**1.0 - Software Development and Design (1.1 – 1.8)**

**1.4 - Compare software development methods (agile, lean, and waterfall):**

**Waterfall Method:**

* Consists of discrete stages which flow from one to another like a waterfall or flow:
  + Requirements
  + Design
  + Implement
  + Test
  + Deliver

|  |  |
| --- | --- |
| Advantages   * Works well when requirements don’t change. * Clearly defined stages * Easy to mange | Disadvantages   * Cannot adapt to change * Hard to go back to previous stages * Entire tested only once |

* When to Use:
  + When definition, not speed, is key to success.
  + The project is short or relatively simple.
  + The project could impact on important systems, such as banking mainframes or public sector databases (though this depends on the project)
  + If the team or organization isn’t in a mature enough state to adopt an agile approach

**Agile Method (Scrum):**

* **Core Agile Values or Tenets:**

|  |  |
| --- | --- |
| **More Valued** | **Less Valued** |
| Individual and interaction | Process and tools |
| Working software | Comprehensive docs |
| Customer collaboration | Contract negotiation |
| Responding to change | Following a plan |

**NOTE: Both are equally valued but stand in a hierarchy.**

* Consist of the stages of the waterfall model done rapidly over a small period and repeated often called **Sprints.**
  + **Sprint** 🡪 Range from 2 to 4 weeks and at the end a “Deliverable” is reached. These begin with a few hours of planning and then works begins.
  + **Backlog 🡪** list of wanted features in a prioritized list developed by the product owner which are discovered in the “Requirement & Analysis Phase”.
  + **User Stories 🡪** list of tasks broken down from a larger feature which is at the top of the backlog.
  + **SCRUM Master 🡪** has the purpose to report and/or help remove obstacles.

|  |  |
| --- | --- |
| Advantages   * Deliver is faster * Fosters teamwork. * Little planning required | Disadvantages   * Requires customer interactions with each sprint. * Not many docs; people dependency * Requires a competent team leader |

* When to use:
  + When speedy production is more valuable than the quality of the product.
  + When the product is meant for an industry with quickly changing standards.

**Lean (Kanban):**

* Similar to lean but uses a work flow chart where work is continuous rather than sprint based such as Kanban card, where when a card is marked in process is must be completed and moved to the complete section before the backlog can be touched.
* Principles:
  + **Eliminate waste, Amplify Learning, deliver fast, Empower Team, Build Integrity, Optimize**

|  |  |
| --- | --- |
| Advantages   * Minimal batch size * Task oriented * Fast delivery | Disadvantages   * Requires discipline * Needs sensible WIP limits |

**1.6 - Identify the advantages of common design patterns (MVC and Observer):**

Design patterns are a set of rules to write code, it delivers a map or guidelines to develop code.

* **Singleton**:
  + Restricted Classes which are/is instantiated once during execution.
  + Creates only one instance of a class.
    - example 🡪 database or device connection methods
* **Observer:**
  + Utilizes **Subjects & Observers:**
    - **Subjects 🡪** Are code which is doing something/task like a ticker counter. It also registers observers.
    - **Observers/Subscribers 🡪** Is code which observers a subject when a event occurs the subjects trigger a callback or notify method from where different observers execute their own code.
      * **Example 🡪 Code Example**
* **Model View Controller**:
  + A high-level abstraction where responsibilities are divided up into three loosely coupled components. With the goal is to make the various layers of the application—such as data access, business logic, and presentation (to the end user)—modular.
  + Abstraction where responsibilities are divided into three loosely coupled components.
    - **Model:** Component that stores data and the logic of the application.
    - **View:** Component that displays data
    - **Controller:** Component that handles logic and directs model and View
      * **Example** 🡪 **Code Example**

**1.1 - Compare data formats (XML, JSON, and YAML):**

|  |  |  |
| --- | --- | --- |
| **JSON** | **YAML** | **XML** |
| {  “customer\_list”: [  {  “name”: “Test”,  “balance”: 19,  “state”: [1,2,2],  “power”: True  }  ]  } | ---  # YAML  customer\_list:  - name: “Test”  balance: 19  state: [1,2,2]  power: True | <?xml version=”1.0”>  encoding=”UTF-8” ?>  <root>  <customer\_list>  <name>Test</name>  <balance>19</balance>  <power>True</power>  </customer\_list>  </root> |
| lightweight  native to JS  compressed to a single line whitespace🡪no meaning | Human Friendly  Ideal for configuration  Whitespace🡪Matters | Legacy support  Supported by SOAP Natively  Heavy in operation |

**1.5 - Explain the benefits of organizing code into methods / functions, classes, and modules:**

**Common Sense.**

**1.3 - Describe the concepts of test-driven development:**

Test-driven development (TDD) refers to the concept of building test case for software before-hand with the objective to streamline development in terms of testing down the line. This allows for changes and modification to the production code with minimal issues as specific tests can be run to quickly verifying the changes did not have unattended effects.

