eb558b-447b-11eb-
b176-39fc4b8b65b3 / HTTP 204
>>> POST to https://10.10.20.65/api/fdm/latest/object/networks / HTTP 200
Created GOOGLE-DNS object, ID:a9b130f2-447b-11eb-b176-5d762101a342
>>> POST to https://10.10.20.65/api/fdm/latest/operational/deploy / HTTP 200
```

```
new_url_object = {  
    "name": "Block_badurl.com",  
    "description": "Sample URL object",  
    "url": "badurl.com",
```

```
    "type": "urlobject",
}
```

```
import requests
import json

MY_ORG_NAME = "network"
MY_NET_NAME = "wireless"
MY_SSID_NAME = "devnet"

unknown_ssid = 255

base_url = "https://api.meraki.com/api/v0"

headers = {
    "Accept": "application/json",
    "Content-Type": "application/json",
    "X-Cisco-Meraki-API-Key": "f9...xxx...8a",
}

def do_API_call(api_url, action = "GET", json = None):
    """
    Basic wrapper for API calls. Parameters:
    - resource: API-specific part of URL (prefixed with $base_url)
    - action: HTTP verb: GET/POST/DELETE etc.
    - json: data for PUT/POST calls
    Return: API response results
    """

    api_call = requests.request(
        method = action,
        url = f"{base_url}/{api_url}",
        headers = headers,
        json = json,
        timeout = 3,
    )

    # debug output
    print (f" >> {action} to {api_call.url} / HTTP {api_call.status_code}")

    api_call.raise_for_status()
    if api_call.text:
        return api_call.json()

def find_networkId(org_name, net_name):
    """
    Parameters:
    - org_name: text organization name, may be partial as soon as unique
    - net_name: text network name, may be partial as soon as unique
```

```

Return: ID of the network for API calls, or None if not found
"""

# get a list of all organizations under account
orgs = do_API_call("organizations")

#loop through them to find org_name and its ID
organizationId = 0
for org in orgs:
    if org["name"].lower() == org_name.lower():
        print(f"Found a match for the {org['name']} organization")
        organizationId = org["id"]
        break

# if found, get a list of the networks and
if organizationId:
    nets = do_API_call(f"organizations/{organizationId}/networks")

    # loop through them to find net_name and its ID
    for net in nets:
        print(f"Found a match for the {net['name']} network")
        if net["name"].lower() == net_name.lower():
            return net["id"]

# by default (not found) return None
return None

def find_ssid_nr(networkId, ssid_name):
"""
Return target SSID's ID in the network or "unknown_ssid" if not found
"""

# get a list of all SSIDs for the network
ssids = do_API_call(f"networks/{networkId}/ssids")

# loop through them to find SSID and its ID
for ssid in ssids:
    if ssid["name"].lower() == ssid_name.lower():
        print(f"Found a match for the {ssid['name']} network, "
              f"SSID# {ssid['number']}")
    )
return ssid["number"]

# Use SSID value of 255 to indicate "not found"
return unknown_ssid

def state_name(state):
    """ Return printable SSID state name """
    return "enabled" if state else "disabled"

def main():

    # obtain networkId from the network name
    my_network = find_networkId(MY_ORG_NAME, MY_NET_NAME)
    if not my_network:
        print(f"Network {MY_NET_NAME} not found in the {MY_ORG_NAME} org")
        exit()

    # obtain SSID # from the SSID name

```

```
# obtain SSID # from the SSID name
my_ssid_nr = find_ssid_nr (my_network, MY_SSID_NAME)
if my_ssid_nr == unknown_ssid:
    print (f"SSID {MY_SSID_NAME} not found in the network {MY_NET_NAME}")
```

```
    exit ()

#Read the current state
ssid_state = do_API_call(f"networks/{my_network}/ssids/{my_ssid_nr}")
print (f'Current state is {state_name(ssid_state["enabled"])}')

#print SSID data structure
print (json.dumps(ssid_state, indent=4))

# toggle SSID's "enabled" value
new_ssid_state = not ssid_state["enabled"]

#Update SSID state
update_req = do_API_call(
    api_url = f"networks/{my_network}/ssids/{my_ssid_nr}",
    action = "PUT",
    json = {"enabled": new_ssid_state},
)

#print a returned new state to confirm
print (f'New state is {state_name(update_req["enabled"])}')

if __name__ == "__main__":
    main()
```

```
>>> GET to https://api.meraki.com/api/v0/organizations / HTTP 200
Found a match for the Home Network org
>>> GET to https://api.meraki.com/api/v0/organizations/888888/networks / HTTP 200
Found a match for the Home Network - Wireless network
>>> GET to https://api.meraki.com/api/v0/networks/L_634444590000000000/ssids / HTTP 200
Found a match for the DEVNET-TEST network, SSID# 9
>>> GET to https://api.meraki.com/api/v0/networks/L_634444590000000000/ssids/9 / HTTP 200
Current state is enabled
{
    "number": 9,
    "name": "DEVNET-TEST",
    "enabled": true,
    "splashPage": "None",
    "ssidAdminAccessible": false,
    "authMode": "psk",
    "psk": "WirelessPassword",
    "encryptionMode": "wpa",
    "wpaEncryptionMode": "WPA2 only",
    "ipAssignmentMode": "NAT mode",
    "minBitrate": 11,
    "bandSelection": "Dual band operation",
    "perClientBandwidthLimitUp": 0,
    "perClientBandwidthLimitDown": 0,
    "visible": true,
    "availableOnAllAps": true,
    "availabilityTags": []
}
>>> PUT to https://api.meraki.com/api/v0/networks/L_634444590000000000/ssids/9 / HTTP 200
New state is disabled
```

```
{  
  {  
    "version": "2.1",  
    "secret": "supersecret",  
    "type": "DevicesSeen",  
    "data": {  
      "apMac": "00:18:0a:13:dd:b0",  
      "apFloors": []  
    }  
  }  
}
```

```
"apTags": [
    "dev",
    "entrance",
    "office"
],
"observations": [
    {
        "ipv4": "/192.168.0.38",
        "location": {
            "lat": 52.5355157,
            "lng": -0.06990350000000944,
            "unc": 1.233417960754815,
            "x": [],
            "y": []
        },
        "seenTime": "2016-09-24T00:06:23Z",
        "ssid": ".interwebs",
        "os": null,
        "clientMac": "18:fe:34:fc:ff:ff",
        "name": "Mission Control",
        "seenEpoch": 1474675583,
        "rssi": 47,
        "ipv6": null,
        "manufacturer": "Espressif"
    },
    {
        "ipv4": "/192.168.0.15",
        "location": {
            "lat": 52.5355157,
            "lng": -0.06990350000000944,
            "unc": 1.5497743004111961,
            "x": [],
            "y": []
        },
        "seenTime": "2016-09-24T00:06:40Z",
        "ssid": ".interwebs",
        "os": "Generic Linux",
        "clientMac": "74:da:38:56:ff:ff",
        "name": "Sat-1",
        "seenEpoch": 1474675600,
        "rssi": 47,
        "ipv6": null,
        "manufacturer": "Edimax Technology"
    }
]
}
```

```
GET /api/v1/compute/RackUnits?$filter=Name eq 'WZP211704KM'  
GET /api/v1/compute/RackUnits?$filter=Model ne 'UCSC-C240-M5SN'  
GET /api/v1/RackUnits?$filter=NumCpuCores ge 6 and AvailableMemory gt 65000  
GET /api/v1/RackUnits?$filter=not(Model eq 'HX220C-M5SX' or Model eq 'HX220C-M5S')  
GET /api/v1/RackUnits?$filter=Model in ('HX220C-M5SX', 'UCSC-C240-M5SN')  
GET /api/v1/RackUnits?$filter=contains(Model,'C240')  
GET /api/v1/RackUnits?$filter=endswith(Model,'M5')  
GET /api/v1/aaa/AuditRecords?$filter=CreateTime gt 2020-06-20T08:00:0.000Z
```

```
GET /api/v1/compute/RackUnits?$filter=Tags/any(t:t/Key eq 'Site')
```

```
GET /api/v1/compute/RackUnits?$filter=Tags/any(t:t/Key eq 'Site' and t/Value eq 'London') and Tags/any(t:t/Key eq 'Environment' and t/Value eq 'Production')
```

```
GET /api/v1/compute/RackUnits?$filter/CreateTime gt now() sub P30D
```

```
GET /api/v1/compute/RackUnits?$select=Vendor,Model,Serial
```

```
GET /api/v1/compute/RackUnits?$select=Model,Serial&filter=endswith(Model,'M5')  
&orderby=Model&top=10
```

```
dn = "sys/chassis-5/blade-2/adaptor-1/host-eth-2"
```

```
topRoot
└─sys
    ├─chassis-1
    ├─chassis-2
    └─chassis-3
        ├─blade-1
        │   └─adaptor-1
        └─blade-2
            ├─adaptor-1
            │   └─host-eth-1
            └─adaptor-2
```

```
""  
"sys"  
"sys/chassis-1"  
"sys/chassis-2"  
"sys/chassis-3"  
"sys/chassis-3/blade-1"  
"sys/chassis-3/blade-1/adaptor-1"  
"sys/chassis-3/blade-2"  
"sys/chassis-3/blade-2/adaptor-1"  
"sys/chassis-3/blade-2/adaptor-1/host-eth-1"  
"sys/chassis-3/blade-2/adaptor-2"
```

```
import os
import requests
import xmltodict

class UCS_API:
    def __init__(self, host, username, password):
        self.host = host
        self.username = username
        self.password = password
        self.cookie = None

    def api_request(self, body):
```

```

"""
Wrapper for API calls to avoid repeating code.
Parameter: body is a complete XML request
Return: dict with the response status and result
"""

api_response = requests.post(
    f"http:///{self.host}/nuova",
    headers={"Content-Type": "application/x-www-form-urlencoded"},
    data=body,
)
api_response.raise_for_status()

status = api_response.status_code      # HTTP status
data = xmltodict.parse(api_response.text) # API data as dictionary rather than XML text

return (status, data)

def login(self):
    body = f'<aaaLogin inName="{self.username}" inPassword="{self.password}" />'

    response = self.api_request(body)
    if response[0] == 200:
        self.cookie = response[1]["aaaLogin"]["@outCookie"]
    return self.cookie

def logout(self):
    body = f'<aaaLogout inCookie="{self.cookie}" />'
    self.api_request(body)

def create_server_from_profile(self, name, template):
    body = (
        f'<configConfMo dn="" cookie="{self.cookie}">'
        f'  <inConfig>'
        f'    <lsServer dn="org-root/ls-{name}"'
        f'      name="{name}"'
        f'      srcTemplName="{template}" />'
        f'  </inConfig>'
        f'</configConfMo>'
    )
    response = self.api_request(body)
    return response

if __name__ == "__main__":
    UCS_HOST = os.environ.get('UCS_HOST', '10.10.20.113')
    UCS_USER = os.environ.get('UCS_USER', 'ucspe')
    UCS_PASS = os.environ.get('UCS_HOST', 'ucspe')
    ucs = UCS_API(UCS_HOST, UCS_USER, UCS_PASS)

    # supply template and target instance names
    template = "DevNet_ServiceProfile_Template"
    name = "DevNet_ServiceProfile_Instance"

    response = ucs.create_server_from_profile(name, template)

    resp_status = response[1]["configConfMo"]["outConfig"]["lsServer"].get("@status")
    if response[0] == 200 and resp_status == "created":
        print(f'The service profile {name} created successfully.')
    else:
        print(f'The service profile {name} was not created.')

    ucs.logout()

```

```
import requests
import json

# default values for the Cisco DevNet sandbox
SANDBOX = "sandboxdnac2.cisco.com"
USERNAME = "devnetuser"
PASSWORD = "Cisco123!"

class DNAC_API:

    def __init__(self, host, username, password):
        self.system_url = f"https://{host}/dna/system/api/v1"
        self.intent_url = f"https://{host}/dna/intent/api/v1"

        self.username = username
        self.password = password
        self.headers = {
            "Content-Type": "application/json",
            "Accept": "application/json",
        }

    # Obtain Authentication Token and
    # add it to "headers" for the future API calls
    self.headers["X-Auth-Token"] = self.get_token()

    def get_token(self):
        """ Perform Basic Authentication and obtain API Token """

        auth = (self.username, self.password)
        auth_resp = requests.post(
            f"{self.system_url}/auth/token",
            auth=auth,
            headers=self.headers,
            verify=False
        )
        auth_resp.raise_for_status()
        return auth_resp.json()["Token"]
```

```

def api_read (self, api_url):
    """ Perform "GET" DNA Center API call """

    response = requests.get(
        f"{self.intent_url}/{api_url}",
        headers=self.headers,
        verify=False
    )
    response.raise_for_status()
    return response.json()

def main():

    requests.packages.urllib3.disable_warnings()
    dnac = DNAC_API(SANDBOX, USERNAME, PASSWORD)

    print ("*** Network Wireless Health status ***")
    site_health = dnac.api_read("site-health")['response']
    for site in site_health:
        if site['wirelessDeviceTotalCount']:          # skip empty sites
            print(f"Site: {site['siteName']}:\n"
                  f"  Wireless Network Health: {site['networkHealthWireless']}% "
                  f"({site['wirelessDeviceGoodCount']})/"
                  f"{site['wirelessDeviceTotalCount']} OK) \n"
                  f"  Wireless Client Health: {site['clientHealthWireless']}% "
                  f"for {site['numberOfWirelessClients']} clients"
            )

    print ("\n*** Client Wireless Health status ***")
    client_health = dnac.api_read("client-health")['response'][0]
    for score in client_health["scoreDetail"]:
        print(
            f"Health score for {score['scoreCategory']['value']}: "
            f"{score['scoreValue']}% for {score['clientCount']} clients"
        )
        # drill down into score categories
        scorelist = score.get('scoreList',{})
        for scoreitem in scorelist:
            if scoreitem['clientCount'] > 0:          # skip empty ones
                print(
                    f"  {scoreitem['scoreCategory']['value']} "
                    f": {scoreitem['clientCount']} clients"
                )

if __name__ == "__main__":
    main()

```

```
*** Network Wireless Health status ***
Site: MX14:
    Wireless Network Health: 100% (6/6 OK)
    Wireless Client Health: 39% for 33 clients
Site: All Buildings:
    Wireless Network Health: 100% (11/11 OK)
    Wireless Client Health: 29% for 56 clients
Site: HQ:
    Wireless Network Health: 100% (4/4 OK)
    Wireless Client Health: 0% for 17 clients
Site: System Campus:
    Wireless Network Health: 100% (6/6 OK)
    Wireless Client Health: 39% for 33 clients
Site: All Sites:
```