**1.2 - Describe parsing of common data format (XML, JSON, and YAML) to Python data structures:**

|  |  |  |
| --- | --- | --- |
| **JSON** | **YAML** | **XML** |
| with open(path, “r”) as file:  Import json  data = json.load(file) | with open(path, “r”) as file:  import yaml  data = yaml.safe\_load(file) | with open(path, “r”) as file:  import xmltodict  data = xmltodict.parse(file.read())[“root”] |
| Load()🡪import native JSON  Loads()🡪import from sting  Dump()🡪write to file  Dumps()🡪JSON—to—serialized string | safe\_load()🡪more secure | xml.etree.ElementTree |

**1.7 - Explain the advantages of version control:**

|  |  |
| --- | --- |
| * Code Review/Reducing Errors * Identifiability * Clarity * Reduced Duplication * Collaboration capabilities * Accountability and visibility * Isolation for a work environment * Safety with backup and restore * Work anywhere | * **Local Control Systems (LCS):** Hosted locally on a computer. * **Centralized Version Control (CVS):** Is a server-client model, where all files are hosted on a server from where users checkout specific files as per needed, which are downloaded to their local version control systems. * **Distributed Version Control Systems (DVCS):** This is a peer-to-peer model which uses a hosting service where a user clones the entire repo and its version history to their own system. Allowing for simultaneous workflows on the entire project. |

**1.8 - Utilize common version control operations with Git:**

* <https://www.youtube.com/watch?v=2sjqTHE0zok>
* Commands: <https://github.com/joshnh/Git-Commands>

**Code Review:**

* **Formal Code Review**: Process where the code is check line by line via a reviewer in a series of meetings until the desirable stage is reached
* **Change-Based Review**: Is a tool assisted process which changes are tested by developers providing comments on improvements. This allows multiple developers/reviewers to simultaneously review the code.
* **Over-the-shoulder Review**: Exactly as it sounds.
* **Email Pass Around**: Exactly as it sounds.

**Testing:**

* **Functional Testing**: Test developed product via testing its functions like database connections etc.
* **Non-Functional Testing**: Examines usability, performance, security, resiliency, compliance, and other issues.
* **Unit-Testing:** Detailed functional testing where small pieces of code are developed and tested.
* **Integration Testing**: Build specific code within the software to support testing like functions, method which run a library like PyTest.
* **Test Driven Development**: This means to build tests and write testing procedures even before writing code. Meaning developing applications targeting at passing specific tests which are catered to needs/features.

**2.0 - Understanding and Using APIs**

General Information:

* Northbound API: which function north of the controller towards the application
* Southbound API: which function south of the controller towards the data plane.

**2.1 - Construct a REST API request to accomplish a task given API documentation:**

REST API Request **🡪** HTTPs request which are made up of 4 components:

* **Uniform Resource Identifier/locator (URI) 🡪** identify the resource and optional queue. Example  
  + 
* **HTTP Methods 🡪** CRUD verbs:
  + Table

    Description automatically generated
* **Header 🡪** HTTP based dict objects key-value pair separated by a (:) colon. Common Types:
  + **Request Header 🡪** information that does not relate to the content of the message.
  + **Entity Header 🡪** describes the contents of the body.
* **Body 🡪**  Contains the request or response data. Example: POST, PUT etc.

REST API Response **🡪** HTTP messages which relay the results of a client HTTP request. This may contain data or other information. Common Types are:

* Section 2.6 Identify the parts of an HTTP response (response code, headers, body) 🡪 Below

**2.2 - Describe common usage patterns related to webhooks:**

* **Webhooks:** Are reverse API which allow applications to get real-time data without the need for polling, where the webhook sends a POST message to the application when a certain event/task is triggered.
  + NOTE: In most cases security validation is required by the webhook.
* **Consuming Webhooks**: Webhooks requirements are:
  + The application must always be running to receive HTTP POST requests.
  + The application must register a URI on the webhook provider, so the provider knows where to send a notification when target events occur.

**2.8 - Compare common API styles (REST, RPC, x`, synchronous, and asynchronous) & 2.3 - Identify the constraints when consuming APIs:**

* **Synchronous:** Meaning the code execution will block (or wait) for the API to return before continuing. Like a conversation with a person where one speaks and then waits for the reply.
* **Asynchronous:** Calls are not block (or wait) for the API call to return from the server. Execution continues in your program, and when the call returns from the server, a "callback" function is executed.
* **Remote Procedure Calls (RPC):** allows for remote execution of code on a remote node in a network where the behavior is as if the code was executed locally. This functions as a “Synchronous” call where the client is blocked until the reply from the server is received. RPC uses XML-RPC.
* **Simple Object Access Protocol (SOAP):** ): is a messaging protocol which can communicate between applications built on different platforms and languages. SOAP uses HTTPS for transport and is based on XML in a web services and client model. SOAP messages have four main components:

|  |  |
| --- | --- |
|  | <?xml version="1.0"?>  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">  <soap:Header/>  <soap:Body>  <soap:Fault>  <faultcode>soap:Server</faultcode>  <faultstring>Query request too large.</faultstring>  </soap:Fault>  </soap:Body>  </soap:Envelope> |
| * **Envelope:** root message which defines the namespace. * **Header:** contain application specific information (Optional). * **Body:** contain the actual SOAP message in XML. * **Fault:** indicates error messages.   + Book Chap-6 Table 6-4, 5. |