```
Wireless Network Health: 100% (11/11 OK)
Wireless Client Health: 29% for 56 clients
```

```
*** Client Wireless Health status ***
Health score for ALL: 29% for 82 clients
Health score for WIRED: 100% for 2 clients
  GOOD: 2 clients
Health score for WIRELESS: 28% for 80 clients
  FAIR: 58 clients
  GOOD: 22 clients
```

```
$ python app_test.py
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
ModuleNotFoundError: No module named 'requests'
ERROR: Job failed: exit status 1
```

```
TypeError: my_module() missing 1 required positional argument: 'method'  
AttributeError: module 'my_module' has no attribute 'APIv2'  
NameError: name 'ENV_PASS' is not defined  
...
```

```
=====
FAIL: test_even (__main__.NumbersTest) (i=1)
-----
Traceback (most recent call last):
  File "subtests.py", line 32, in test_even
    self.assertEqual(i % 2, 0)
AssertionError: 1 != 0
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: devcor-app
spec:
  replicas: 3 # tells deployment to run 3 pods matching the template
  selector:
    matchLabels:
      app: devcor-app
  template:
    metadata:
      labels:
        app: devcor-app
    spec:
      containers:
        - name: devcor-container
          image: devcor-container:latest
```

```
  ports:  
    - containerPort: 5000
```

```
apiVersion: v1
kind: Service
metadata:
  name: devcor-service
spec:
  selector:
    app: devcor-app
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 5000
  externalIPs:
    - 192.0.2.100
```

```
$ docker run -it python
Python 3.8.5 (default, Sep  1 2020, 18:44:24)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("Hello, docker!")
```

```
Hello, docker!  
>>>
```

```
$ docker run -it python bash
root@c53f5f79a8b1:~# echo "Hello, bash!"
Hello, bash!
```

```
$ docker ps
CONTAINER ID        IMAGE       COMMAND      CREATED     STATUS      PORTS          NAMES
88c7dd2c5604        python      "python3"    5 seconds ago   Up 4 seconds
a7d3709d880b        httpd      "httpd-foreground" 27 seconds ago   Up 25 seconds   0.0.0.0:8080->80/tcp   stoic_cartwright
```

```
$ docker run -it python
Unable to find image 'python:latest' locally
latest: Pulling from library/python
d6ff36c9ec48: Pull complete
c958d65b3090: Pull complete
edaf0a6b092f: Pull complete
80931cf68816: Pull complete
7dc5581457b1: Pull complete
87013dc371d5: Pull complete
dbb5b2d86fe3: Pull complete
4cb6f1e38c2d: Pull complete
c2df8846f270: Pull complete
Digest: sha256:bc765f71aaa90648de6cfa356ec201d50549031a244f48f8f477f386517c5d1b
Status: Downloaded newer image for python:latest
Python 3.8.5 (default, Sep 1 2020, 18:44:24)
```

IMAGE	CREATED	CREATED BY	SIZE
a7cda474cef4	3 days ago	/bin/sh -c #(nop) CMD ["python3"]	0B
<missing>	3 days ago	/bin/sh -c set -ex; wget -O get-pip.py "\$P..."	7.24MB
<missing>	3 days ago	/bin/sh -c #(nop) ENV PYTHON_GET_PIP_SHA256...	0B
<missing>	3 days ago	/bin/sh -c #(nop) ENV PYTHON_GET_PIP_URL=ht...	0B
<missing>	3 days ago	/bin/sh -c cd /usr/local/bin && ln -s idle3...	32B
<missing>	3 days ago	/bin/sh -c set -ex && wget -O python.tar.x...	53MB
<missing>	3 days ago	/bin/sh -c #(nop) ENV PYTHON_VERSION=3.8.5	0B
<missing>	3 days ago	/bin/sh -c #(nop) ENV GPG_KEY=E3FF2839C048B...	0B
<missing>	3 days ago	/bin/sh -c apt-get update && apt-get install...	17.9MB
<missing>	3 days ago	/bin/sh -c #(nop) ENV LANG=C.UTF-8	0B
<missing>	3 days ago	/bin/sh -c #(nop) ENV PATH=/usr/local/bin:/...	0B
<missing>	3 days ago	/bin/sh -c set -ex; apt-get update; apt-ge...	510MB
<missing>	4 weeks ago	/bin/sh -c apt-get update && apt-get install...	146MB
<missing>	4 weeks ago	/bin/sh -c set -ex; if ! command -v gpg > /...	17.5MB
<missing>	4 weeks ago	/bin/sh -c apt-get update && apt-get install...	16.5MB
<missing>	4 weeks ago	/bin/sh -c #(nop) CMD ["bash"]	0B
<missing>	4 weeks ago	/bin/sh -c #(nop) ADD file:4b03b5f551e3fbdf4...	114MB

```
docker build -t devnet/sample -f ~/Dockerfile <context>
```

```
FROM python:3.7

ENV CONTROLLER 10.1.1.100
ENV USER_NAME dev_admin

WORKDIR /usr/src/app

COPY package.json ./packages
COPY . .

RUN pip install -r requirements.txt

EXPOSE 8080

ENTRYPOINT [ "python3", "-E"]
CMD [ "app.py" ]
```

```
import logging
logger = logging.getLogger(app_name)
logger.setLevel(logging.INFO)
logger.info(f'Starting {app_name} in a {run_mode} mode')
```

```
ACCESS_USER="admin"
ACCESS_SECRET = "my_own_password"
result1= API_call (url, user=ACCESS_USER, password=ACCESS_SECRET)
result2= API_call (url, user="admin", password="my_own_password")
```

```
ACCESS_USER=os.environ["ENV_ACCESS_USER"]
ACCESS_SECRET=os.environ["ENV_ACCESS_SECRET"]
result= API_call (url, user=ACCESS_USER, password=ACCESS_SECRET)
```

```
DEVICE_IP="10.1.1.250"
query=f"SELECT username, password FROM secrets WHERE deviceIP='{DEVICE_IP}'"

db = mysql.connect(host=db_host, user="dba", passwd="dbpass", database="my_db")
cursor = db.cursor()
cursor.execute(query)
ACCESS_USER=cursor["username"]
ACCESS_SECRET=cursor["password"]

result= API_call (url, user=ACCESS_USER, password=ACCESS_SECRET)
```

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      05:57:c8:0b:28:26:83:a1:7b:0a:11:44:93:29:6b:79
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert
SHA2 High Assurance Server CA
    Validity
      Not Before: May 5 00:00:00 2020 GMT
      Not After : May 10 12:00:00 2022 GMT
    Subject: C = US, ST = California, L = San Francisco, O = "GitHub,
Inc.", CN = github.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public-Key: (2048 bit)
      Modulus:
        00:bb:32:b4:d0:d8:9e:9a:9e:8c:79:29:4c:2b:a8:
        [..... removed for brevity ..]
        18:b5
      Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Authority Key Identifier:
        keyid:51:68:FF:90:AF:02:07:75:3C:CC:D9:65:64:62:A2:12:B8:59:72:3B
        X509v3 Subject Key Identifier:
          63:02:D2:5D:02:5F:F7:8D:D5:5A:12:9E:76:11:36:96:86:2C:8A:48
        X509v3 Subject Alternative Name:
          DNS:github.com, DNS:www.github.com
        X509v3 Key Usage: critical
          Digital Signature, Key Encipherment
        X509v3 Extended Key Usage:
          TLS Web Server Authentication, TLS Web Client Authentication
        X509v3 CRL Distribution Points:
        Full Name:
```

URI:<http://crl3.digicert.com/sha2-ha-server-g6.crl>

Full Name:

URI:<http://crl4.digicert.com/sha2-ha-server-g6.crl>

X509v3 Certificate Policies:

Policy: 2.16.840.1.114412.1.1

CPS: <https://www.digicert.com/CPS>

Policy: 2.23.140.1.2.2

Authority Information Access:

OCSP - URI:<http://ocsp.digicert.com>

CA Issuers -

URI:<http://cacerts.digicert.com/DigiCertSHA2HighAssuranceServerCA.crt>

X509v3 Basic Constraints: critical

CA:FALSE

**Signature Algorithm: sha256WithRSAEncryption**

86:32:8f:9c:15:b8:af:e8:d1:de:08:3a:44:0e:71:20:24:d6:

[..... removed for brevity .....

27:4a:7a:49

```
import html

print (html.escape ("<script type='application/javascript'>"))
<script type="application/javascript">
```

```
<a href="http://bank.com/transfer.do?acct=MARIA&amount=1000">My Pictures!</a>
```

```
statement = "SELECT * FROM users WHERE name = '" + userName + "';"
```

```
$ cat Dockerfile
FROM python
ENTRYPOINT ["echo","Blue"]
CMD ["Green"]
$ docker build -t colors . >/dev/null
$ docker run -it colors "Red"
```

```
MDT#show running-config | sec tele
telemetry ietf subscription 100
encoding encode-kvvpb
filter xpath /process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/five-seconds
source-address 192.168.2.100
source-vrf Mgmt-vrf
stream yang-push
update-policy periodic 500
receiver ip address 172.16.10.10 57000 protocol grpc-tcp
```

```
leaf host-name {  
    type string;  
    description  
        "Hostname for this system.";  
}
```

```
<host-name>my.example.com</host-name>
```

```
leaf-list domain-search {  
    type string;  
    description  
        "List of domain names to search."  
}
```

```
<domain-search>ns1.example.com</domain-search>
<domain-search>ns2.example.com</domain-search>
<domain-search>ns3.example.com</domain-search>
```

```
container system {
    container login {
        leaf message {
            type string;
            description
                "Message given at start of login session.";
        }
    }
}
```

```
<system>
  <login>
```

```
<message>Good morning</message>
</login>
</system>
```

```
list user {  
    key "name";  
    leaf name {  
        type string;  
    }  
    leaf full-name {  
        type string;  
    }  
    leaf class {  
        type string;  
    }  
}
```

```
<user>
  <name>glocks</name>
  <full-name>Goldie Locks</full-name>
  <class>intruder</class>
</user>
<user>
  <name>snowey</name>
  <full-name>Snow White</full-name>
  <class>free-loader</class>
</user>
<user>
  <name>rzell</name>
  <full-name>Rapun Zell</full-name>
  <class>tower</class>
</user>
```

```
import ietf-yang-types { prefix yang; }
leaf birthday { type yang:date-and-time; }
```

```
typedef percent {
    type uint8 {
        range "0 .. 100";
    }
}
```

```
leaf completed {  
    type percent;  
}
```

```
CAT9K(config)#aaa new-model
CAT9K(config)#aaa authentication login default local
CAT9K(config)#aaa authorization exec default local
CAT9K(config)#username admin privilege 15 secret cisco
CAT9K(config)#ip http secure-server
CAT9K(config)#ip http authentication local
CAT9K(config)#restconf

CAT9K#show platform software yang-management process
confd          : Running
nesd           : Running
syncfd         : Running
ncsshd         : Not Running
dmiauthd       : Running
nginx          : Running
ndbmand        : Running
pubd           : Running
gnmib          : Not Running
```

```
import requests
import json

requests.packages.urllib3.disable_warnings()

host="192.168.2.4"
port="443"
username="admin"
password="cisco"

base = f"https://{{host}}:{{port}}/restconf/data"

auth = requests.auth.HTTPBasicAuth(username, password)
headers = {
    'Content-Type': 'application/yang-data+json',
    'Accept': 'application/yang-data+json',
}

def restconf_read (url):
```

```
r = requests.request("GET", url, auth=auth, headers=headers, verify=False)
print(f"GET result code: {r.status_code}")
if r.status_code >= 400:
    print(f">>Error: {r.json()['errors']['error'][0]['error-message']} ")
else:
    print(r.text)

def restconf_write (url, method, write_data):
    r = requests.request(method, url,
        auth=auth, headers=headers, data=json.dumps(write_data), verify=False)
    print(f"Write ({method}) result code: {r.status_code}")
    if r.status_code >= 400:
        print(f">>Error: {r.json()['errors']['error'][0]['error-message']} ")

def restconf_delete (url):
    r = requests.request("DELETE",url,auth=auth,headers=headers,verify=False)
    print(f"DELETE result code: {r.status_code}")
    if r.status_code >= 400:
        print(f">>Error: {r.json()['errors']['error'][0]['error-message']} ")
```

```
# predefine URLs to avoid repetition

# parent leaf for all interfaces configuration
url_interfaces = f"{base}/Cisco-IOS-XE-native:interface/"

# specific leaf for the Vlan10 configuration
url_vlan10 = f"{base}/Cisco-IOS-XE-native:interface/Vlan=10/"

# specific leaf for the Vlan10 IP address configuration
url_vlan10_ip = f"{base}/Cisco-IOS-XE-native:interface/Vlan=10/" \
    "ip/address/primary"

# Vlan10 definition
vlan10_data={
    "Vlan": {
        "name": "10",
        "description": "DevNet Vlan SVI interface",
        "ip": {
            "address": {
                "primary": {
                    "address": "10.10.0.1",
                    "mask": "255.255.255.0"
                }
            }
        }
    }
}

# new IPs for the interface
vlan10_ip_1 = {
    "primary" : {
```

```

        "address": "10.10.0.10",
        "mask": "255.255.255.0"
    }
}

vlan10_ip_2 = {
    "primary" : {
        "address": "10.10.0.20",
        "mask": "255.255.255.0"
    }
}

# first, DELETE Vlan10 interface for a fresh start. Errors may be ignored
restconf_delete (url_vlan10)
DELETE result code: 204

# Create and configure the interface in a single call with POST call
restconf_write (url_interfaces, "POST", vlan10_data)
Write (POST) result code: 201

# POST is not an idempotent operation! Repeat it and it will fail
restconf_write (url_interfaces, "POST", vlan10_data)
Write (POST) result code: 409
>>Error: object already exists: /ios:native/ios:interface/ios:Vlan[ios:name='10']

# Read interface configuration to confirm it was successfully created
restconf_read (url_vlan10)
GET result code: 200
{
    "Cisco-IOS-XE-native:Vlan": {
        "name": 10,
        "description": "DevNet Vlan SVI interface",
        "ip": {
            "primary": {
                "address": {
                    "primary": {
                        "address": "10.10.0.1",
                        "mask": "255.255.255.0"
                    }
                }
            }
        }
    }
}

# Now change the IP address on the interface with the PATCH call
restconf_write (url_vlan10_ip, "PATCH", vlan10_ip_1)
Write (PATCH) result code: 204

# Note the changed IP
restconf_read (url_vlan10_ip)
GET result code: 200
{
    "Cisco-IOS-XE-native:primary": {
        "address": "10.10.0.10",
        "mask": "255.255.255.0"
    }
}

# Change the IP address on the interface with the PUT call.
restconf_write (url_vlan10_ip, "PUT", vlan10_ip_2)
Write (PUT) result code: 204

restconf_read (url_vlan10_ip)
GET result code: 200