* **Representational State Transfer (REST):** is an architecture for passing information between two parties via several protocols with HTTP/HTTPS being the most common. REST constraints are:  
  + **Client-Server:** the client and server are independent of each other.
  + **Stateless:** Request must contain all information needed for the communication.
  + **Cache:** Responses must state whether the response is cacheable or non-cacheable. If it is cacheable, the client can use the data from the response for later requests.
  + **Uniform Interface:** the principles for the interface are:  
    - **Identification of resources:** meaning the resource for modification/access must be identified in the request.
    - **Manipulation of resources through representations**: means the representation sent by the server must contain the data required for client for manipulation.
    - **Self-descriptive messages:** Where each message sent/received contains all information required to process the message.
    - **Hypermedia as the engine of application state:** Where data sent by the server must contain additional actions and resources available for the client to access supplemental information about the resource
  + **Layered System:** the system is made of hierarchical layers which provide services only to the layer above it.
  + **Code on Demand:** Optional and refers to the fact that information retrieved by a REST service can include executable code or links.
* **Pagination:** allows to limit the number of results returned based on a configured/entered range. Types of this include:
  + **Offset Based**: identifies offset and limit example “returns the device between 100 -150”.
  + **Keyset-Based**: it works by returning a pointer dataset.
* **Rate Limiting and Monetization**: This allows to limit the rate of requests to a server and can be implemented in the follow algorithms:
  + **Leaky Bucket:** Puts all request in an ordered queue and processes in sequence once the queue is full request are discarded.
  + **Token Bucket:** This gives a user a defined number of tokens that can be use within a certain time if not used they accumulate. However, when all tokens are used the request is dropped.
  + **Fixed Window:** This gives a pre-defined rate for a client such as 10 request per hour with accumulation not being possible.
  + **Sliding Window:** This uses time windowing such as rate is five requests per minute, when the server receives a new request, it checks how many requests have been made in the last 60 seconds. If five requests have already been made, then the new request will be rejected.
  + NOTE: exceeding these limited results in 403: Forbidden or 429: Too Many Requests.

**2.7 - Utilize common API authentication mechanisms: basic, custom token, and API keys:**

* **HTTP Header 🡪** Authorization: <auth\_type> <credentials>
* **Authentication 🡪**  Verify User Identity.
* **Authorization 🡪** Proving User Identity.
* **REST API Authentication Mechanism:**
  + **Basic 🡪** clear text pass and username encoded in Base64.
  + **Bearer of Token 🡪** Standard HTTP authentication scheme, with technologies like OAuth and SSO. These tokens are generated by authentication servers which comprise of a string created via a cryptographic function which uses a base64 encoded user and password. Every request includes this token.
  + **API Keys 🡪** Unique alphanumeric strings generated via a server assigned to a user which is obtained usually through login credentials. Types Include:
    - **Public API Keys 🡪** Shareable and allows access to data via query strings.
    - **Private API Keys 🡪** Kept Secret.

**2.9 - Construct a Python script that calls a REST API using the requests library:**

* See Python Code in Section 2.0

**2.4 - Explain common HTTP response codes associated with REST APIs & 2.5 - Troubleshoot a problem given the HTTP response code, request, and API documentation**

**HTTP-Status Codes & Explanations - HTTP/1.1:**

* **1xx Informational 🡪** request received continuing processing.
* **2xx Success 🡪** Successful
* **3xx Redirect 🡪** More actions needed to complete request.
* **4xx Client Error 🡪** 
  + **400 🡪**  Bad Request.
  + **401 🡪** Unauthorized Authentication Failed.
  + **403 🡪** Forbidden Authorization Failed.
  + **404 🡪**  Not Found.
  + **405 🡪** Method not allowed, request was fine, but server does not understand action.
  + **406 🡪** Not Acceptable
  + **407 🡪**  Proxy authentication required.
  + **409 🡪** Conflict with target.
  + **415 🡪** Unsupported Media Type.
  + **429 🡪** To Many Requests.
* **5xx Server Error 🡪**
  + **500 🡪** Internal Server Error
  + **501 🡪** Not Implemented does not support function.
  + **502 🡪** Bad Gateway response from an inbound server invalid.
  + **503 🡪** Service Unavailable
  + **504 🡪** Gateway Timeout
* **Use Example for practice this is more related to 2.5 then 2.4.**

**2.6 - Identify the parts of an HTTP response (response code, headers, body):**

|  |  |
| --- | --- |
| **GET /jokes/random?firstName=John&amp;lastName=Doe HTTP/1.1**  **######## Request Headers ##################**  **Accept: application/vnd.github.mercy-preview+json**  **User-Agent: PostmanRuntime/7.25.0**  **Cache-Control: no-cache**  **Postman-Token: 20c01322-eff1-4dc9-bfc4-5b6bfaacbbf2**  **Host: api.icndb.com**  **Accept-Encoding: gzip, deflate, br**  **Connection: keep-alive**  **HTTP/1.1 200 OK <--------- Response Code** | **######## Response Headers ##################**  **Date: Wed, 17 Jun 2020 12:27:58 GMT**  **Content-Type: application/json**  **Transfer-Encoding: chunked**  **Connection: keep-alive**  **Set-Cookie: \_cfduid=d279ef5bb052a488b087a72c7a902bc751592396878; expires=Fri, 17-Jul-20 12:27:58 GMT; path=/; domain=.icndb.com; HttpOnly; SameSite=Lax**  **Access-Control-Allow-Origin: \***  **Access-Control-Allow-Methods: GET**  **Cache-Control: no-cache, must-revalidate**  **Expires: Sat, 26 Jul 1997 05:00:00 GMT**  **Vary: User-Agent**  **CF-Cache-Status: DYNAMIC**  **cf-request-id: 0363d92a750000f3ef0db09200000001**  **Server: cloudflare**  **CF-RAY: 5a4cc48a5848f3ef-LHR**  **Content-Encoding: gzip** |
| **######## Body ##################**  **{ "type": "success", "value": { "id": 282, "joke": "John Norris can skeletize a cow in two minutes.", "categories": [] } }**  **# REST Constraints Cachable Headers**  **Last Modified:**  **Cache Control:**  **Last Modified** |  |

**3.0 Cisco Platforms and Development**

**3.1 - Construct a Python script that uses a Cisco SDK given SDK documentation:**

See Code Examples

**3.2 - Describe the capabilities of Cisco network management platforms and APIs (Meraki, Cisco DNA Center, ACI, Cisco SD-WAN, and NSO):**

**Meraki:**

* **Centralized Management and Control 🡪**  via the web-based UI and REST API single source management.
* **Built-in High Availability 🡪** handles behind the scenes via the cloud-based management.
* **Mobility and Enrollment 🡪** rapid onboarding and management access anywhere is supported.
* **Remote Deploy Applications 🡪** can deploy on devices and manage them.
* **Dashboard API 🡪** provides methods to interact directly with the Meraki cloud platform and Meraki managed devices. Using the API, some of the use cases are as follows:
  + Add new organizations, administrators, networks, devices, VLANs, and more
  + Onboard and off-board employees
  + Build your dashboard for store managers or field technicians
* **Scanning API 🡪** enables Cisco Meraki users to detect and aggregate real-time data for custom applications. The Scanning API delivers data in real-time from the Meraki cloud and can be used to detect Wi-Fi and Bluetooth Low Energy (BLE) devices in real-time. The elements are exported via an HTTP POST of JSON data to a specified destination server.
* **mV Sense API 🡪 provides a collection of endpoints to interact with Meraki cameras, zone, and analytics.**
* **External Captive Portal API 🡪** extends the power of the built-in Meraki splash page functionality by providing complete control of the content and authentication process offering to redirect login and authentication to client-provided servers, using your own authentication, authorization, and accounting (AAA) servers as well.
* **Wireless Health API 🡪** allows you to retrieve wireless health information such as connection health, connection failures, and network latency.

**Cisco SD-WAN:**

* **Centralized Management and Control 🡪** via the vManage controller web-based UI and REST API.
* **Client Segmentation 🡪** provides separation for different client & organizations.
* **Integrated Security 🡪** has specific security application control.
* **Zero-touch Provisioning 🡪**  via the vBond controller.

**Cisco ACI:**

* **Centralized Management and Control 🡪**  via the APIC controller.
* **Policy Enforcement 🡪** via EPGs and Contracts.
* **Multi-site 🡪** supports multiple sites.
* **Automated Fabric Provisioning 🡪** via APIC controller

**Cisco DNA Center:**

* **Intent API 🡪** is a northbound REST API that exposes specific capabilities of the Cisco DNA Center platform.
* **Software Image Management (SWIM) API 🡪** enables you to retrieve information about available software images, import images into Cisco DNA Center.
* **PnP API 🡪** enables you to manage PnP projects, settings, workflows, virtual accounts, and PnP-managed devices.
* **Operational tools 🡪** enable you to configure and manage CLI templates, discover network devices, configure network settings, and trace paths through the network.
* **Authentication API 🡪** Used to authenticate users and such.
* **Integration API 🡪** Used to integrate with other services.

**3.3 - Describe the capabilities of Cisco compute management platforms and APIs (UCS Manager, UCS Director, and Intersight):**

**Unified Computing Manager:**

* **Cisco UCSM 🡪** 106 & **Cisco UCS Central 🡪** 10,000 servers.
* **Service Profiles 🡪** used to assign identity to servers
  + 1 per server at a time
* **UCS Manager Servers Supported 🡪** Blade Chassis (B-Series), (C-Series) connected to UCS Fabric Interconnects (FIs).
* **UCS Management Information Model 🡪** Same as MIT with the top being “SYS” which contain Distinguished and Relative Names. **NOTE**: This model is stored in the “Data Management Engine (DME)”.
* **Ports 🡪 443 & 80**
* **REST API 🡪** The UCS Manager programmatic interface is the XML API
* **Authentication Methods 🡪**
  + **aaaLogin**: Login method
  + **aaaRefresh**: Refreshes the authentication cookie
  + **aaaLogout**: Exits the session and deactivates the corresponding authentication cookie
* XML API endpoint, [*https://10.10.20.110/nuova*](https://10.10.20.110/nuova)*,* Content-Type header, application/xml

**Cisco Unified Computing Director:**

* **Features 🡪** extends Cisco UCS to provide visibility and management of data center infrastructure components.
  + Create , clone, and deploy services and templates.
  + Monitor.
  + Manage secure multi-tenant environments.
  + Supporting Automated workflows
    - VM provisioning, Storage resource configs, Tenant onboarding etc.
  + **NOTE:** API v1 🡪 JOSN or XML \_\_\_\_\_ APIv2 🡪 XML only, See code for example
  + Each REST API request must be associated with an HTTP header called **X-Cloupia-Request-Key**

**Cisco Intersight:**

This is a Software as a Service (SaaS) system management platform capable of managing infrastructure via a web-based cloud UI. The REST API allows access to the “Management Information Model” (MIM) same/like MIT.

* The Management Information Tree (MIT) is a tree structure with nodes where each node represents a Managed Object (MO) or a group of them. Like seen before.
* API Keys can be generated via the web client or through the SDK.
* **Authentication Methods 🡪** API keys, Session cookies

**3.4 - Describe the capabilities of Cisco collaboration platforms and APIs (Webex Teams, Webex devices, Cisco Unified Communication Manager including AXL and UDS interfaces, and Finesse):**

**Cisco Unified Communications Manager 🡪**

* IP-Based solution which supports Voice, Video, data in a single on-prem box with call control and session management.
* Internal Database 🡪 Subscriber Cluster Architecture.
* **REST API 🡪 Administrative XML Layer:**
  + SOAP based interface supporting XML and every operation supported via the web GUI.
    - Graphical user interface, chart

      Description automatically generated
    - Graphical user interface, text, application, email

      Description automatically generated
  + User Data Services 🡪 enables modification of UC config database.  
    - Supports Single Sign-on SSO and Basic Authentication.   
        