```

```
{  
  "Cisco-IOS-XE-native:primary": {  
    "address": "10.10.0.20",  
    "mask": "255.255.255.0"
```

}

```
interface GigabitEthernet1/0/2
switchport access vlan 10
switchport mode access
switchport nonegotiate
channel-group 10 mode active
lacp rate fast
```

```
url_interface = f"{base}/Cisco-IOS-XE-native:native/interface/" \
                 f"GigabitEthernet=1%2F0%2F2/"
restconf_read(url_interface)

GET result code: 200
{
    "Cisco-IOS-XE-native:GigabitEthernet": {
        "name": "1/0/2",
        "switchport": {
            "Cisco-IOS-XE-switch:access": {
                "vlan": {
                    "vlan": 10
                }
            },
            "Cisco-IOS-XE-switch:mode": {
                "access": {
                }
            },
            "Cisco-IOS-XE-switch:nonegotiate": [null]
        },
        "Cisco-IOS-XE-ethernet:channel-group": {
            "number": 10,
            "mode": "active"
        },
        "Cisco-IOS-XE-ethernet:lACP": {
            "rate": "fast"
        }
    }
}
```

```
# predefine URLs to avoid repetition

# parent leaf for all VLAN configuration
url_vlan=f"{base}/Cisco-IOS-XE-native:native/vlan/"

# specific leaf for the Vlan10 name configuration
url_vlan10_name=f"{base}/Cisco-IOS-XE-native:native/vlan/vlan-list=10/name/"
```

```

# specific leaf for the Vlan11 configuration
url_vlan11=f"{base}/Cisco-IOS-XE-native:vlan/vlan-list=11/"

# initial set of VLANs
base_vlans = {
    "Cisco-IOS-XE-native:vlan": {
        "Cisco-IOS-XE-vlan:vlan-list": [
            {
                "id": 10,
                "name": "DEVNET-10"
            },
            {
                "id": 11,
                "name": "DEVNET-11"
            },
        ]
    }
}

# additional VLAN to be created
more_vlans = {
    "Cisco-IOS-XE-vlan:vlan-list": [
        {
            "id": 99,
            "name": "DEVNET-99"
        },
    ]
}

# new name for VLAN10
vlan10_newname = { "Cisco-IOS-XE-vlan:name": "RESTCONF_Patched_VLAN" }

# fresh start
# PUT command will replace any existing VLAN database with a new one
restconf_write (url_vlan, "PUT", base_vlans)
Write (PUT) result code: 204

# Read VLAN configuration to confirm it was successfully created
restconf_read (url_vlan)
GET result code: 200
{
    "Cisco-IOS-XE-native:vlan": {
        "Cisco-IOS-XE-vlan:vlan-list": [
            {
                "id": 10,
                "name": "DEVNET-10"
            },
            {
                "id": 11,
                "name": "DEVNET-11"
            },
        ]
    }
}

# add one more VLAN with the POST command
restconf_write (url_vlan, "POST", more_vlans)
Write (POST) result code: 201

```

```
# Read VLAN configuration to confirm
restconf_read (url_vlan)
GET result code: 200
```

```

{
  "Cisco-IOS-XE-native:vlan": {
    "Cisco-IOS-XE-vlan:vlan-list": [
      {
        "id": 10,
        "name": "DEVNET-10"
      },
      {
        "id": 11,
        "name": "DEVNET-11"
      },
      {
        "id": 99,
        "name": "DEVNET-99"
      }
    ]
  }
}

# delete one VLAN we don't need
restconf_delete (url_vlan11)
DELETE result code: 204

# Read VLAN configuration to confirm
restconf_read (url_vlan)
GET result code: 200
{
  "Cisco-IOS-XE-native:vlan": {
    "Cisco-IOS-XE-vlan:vlan-list": [
      {
        "id": 10,
        "name": "DEVNET-10"
      },
      {
        "id": 99,
        "name": "DEVNET-99"
      }
    ]
  }
}

# change the name of VLAN 10
restconf_write (url_vlan10_name, "PATCH", vlan10_newname)
Write (PATCH) result code: 204

restconf_read (url_vlan)
GET result code: 200
{
  "Cisco-IOS-XE-native:vlan": {
    "Cisco-IOS-XE-vlan:vlan-list": [
      {
        "id": 10,
        "name": "RESTCONF-Patched_VLAN"
      },
      {
        "id": 99,
        "name": "DEVNET-99"
      }
    ]
  }
}

```

} <sup>J</sup>

```

# predefine URLs to avoid repetition

# parent leaf for all VLAN configuration
url_route=f"{base}/Cisco-IOS-XE-native:native/ip/route/"

# read only prefix and route name
url_route_filtered=f"{base}/Cisco-IOS-XE-native:native/ip/route/" \
    "ip-route-interface-forwarding-list?fields=prefix;fwd-list/name"

# specific leaf for the route name configuration
url_route10_name = f"{base}/Cisco-IOS-XE-native:native/" \
    "ip/route/ip-route-interface-forwarding-list=10.10.0.0,255.255.0.0/" \
    "fwd-list=192.168.2.1/name/"

# specific leaf for the route configuration
url_route11 = f"{base}/Cisco-IOS-XE-native:native/" \
    "ip/route/ip-route-interface-forwarding-list=10.11.0.0,255.255.0.0/"

# initial set of routes
base_routes = {
    "Cisco-IOS-XE-native:route": [
        {
            "ip-route-interface-forwarding-list": [
                {
                    "prefix": "10.10.0.0",
                    "mask": "255.255.0.0",
                    "fwd-list": [
                        {
                            "fwd": "192.168.2.1",
                            "name": "test",
                        }
                    ]
                },
                {
                    "prefix": "10.11.0.0",
                    "mask": "255.255.0.0",
                    "fwd-list": [
                        {
                            "fwd": "192.168.2.1"
                        }
                    ]
                }
            ]
        }
    ]
}

# additional route to be created
more_routes = {
    "Cisco-IOS-XE-native:ip-route-interface-forwarding-list": [
        {
            "prefix": "10.20.0.0",
            "mask": "255.255.255.0",
        }
    ]
}

```

```

        "fwd-list": [
            {
                "fwd": "192.168.2.254"
            }
        ]
    }
}

# new name for route
route10_newname = { "name": "DevNet_route" }

# fresh start
# PUT command will REPLACE any existing static routes with new ones
restconf_write (url_route, "PUT", base_routes)
Write (PUT) result code: 204

# Read static routes to confirm, filter data
restconf_read (url_route_filtered)
GET result code: 200
{
    "Cisco-IOS-XE-native:ip-route-interface-forwarding-list": [
        {
            "prefix": "10.10.0.0",
            "fwd-list": [
                {
                    "name": "test"
                }
            ],
            {
                "prefix": "10.11.0.0"
            }
        ]
    }
}

# add one more route with the POST command
restconf_write (url_route, "POST", more_routes)
Write (POST) result code: 201

# Read static routes to confirm
restconf_read (url_route_filtered)
GET result code: 200
{
    "Cisco-IOS-XE-native:ip-route-interface-forwarding-list": [
        {
            "prefix": "10.10.0.0",
            "fwd-list": [
                {
                    "name": "test"
                }
            ],
            {
                "prefix": "10.11.0.0"
            },
            {
                "prefix": "10.20.0.0"
            }
        }
    ]
}