      Graphical user interface, text, application, email

      Description automatically generated

**Cisco Finesse 🡪**

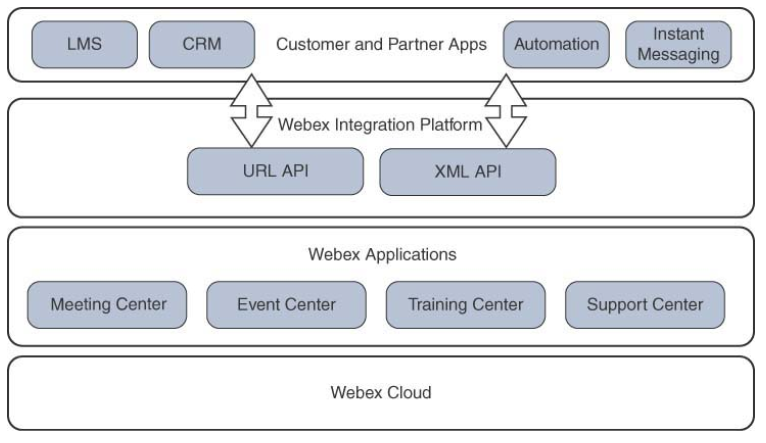
Web-based contact center agent and supervisor desktop which sits on either the Contact Center Enterprise or Unified Contact Center Express. It has a REST API and JavaScript API.

* **Contact Center 🡪** Centralized location for handling service request.
* **Inbound Tasks 🡪** incoming request
* **Outbound Tasks 🡪**  reply to requests.
* **Call Routing 🡪**  uses a skill-based call routing for call distribution and agent monitor for availability. States are Ready, Not Ready, Talking, Work, Logout.
* **REST API 🡪**
  + HTTP & HTTPS based.
  + Base64-encoded Authentication or Single Sign-on Tokens.
  + API Classified into
    - User, Dialog, Queue, Team, ClientLog, Task Routing APIs, Single Sign-On, TeamMessage.
* **API Offered 🡪**
  + **Cisco Finesse desktop APIs 🡪** used by agents and supervisors to communicate between other Cisco entities
  + **Cisco Finesse configuration APIs**
  + **Cisco Finesse serviceability APIs**
  + **Cisco Finesse notifications**

**Cisco Webex Teams API 🡪**

* **Authentication 🡪**
  + **Personal Access Token 🡪**  Single Person – 12 Hours.
  + **Integrations 🡪** OAuth or OAuth-2 with access scope definitions. Table 10-2 in the book details API
* **Targets 🡪**
  + Organizations API 🡪 <https://webexapis.com/v1/organizations> {Supported Methods: GET }
  + Team API 🡪 <https://webexapis.com/v1/teams> {Supported Methods: GET, POST, PUT, DELETE}
  + Room API 🡪 <https://webexapis.com/v1/rooms> {Supported Methods: GET, POST, PUT DELETE}
  + Membership API 🡪 <https://webexapis.com/v1/memberships> {Supported Methods: GET, POST, PUT, DELETE}
  + Message API 🡪 <https://webexapis.com/v1/messages> {Supported Methods: GET, POST, DELETE}
* **Bots 🡪**
  + **Types 🡪**
    - Notification, Controller, Assistant
  + **Frameworks and Tools** **🡪**
    - **Flint**:
      * Open source with support for regex patterns matching for messages and more.
    - **Botkit**:
      * Open source with advanced conversational support as well as integrations with a comprehensive array of natural language processing and storage providers.
* **Guest Issuer 🡪**
  + Gives temp access to users within an organization.
  + Authenticated via a JSON Web Token (JWT).

**Cisco Webex Meetings API 🡪**

* **URL API 🡪**
  + Web-based delivers external hook into Webex Services such as Single-sign-on (SSO) etc.
* **XML API 🡪**
  + Is an advanced integration way which uses a service-oriented architecture (SOA) to provide comprehensive services to external applications wishing to interact with one or more Webex services.
  + 
* **Authentication 🡪**
  + Admin Account, User Account, Webex Meetings Integrations. Examples in the code section

**Cisco Webex Teams SDK 🡪**

* Supports multiple languages (Java, Node.js, PHP, Python Go)

**Cisco Device xAPI 🡪**

**Authentication 🡪**

* Uses HTTP basic access authentication as a user with the ADMIN role
* **Basic Authentication** 🡪 full auth every request 🡪 may affect performance
* **Session Authentication**:
  + Uses username and password via a POST method **🡪 SessionId-cookie.**
  + This cookie is used for subsequent requests.
* Examples 🡪 Python Code
* **Event Notification Webhook**:
  + Can be used to make a device endpoint port HTTP event notification on changes to the API state.
  + Command for is **xCommand HttpFeedback**.

**3.5 - Describe the capabilities of Cisco security platforms and APIs (Firepower, Umbrella, AMP, ISE, and ThreatGrid):**

**Cisco AMP Endpoint Security 🡪**

Endpoint security solution which supports several platforms with a cloud-management tool. AMP supports:

|  |  |  |
| --- | --- | --- |
| **Prevention** | **Detection** | **Response and Automation** |
| Identifies Malware | Continuous Monitoring | Automated Investigation and Remediation |
| Global threat intelligence | Ensures visibility | Provides 🡪 Sandboxing |
|  |  | Reporting |

**AMP Endpoint API:**

* **Authentication 🡪** 
  + Support 🡪 Client ID via API Key & Basic HTTP Authentication Base 64-encoed string.
* **API Rate Limits 🡪**
  + X-Rate-Limit 🡪 Number of allowed requests in a current period.
  + X-Rate-Remaining 🡪 Number of requests left before reaching the limit.
  + X-Rate-Limit-Reset 🡪 Number of seconds before the limit is reset.
* **API Pagination 🡪**
  + Uses 🡪 locations within the response in an offset parament.
  + Offset value 🡪 in the body of the request such as:
    - {“offset”: 250}

**Cisco Firepower Products:**

Firepower Management Center (FMC) is a central management console for the Firepower Threat Defense (FTD). This console can configure all aspects of the FTD. FMC is also a central configuration database supporting filesharing objects and policies. The APIs allows for automated configuration of devices.