```

```
    }

# delete one route we don't need
```

```
restconf_delete (url_route11)
DELETE result code: 204

# Read static routes to confirm
restconf_read (url_route_filtered)
GET result code: 200
{
    "Cisco-IOS-XE-native:ip-route-interface-forwarding-list": [
        {
            "prefix": "10.10.0.0",
            "fwd-list": [
                {
                    "name": "test"
                }
            ],
            {
                "prefix": "10.20.0.0"
            }
        ]
    }
}

# change the name of the 10.10.0.0 route
restconf_write (url_route10_name, "PATCH", route10_newname)
Write (PATCH) result code: 204

# Read full static routes information
restconf_read (url_route)
GET result code: 200
{
    "Cisco-IOS-XE-native:route": [
        "ip-route-interface-forwarding-list": [
            {
                "prefix": "10.10.0.0",
                "mask": "255.255.0.0",
                "fwd-list": [
                    {
                        "fwd": "192.168.2.1",
                        "name": "DevNet_route"
                    }
                ],
                {
                    "prefix": "10.20.0.0",
                    "mask": "255.255.255.0",
                    "fwd-list": [
                        {
                            "fwd": "192.168.2.254"
                        }
                    ]
                }
            }
        ]
    }
}
```

```
url_route = \
f'{base}/ietf-routing:routing/routing-instance=default/routing-protocols/" \
f"routing-protocol=static,1/static-routes/ipv4"

restconf_read (url_route)
GET result code: 200
{
    "ietf-ipv4-unicast-routing:ipv4": {
        "route": [
            {
                "destination-prefix": "10.10.0.0/16",
                "next-hop": {
                    "next-hop-address": "192.168.2.1"
                }
            },
            {
                "destination-prefix": "10.20.0.0/24",
                "next-hop": {
                    "next-hop-address": "192.168.2.254"
                }
            }
        ]
    }
}
```

```
tasks:
  - name: run multiple commands on remote nodes
    ios_command:
      commands:
        - show version
        - show interfaces
```

```
tasks:
  - name: configure top level configuration
    ios_config:
      lines: hostname {{ inventory_hostname }}
```

```
- name: configure interface settings
  ios_config:
    lines:
      - description test interface
      - ip address 172.31.1.1 255.255.255.0
    parents: interface Ethernet1

- name: load new acl into device
  ios_config:
    parents: ip access-list extended ACL-IN
    lines:
      - 10 permit ip host 1.1.1.1 any log
      - 20 permit ip host 2.2.2.2 any log
      - 30 permit ip host 3.3.3.3 any log
    before:
      - interface GigabitEthernet0/1
      - no ip access-group ACL-IN in
      - no ip access-list extended ACL-IN
    after:
      - interface GigabitEthernet0/1
      - ip access-group ACL-IN in
  match: exact
  replace: block

- name: save running to startup when modified
  ios_config:
    save_when: modified
```

```
tasks:
  - name: configure black hole in vrf blue depending on tracked item 10
    ios_static_route:
      prefix: 192.168.2.0
      mask: 255.255.255.0
      vrf: blue
      interface: null0
      track: 10

  - name: remove route
    ios_static_route:
      prefix: 192.168.2.0
      mask: 255.255.255.0
      next_hop: 10.0.0.1
      state: absent
```

```
tasks:
  - name: Merge provided configuration with device configuration
    ios_vlans:
      config:
        - name: Vlan_10
          vlan_id: 10
          state: active
          shutdown: disabled
        - name: Vlan_20
          vlan_id: 20
          state: active
          shutdown: enabled
```

```
state: merged

- name: Override all VLANs with the provided configuration
  ios_vlans:
    config:
      - name: Vlan_10
        vlan_id: 10
  state: overridden
```

```
tasks:
  - name: configure a vrf named management
    ios_vrf:
      name: management
      description: oob mgmt vrf
      interfaces:
        - Management1

  - name: remove a vrf named test
    ios_vrf:
      name: test
      state: absent

  - name: configure a set of VRFs and purge any others
    ios_vrf:
      vrf:
        - red
        - blue
        - green
      purge: yes
```

```
$ cat hosts.yml
[cisco]
cat9k ansible_host=192.168.2.4

[cisco:vars]
ansible_user=admin
ansible_password=cisco
ansible_connection=network_cli
ansible_network_os=ios

$ cat cdp-descr.yml
---
- name: Sample Ansible play
  hosts: cisco
  gather_facts: no

  tasks:
    - name: Obtain Cisco device information
      ios_facts:
        gather_subset:
          - interfaces
```

```

- name: Display provisioned device name
  debug:
    msg: "Working on: {{ ansible_net_hostname }}"

- name: Update interface description based on CDP/LLDP info
  ios_config:
    lines:
      - description Link to {{ item.value[0].port }} on {{item.value[0].host }}
        parents: interface {{ item.key }}
        save_when: changed
    with_dict: "{{ansible_facts.net_neighbors}}"

- name: save running to startup when modified
  ios_config:
    save_when: modified

$ ansible-playbook cdp-descr.yml

PLAY [Sample Ansible play]
*****
TASK [Obtain Cisco device information]
*****
ok: [cat9k]

TASK [Display provisioned device name]
*****
ok: [cat9k] => {
    "msg": "Working on: CAT9K-01"
}

TASK [Update interface description based on CDP/LLDP info]
*****
changed: [cat9k] => (item={'key': 'GigabitEthernet1/0/23', 'value': [{'host': 'CAT9K-04', 'platform': 'cisco C9200-24P', 'port': 'GigabitEthernet1/0/24'}]})
changed: [cat9k] => (item={'key': 'GigabitEthernet1/0/24', 'value': [{'host': 'CAT9K-05', 'platform': 'cisco C9200-24P', 'port': 'GigabitEthernet1/0/24'}]})
changed: [cat9k] => (item={'key': 'GigabitEthernet1/0/21', 'value': [{'host': 'CAT9K-02', 'platform': 'cisco C9200-24P', 'port': 'GigabitEthernet1/0/24'}]})
changed: [cat9k] => (item={'key': 'GigabitEthernet1/0/22', 'value': [{'host': 'CAT9K-03', 'platform': 'cisco C9200-24P', 'port': 'GigabitEthernet1/0/24'}]})
[WARNING]: To ensure idempotency and correct diff the input configuration lines
should be similar to how they appear if present in the running configuration on
device

TASK [save running to startup when modified]
*****
changed: [cat9k]

PLAY RECAP
*****
***** cat9k : ok=4    changed=2    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0