**Firepower Management Center API:**

* **Compatibility 🡪** 
  + FDM / FTD-API / CDO 🡪 can co-exist.
  + FMC 🡪 for advance cases
* **Architecture** 🡪
  + FMC can run on 🡪 vSphere or AWS
  + Firepower Management Center (FMC) 🡪 multi-device manager for large enterprise deployments.
  + Firepower Device Manager (FDM) 🡪 "single" device manager contains the FDM and Next Generation Firewall APIs.
* **Integrations 🡪**
  + ISE, Threat Grid, Umbrella etc.
* **Authentication 🡪**
  + **Basic authentication** 🡪 requests to get a persistence token.
  + **Firepower APIs use token-based authentication**.
  + **Token life 🡪 30 mints.**
* **API Limits 🡪**
  + - **FMC 🡪 120 Messages/per minute from a single IP @ Payload Size 🡪 20480 bytes.**
    - **FTM 🡪 API’s upper limit value is 1000.**

**Cisco Identity Service Engine (ISE):**

* **Architecture 🡪** Consists of nodes with defined node types as follows:
  + **Admin Node 🡪** Admin Operations.
  + **Policy Service Node 🡪** Provides network access, provisioning, profiling services etc.
  + **Monitoring Node 🡪**  As the name suggests
  + **pxGrid Node 🡪** Enables the system to exchange policy and configuration data between nodes.
* **Integrations 🡪** RBAC (role-based access control), Okta/SAML Single-Sign On (SSO), Lightweight Directory Access Protocol (LDAP), Active Directory (AD).
* **Scale 🡪**  250, 000 Active and up to 1M Devices
* **REST APIs:**
  + **Session API 🡪** allows developers to gather session and node-specific information.
    - **External RESTful Services (ERS) API 🡪** enables operation of the follow resources:
      * Endpoints, Endpoint identity groups, Guest users and internal users, Identity groups, Portals, Profiler policies, Network devices, Network device groups, Security groups
* **Authentication (ERS API) 🡪**
  + Uses HTTP basic authentication 🡪 username & password via Base64-encoded.
  + Supports 🡪 GET, POST, PUT, DELTE and others

**Cisco Threat Grid:**

Malware analysis platform which can be deployed as an appliance or in the cloud. Integration is also supported:

**Grid API 🡪**

* **Format 🡪** All cares are made @ <https://panacea.threatgrid.com/api/>
  + **Example** 🡪 **<ver>/<api-endpoint>?q=<query>&api\_key=apikey**
* **Authentication 🡪** go to UI Page 🡪 Manage Users 🡪 User details page 🡪 copy API key
* **Example 🡪 in Code.**

**Feeds 🡺**

* **Threat Grid supports three different types of feeds:**
  + **Sample feeds:** These are all observables seen**.**
  + **Indicator of Compromise (IOC) feeds:** These are observables seen via business intelligence. IOCs are used to indicate that the system has been affected by some form of malware.
  + **Curated feeds:** These are highly curated and high-confidence feeds.

**Cisco Umbrella:**

This a cloud-based secure gateway which processes DNS request to analyze and learn about various activities and locking requests which are unwanted or are from malicious destinations before a connection can be established. Functions on several protocols such as HTTP, HTTPS, and others. Processed steps:

* 1. Analyze DNS request.
* 2. Checks associated policies.
* 3. Allow IP address queue to the user.
  + 4. If DNS request triggers a policy
  + 5. Blocks the page and inform user with a message.

**Cisco Umbrella AIPs 🡪**

* Supports or House the follow specific APIs:
  + Management API
  + Reporting API
  + Console Reporting API
  + Network Device Management API
  + Enforcement API
  + Investigate API
  + NOTE: Details on all of these are in the book, memorize if needed but not sure.
* Authentication:
  + Support HTTP-basic authentication with Base64 encoding.
* NOTE: Examples in the code.

**4.0 - Application Deployment and Security:**

**4.1 - Describe benefits of edge computing:**

* **User Benefits 🡪** Data processing and storage moved close to the user.
* **Content Delivery Network 🡪** Distributed platform of servers which minimizing delays.

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Speed/Latency  Privacy  Resiliency  Scalability | Resource Requirements  Infrastructure Complexity  Distribution of Security, Knowledge, etc. |

**4.2 - Identify attributes of different application deployment models (private cloud, public cloud, hybrid cloud, and edge):**

* **On-Premises 🡪** This refers to the traditional DC where all equipment is housed on site.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Security is completely controlled by the business.  Devices can be accessed manually if needed. | Rigid.  Takes time to scale, modify or change. |

* **Private Cloud or Infrastructure as a Service 🡪** Provisioned by a single organization or leased by an entity and follows the same principles of cloud deployment with the key distinction of everything is own and house locally.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Provides self-access to provisionally with needed tickets.  Hardware is housed within premises or within reach.  All resources are within the domain of the organization. | Complicated. |

* **Public Cloud 🡪** Provisioned by providers for open utilization by the public and exists on premises of the provider.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Easily accessible.  Pay as you go model.  Flexible.  No responsibility of hardware. | Content is stored on third party servers.  Security is depended on the provider. |

* **Hybrid Cloud 🡪** A combination of both public/private cloud linked via the interface or a WAN circuit. Consisting of On-Prem and Off-Prem hardware/resources

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Delivers large scalability and Flexibility. | Security is a concern.  NOTE: Combined of Public and Private Clouds. |

* **Community Cloud 🡪** deployment dedicated for usage by a community, organizations or businesses such as University campuses.
  + **Advantages and Disadvantages** 🡪 Combination of all of the above.
* **Edge Cloud:** These are cloud deployment closer to where they are consumed such as local retail business or smart cars. These clouds are usually connected to a central cloud for much larger processing capacity and are connected to each other that is what makes them edge cloud.