```

```
CAT9K-01#show run | b net1/0/21
interface GigabitEthernet1/0/21
  description Link to GigabitEthernet1/0/24 on CAT9K-02
!
interface GigabitEthernet1/0/22
  description Link to GigabitEthernet1/0/24 on CAT9K-03
```

```
!
interface GigabitEthernet1/0/23
description Link to GigabitEthernet1/0/24 on CAT9K-04
!
interface GigabitEthernet1/0/24
description Link to GigabitEthernet1/0/24 on CAT9K-05
```

```
ntp_server { '1.2.3.4':
  ensure => 'present',
  key => 94,
  prefer => true,
  minpoll => 4,
  maxpoll => 14,
  source_interface => 'Vlan 42',
}
```

```
# create "devnet" OSPF process
cisco_ospf {"devnet":
    ensure => present,
}

# default VRF (global) settings for the "devnet" OSPF process
cisco_ospf_vrf {"devnet default":
    ensure => 'present',
    default_metric => '5',
    auto_cost => '46000',
}

# interface settings for the "devnet" OSPF process
cisco_interface_ospf {"Ethernet1/2 devnet":
    ensure => present,
    area => 200,
    cost => "200",
}
```

```
node 'default' {
  cisco_yang { 'my-config':
    ensure => present,
    target => '{"Cisco-IOS-XR-infra-rsi-cfg:vrf": [null]}',
    source => '{"Cisco-IOS-XR-infra-rsi-cfg:vrf": [
      "vrf": [
        "name": "VRF1"
      ]
    ]}'
  }
}
```

```
{  
    "vrf-name": "VOIP",  
    "description": "Voice over IP",  
    "vpn-id": {  
        "vpn-oui": 875,  
        "vpn-index": 3  
    },  
    "create": [  
        null  
    ]  
},  
}  
}
```

```
node 'default' {
  cisco_yang_netconf { 'my-config':
    target => '<vrfs xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg"/>',
    source => '<vrfs xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg">
      <vrf>
        <vrf-name>INTERNET</vrf-name>
        <create/>
        <description>Generic external traffic</description>
        <vpn-id>
          <vpn-oui>875</vpn-oui>
          <vpn-index>22</vpn-index>
        </vpn-id>
      </vrf>
    </vrfs>',
    mode => replace,
    force => false,
  }
}
```

```
$ docker save iox_app -o iox_app.tar
```

```
Device(config)#iox
*Apr  6 15:28:22: %IM-6-IOX_ENABLEMENT: Switch 1 R0/0: ioxman: IOX is ready.

Device# show iox-service

IOx Infrastructure Summary:
-----
IOx service (CAF) 1.11.0.4      : Running
IOx service (HA)                 : Running
IOx service (IOxman)             : Running
Libvirtd 1.3.4                  : Running
```

```
Device# app-hosting install appid iox_app package usbflash1:iox_app.tar
```

```
Device# app-hosting activate appid iox_app
```

```
Device# app-hosting start appid iox_app
```

```
Device# app-hosting stop appid iox_app
```

```
Device# app-hosting deactivate appid iox_app
```

```
Device# app-hosting uninstall appid iox_app
```

```
CAT9K(config)#app-hosting appid guestshell
CAT9K(config-app-hosting)# app-vnic management guest-interface 0

CAT9K# guestshell enable
Guestshell enabled successfully

CAT9K# guestshell run bash
[guestshell@guestshell ~]$ dohost "show system mtu"
Global Ethernet MTU is 1500 bytes.

[guestshell@guestshell ~]$ dohost "conf t ; hostname CAT9K-bash"
[guestshell@guestshell ~]$ exit

CAT9K-bash# guestshell run python3
Python 3.6.8 (default, Nov 21 2019, 22:10:21)
[GCC 8.3.1 20190507 (Red Hat 8.3.1-4)] on linux
>>> import cli
>>> cli.cli("show system mtu")
'Global Ethernet MTU is 1500 bytes.\n'
>>> cli.configurep("hostname CAT9K-python")
'Line 1 SUCCESS:  hostname CAT9K-python\n'
>>> quit()

CAT9K-python#
```

```
interface GigabitEthernet 0/0
  vrf forwarding Mgmt-vrf
  ip address 192.168.2.50 255.255.255.0

app-hosting appid iox_app
  app-vnic management guest-interface 0
    guest-ipaddress 192.168.2.100 netmask 255.255.255.0
  !
name-server0 8.8.8.8
app-default-gateway 192.168.2.1 guest-interface 0
```

```
interface GigabitEthernet1/0/1
    switchport trunk allowed vlan 10-12
    switchport mode trunk
!
interface GigabitEthernet1/0/2
    switchport mode access
    switchport access vlan 20
!
interface AppGigabitEthernet
    switchport trunk allowed vlan 10-12,20
    switchport mode trunk
!
app-hosting appid iox_app_vlan
    app-vnic AppGigabitEthernet trunk
        vlan 10 guest-interface 0
        guest-ipaddress 192.168.0.1 netmask 255.255.255.0
!
app-hosting appid iox_app_trunk
    app-vnic AppGigabitEthernet trunk
        guest-interface 0
```

```
interface GigabitEthernet1/0/20
  ip address dhcp
  ip nat outside
!
interface VirtualPortGroup1
  ip address 10.1.0.1 255.255.255.0
  ip nat inside
!
ip nat inside source list NAT_ACL interface GigabitEthernet1/0/20 overload
ip access-list standard NAT_ACL
  permit 10.1.0.0 0.0.0.255

app-hosting appid iox_app
  app-vnic gateway1 virtualportgroup 1 guest-interface 0 guest-ipaddress
  10.1.0.10 netmask 255.255.255.0
    app-default-gateway 10.1.0.1 guest-interface 0
```

```
$ curl -k -u admin:cisco -H 'Accept: application/yang-data+json' \
https://192.168.2.4/restconf/data/Cisco-IOS-XE-native:native/interface/Vlan=10
{
    "Cisco-IOS-XE-native:Vlan": {
        "name": 10,
        "description": "DevNet Vlan SVI interface",
        "ip": {
            "address": {
                "primary": {
                    "address": "10.10.0.1",
                    "mask": "255.255.255.0"
                }
            }
        }
    }
}
```

```
$ curl -k -u admin:cisco \
https://192.168.2.4/.well-known/host-meta
<XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
  <Link rel='restconf' href='/restconf'/>
</XRD>
```

```
$ curl -k -u admin:cisco \
https://192.168.2.4/restconf/
<restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <data/>
  <operations/>
  <yang-library-version>2016-06-21</yang-library-version>
</restconf>
```

```
$ curl -k -u admin:cisco -H 'Accept: application/yang-data+json' \
https://192.168.2.4/restconf/data/modules-state/
{
    "ietf-yang-library:modules-state": {
        "module-set-id": "349ef7f26dc8311bd6c10c0640b98636",
        "module": [
            {
                "name": "BGP4-MIB",
                "revision": "1994-05-05",
                "schema": "https://192.168.2.4:443/restconf/tailf/modules/BGP4-
MIB/1994-05-05",
                "namespace": "urn:ietf:params:xml:ns:yang:smiv2:BGP4-MIB",
                "conformance-type": "implement"
            },
            [... rest skipped ...]
```

```
$ python3 -m venv pyang
$ source pyang/bin/activate
(pyang)$ cd pyang/
(pyang)$ pip install pyang
[...]
Successfully installed lxml-4.6.3 pyang-2.4.0
(pyang)$ git clone https://github.com/YangModels/yang.git
[...]
Checking out files: 100% (44132/44132), done.
(pyang)$ cd yang/vendor/cisco/xe/1731/

(pyang)$ pyang Cisco-IOS-XE-native.yang -f tree --ignore-errors --tree-depth=1
module: Cisco-IOS-XE-native
  +-rw native
```

```
(pyang)$ pyang Cisco-IOS-XE-native.yang -f tree --ignore-errors \
--tree-path=native --tree-depth=2

module: Cisco-IOS-XE-native
  +-rw native
    +-rw default
    |
    |   ...
    |   +-rw bfd
[.....]
  +-rw interface
[.....]
```

```
(pyang)$ pyang Cisco-IOS-XE-native.yang -f tree --ignore-errors \
--tree-path=native/interface --tree-depth=3

module: Cisco-IOS-XE-native
  +-rw native
    +-rw interface
      +-rw AppNav-Compress* [name]
[.....]
      +-rw GigabitEthernet* [name]
[.....]
      +-rw Vlan* [name]
```

```
(pyang)$ pyang Cisco-IOS-XE-native.yang -f tree --ignore-errors \
--tree-path=native/interface/Vlan

module: Cisco-IOS-XE-native
  +-rw native
    +-rw interface
      +-rw Vlan* [name]
        +-rw name                      uint16
        +-rw description?              string
        +-rw switchport-conf
          |   +-rw switchport?    boolean
[... rest skipped ...]
```

```
$ curl -k -u admin:cisco -H 'Accept: application/yang-data+json' \
https://192.168.2.4/restconf/data/Cisco-IOS-XE-native:native/interface/Vlan
{
  "Cisco-IOS-XE-native:Vlan": [
    {
      "name": 1,
```

```
"ip": {
    "address": {
        "primary": {
            "address": "192.168.2.4",
            "mask": "255.255.255.0"
        }
    }
},
{
    "name": 10,
    "description": "DevNet Vlan SVI interface",
    "ip": {
        "address": {
            "primary": {
                "address": "10.10.0.1",
                "mask": "255.255.255.0"
            }
        }
    }
}
]
```

```
$ curl -k -u admin:cisco -H 'Accept: application/yang-data+json' \
https://192.168.2.4/restconf/data/Cisco-IOS-XE-native:native/interface/Vlan=10
{
    "Cisco-IOS-XE-native:Vlan": {
        "name": 10,
        "description": "DevNet Vlan SVI interface",
        "ip": {
            "address": {
                "primary": {
                    "address": "10.10.0.1",
                    "mask": "255.255.255.0"
                }
            }
        }
    }
}
```

```
(pyang)$ pyang Cisco-IOS-XE-native.yang -f tree --ignore-errors \
--tree-path=native/ip/route/ip-route-interface-forwarding-list --tree-depth=4
module: Cisco-IOS-XE-native
  +-rw native
    +-rw ip
      +-rw route
        +-rw ip-route-interface-forwarding-list* [prefix mask]
```

```
$ curl -k -u admin:cisco -H 'Accept: application/yang-data+json' \
https://192.168.2.4/restconf/data/Cisco-IOS-XE-native:ip/route/ \
ip-route-interface-forwarding-list=0.0.0.0,0.0.0.0
{
  "Cisco-IOS-XE-native:ip-route-interface-forwarding-list": {
    "prefix": "0.0.0.0",
    "mask": "0.0.0.0",
    "fwd-list": [
      {
        "fwd": "192.168.2.1"
      }
    ]
  }
}
```

```
$ curl -k -u admin:cisco -H 'Accept: application/yang-data+json' \
"https://192.168.2.4/restconf/data/Cisco-IOS-XE-native/\" \
"interface/Vlan=10?fields=name;description"
{
  "Cisco-IOS-XE-native:Vlan": {
    "name": 10,
    "description": "DevNet Vlan SVI interface"
  }
}
```

```
$ cat Dockerfile
FROM python
ENTRYPOINT ["echo","Blue"]
CMD ["Green"]
$ docker build -t colors . >/dev/null
$ docker run -it colors "Red"
```

# Contents

Cover Page

Title Page

Copyright Page

About the Author

About the Technical Reviewer

Dedication

Table of Contents

Table of Figures

Introduction

## 1. Software Development and Design

1.1 Describe distributed applications related to the concepts of front end, back end, and load balancing

1.2 Evaluate an application design considering scalability and modularity

1.3 Evaluate an application design considering high-availability and resiliency (including on-premises, hybrid, and cloud)

1.4 Evaluate an application design considering latency and rate-limiting

1.5 Evaluate an application design and implementation considering maintainability

1.6 Evaluate an application design and implementation considering observability

1.7 Diagnose problems with an application given logs related to an event

- 1.8 Evaluate choice of database types with respect to application requirements (such as relational, document, graph, columnar, and Time Series)
  - 1.9 Explain architectural patterns (monolithic, services-oriented, microservices, and event-driven)
  - 1.10 Utilize advanced version control operations with Git
  - 1.11 Explain the concepts of release packaging and dependency management
  - 1.12 Construct a sequence diagram that includes API calls
- 1.13 Chapter 1 Review Questions
2. Using APIs
  - 2.1 Implement robust REST API error handling for timeouts and rate limits
  - 2.2 Implement control flow of consumer code for unrecoverable REST API errors
  - 2.3 Identify ways to optimize API usage through HTTP cache controls
  - 2.4 Construct an application that consumes a REST API that supports pagination
  - 2.5 Describe the steps in the OAuth2 three-legged authorization code grant flow
- 2.6 Chapter 2 Review Questions
3. Cisco Platforms
  - 3.1 Construct API requests to implement ChatOps with Webex API
  - 3.2 Construct API requests to create and delete objects using Firepower device management (FDM)

3.3 Construct API requests using the Meraki platform to accomplish these tasks

3.4 Construct API calls to retrieve data from Intersight

3.5 Construct a Python script using the UCS APIs to provision a new UCS server given a template

3.6 Construct a Python script using the Cisco DNA center APIs to retrieve and display wireless health information

3.7 Describe the capabilities of AppDynamics when instrumenting an application

3.8 Describe steps to build a custom dashboard to present data collected from Cisco APIs

3.9 Chapter 3 Review Questions

#### 4. Application Deployment and Security

4.1 Diagnose a CI/CD pipeline failure (such as missing dependency, incompatible versions of components, and failed tests)

4.2 Integrate an application into a prebuilt CD environment leveraging Docker and Kubernetes

4.3 Describe the benefits of continuous testing and static code analysis in a CI pipeline

4.4 Utilize Docker to containerize an application

4.5 Describe the tenets of the “12-factor app”

4.6 Describe an effective logging strategy for an application

4.7 Explain data privacy concerns related to storage and transmission of data

4.8 Identify the secret storage approach relevant to a given scenario

4.9 Configure application-specific SSL certificates

4.10 Implement mitigation strategies for OWASP threats (such as XSS, CSRF, and SQL injection)

4.11 Describe how end-to-end encryption principles apply to APIs

4.12 Chapter 4 Review Questions

## 5. Infrastructure and Automation

5.1 Explain considerations of model-driven telemetry (including data consumption and data storage)

5.2 Utilize RESTCONF to configure a network device including interfaces, static routes, and VLANs (IOS XE only)

5.3 Construct a workflow to configure network parameters with:

5.4 Identify a configuration management solution to achieve technical / business requirements

5.5 Describe how to host an application on a network device (including Catalyst 9000 and Cisco IOx-enabled devices)

5.6 Chapter 5 Review Questions

## 6. Appendix A: RESTCONF URI Demystified (IOS XE)

## 7. Appendix B: Answers to Chapter Review Questions

7.1 Answers to Chapter 1: Software Development and Design

7.2 Answers to Chapter 2: Using APIs

7.3 Answers to Chapter 3: Cisco Platforms

## 7.4 Answers to Chapter 4: Application Deployment and Security

## 7.5 Answers to Chapter 5: Infrastructure and Automation

### Code Snippets

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190