**4.3 - Identify the attributes of these application deployment types:**

* **Bare Metal 🡪** Functions by deploying the application directly on a physical device.

|  |  |  |
| --- | --- | --- |
| **Advantages** | **Disadvantages** | **Usage Cases** |
| Easily accessible.  Pay as you go model.  Flexible.  No responsibility of hardware. | Content is stored on third party servers.  Security is depended on the provider. | Base infrastructure for hypervisors. |

* **Virtual Machines 🡪** A Virtual environment hosted on a hypervisor.

|  |  |  |
| --- | --- | --- |
| **Advantages** | **Disadvantages** | **Usage Cases** |
| Provides application separation.  Effective usage of available resources.  Easy Backup. | Maybe Complicated.  Licensing may be required for Hypervisor. | Virtual Environments  SDN |
| **Terms** | Hypervisor 🡪 Software that creates VMs and runs on bare metal device.  Tier 1 🡪 Hypervisor  Ter 2 🡪 Applications under an existing application. | |

* **Container Based Infrastructure 🡪** Functions by deploying applications in isolated space within an operating system with all dependencies with only the base operating system being shared.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Solve versioning issues.  Effective usage of hardware space as only one OS is needed.  Support cloud native infrastructure.  Application Stateless. | Complicated |

* **Stateless Computing 🡪** Takes advantage of a modern trend towards applications that are built around services — that is, the application makes a call to another program or workload to accomplish a particular task — to create an environment where applications are made available on an “as needed” basis.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| Quick deployment.  Elastic as the infra is handled by a provider.  Only pay for what is used. | Provider reliant. |

**4.4 - Describe components for a CI/CD pipeline in application deployments:**

The CI/CD is a software development method for developing software system with a focus on automation resulting in order of speed increased, reliability and accurate development. The components are as follows:

* **Continuous Integration 🡪** Is the idea where all developers continually merge change to the main branch of the existing application, thus making changes small and the potential for problems is lower. Therefore, devs are expected to preform extensive testing on the code before mergers MOST LIKELY AUTOMATED.

|  |  |
| --- | --- |
| **Components** | **Examples of a pipeline** |
| Source Code Control   * Version Control Management System * GIT   Build Automation   * Automatic Compilation   Unit Testing   * Automating the testing of individual components of software * Test Driven Development   Branch Merging   * Merging Branches   Integration Testing   * How the individual components come together. | Code Compilation  Unit Test Execution  Static Code Analysis  Integration testing  Packaging and versioning  Publishing the version package to Docker Hub OR  Other package repositories |

* **Continuous Delivery 🡪** : Is the process of preforming development in sprints where the application at any given time is ready for deployment but is not necessary. Therefore, every change is engineered to be self-contained.

|  |  |
| --- | --- |
| **Processes** | **Examples of a pipeline** |
| Utilizes Test Driven Development.  If a bug is discovered all development stops until it is fixed/deployed. | Start with version artifact.  Automatically deploy the version on staging.  Run integration, security, performance, scale, and other tests.  If all test passed mark the built as suitable for production. |

* **Continuous Deployment 🡪** Is the process where the changes are made, tested, integrated with the main branch, and tested again, they are deployed to production in an automated way. Therefore, Continuous Deployment is a special case of Continuous Delivery, in which every build that gets marked as ready for production gets deployed.

|  |
| --- |
| **Processes** |
| Central Repository   * Pushing the code into the centralized repo.   System Testing   * How entire system is tested. * Code has access to all system resources it needs.   Deployment   * Environment Cloning * Infrastructure as Code   User-Acceptance Testing |

* **CI/CD Benefits 🡪**
  + **Integration with agile methodologies 🡪** Agile development is built around the idea of short sprints, after which the developer team delivers a functional application with some subset of the required features. CI/CD does not just work within that framework, it echoes it, with every commit a version of the “deliver a working version of the software” concept.
  + **Shorter Mean Time to Resolution (MTTR) 🡪** Because change sets are small, it becomes much easier to isolate faults when they do occur, and to either fix them or roll them back and resolve any issues.
  + **Automated deployment 🡪** With automated testing and predictable deployment comes the ability to do automated deployments, which means it’s possible to use deployment strategies such as blue-green deployment, in which one set of users gets the new feature set and the rest gets the old. This process enables you to get live testing of the new feature to ensure it’s functioning as expected before rolling it out to the entire user base.
  + **Less disruptive feature releases 🡪** With development proceeding in small chunks that always result in a deployable artifact, it’s possible to present users with incremental changes rather than large-scale changes
  1. **- Construct a Python unit test 🡪**

See python code in the repo.

**4.6 - Interpret contents of a Dockerfile AND 4.7 - Utilize Docker images in local developer environment:**

Docker is a containerization application which wraps several technologies which are as follows:

* **Namespaces:** which isolate different parts of the running container. For example, the process itself is isolated in the pid (process ID) namespace, the filesystem is isolated in the mnt (mount) namespace, and networking is isolated in the net namespace**.**

Graphical user interface, application

Description automatically generated

* **Control Groups or cgroups:** are a standard linux concept that enables the system to limit the resources, such as RAM or storage, used by an application.
* **Union File System or UnionFS:** are file systems that are built layer by layer, combining resources.
* **Docker Architecture**: Uses a Client-Server architecture which utilizes RESTAPI from the client cli to the docker daemon which builds and maintains the containers as follows:

Graphical user interface, diagram, application

Description automatically generated

* **Docker Commands Local Env:**
  + docker build -t dockerfile:version . # building a container from dockerfile
  + docker images # show docker images
  + docker run -d -p 5000:5000 dockerfile # running container -d -> detached, -p port forwarding
  + docker stop <container ID> # stopping a container
  + docker create # Create a container from an image.
  + docker start # Start an existing container.
  + docker run # Create a new container and start it.
  + docker ls # List running containers.
  + docker inspect # Get detailed information regarding the container.
  + docker logs # Print run logs from the container’s execution.
  + docker kill # Stop the main process in the container abruptly.
  + docker rm # Delete a stopped container.
  + docker ps # list running containers
  + docker ps -a # list all past containers
  + docker container rm <NAME> # removing a container
  + docker tag <IMAGE ID> <DOCKER ID> # for pushing the container into docker hub
  + docker push <DOCKER ID> # push
  + docker rmi <IMAGE ID> -f # removing the image locally
  + docker pull <DOCKER ID> # getting the image using docker ID
* **Docker File:** This a simple bash like scrip which builds a container the scrip commands are:
  + FROM #Selects the base image used to start the build process.
  + MAINTAINER #Lets you select a name and email address for the image creator.
  + RUN #Creates image layers and executes commands within a container.
  + CMD #Executes a single command within a container. Only one can exist in a Dockerfile.
  + WORKDIR #Sets the path where the command defined with CMD is to be executed.
  + ENTRYPOINT #Executes a default application every time a container is created with the image.
  + ADD #Copies the files from locally, remotely via a URL into the container’s file system.
  + ENV #Sets environment variables within the container.
  + EXPOSE #Associates a specific port for networking binding.
  + USER #Sets the UID (or username) of the user that is to run the container.
  + VOLUME #Sets up a sharable directory that can be mapped to a local host directory.
  + LABEL #Provides a label to identify the created Docker image.
* **Example:**
  + **# Build docker image with a tag**
  + **docker build -t flaskapi:latest .**
  + **# Run container in detached mode to host port 5000 and docker port 5000**
  + **docker run -d -p 5000:5000 flaskapi:latest**
  + **# Browser to container**
  + **http://127.0.0.1:5000/api/endpoint**
  + **# Show running containers.**
  + **docker ps**
  + **# Show all containers.**
  + **docker -a**
  + **# Push image to Dockerhub**
  + **# Login**
  + **docker login --username 236974534**
  + **# Tag image**
  + **docker tag 36fc4157f1bd 236974534/flaskapi-demo:latest**
  + **# Push image**
  + **docker push 236974534/flaskapi-demo**
  + **4.11 - Utilize Bash commands (file management, directory navigation, and environmental variables):**

Look Up tutorial or something its basic stuff

**4.8 - Identify application security issues related to secret protection, encryption (storage and transport), and data handling AND 4.10 - Describe top OWASP threats (such as XSS, SQL injections, and CSRF):**

**Encrypting Data at rest 🡪** is to encrypt the data in storage so that is understandable without proper credentials.

* **Two-Way Encryption 🡪** is where one encryption key is utilized for both encryption and decryption of data.
* **One-Way Encryption 🡪** is where the encryption key is used to only encrypt data.

**Securing Data in Transport 🡪**

* **SHH 🡪** Provides authentication & encryption between source and target machine, making it difficult or impossible to snoop on your actions
* **TLS 🡪** provides message authentication and stronger ciphers utilized by HTTPS.
* **VPN 🡪** delivers security via encryption entire communication pathway from given endpoints.

**SQL Injection 🡪**

* Command injection via web forms which are build/coded without best practices either by single commands or batch execution.
* db\_datareader, db\_datawriter and db\_owner are the default roles in MySQL.

**SQL Prevention 🡪**

* **Least privilege 🡪** Allow as needed privileges
* **Prepared Statements 🡪** Uses predefined variable comparison instead of direct input execution.
* **Stored procedures 🡪** the SQL code is already stored in a database itself ready to be called by an application.
* **Whitelist Input 🡪** As expected
* **Escaping user-supplied input 🡪** Utilizes escape where the entered query is ended and is not used directly as input for the actual SQL query.

**Cross Site Scripting (CSS or XSS) 🡪** is a client-side code injection attack. The attacker aims to execute malicious scripts in a web browser of the victim by including malicious code in a legitimate web page or web application. The actual attack occurs when the victim visits the web page or web application that executes the malicious code.

* **Prevention 🡪**
  + Sanitizing Data: this means to first make sure the data is in a place where it can’t cause harm such as inside of proper id brackets and such.
  + Limiting Display: this means to not display data which cannot be validated or sanitized properly.

**Cross Site Request Forgery (CSRF) 🡪** is another form of attack with uses pre-authenticated tokens or cookies to use as verification on a different site to gain unauthorized access. This is done via making the user click on a link which runs an embedded script or simply calls the authenticated cookies to be used somewhere else.

* + **Prevention 🡪**
    - Careful with weird links and such.

**Top 10 OWASP 🡪** Cisco Website ( <https://owasp.org/www-project-top-ten/> ) – Memorize

**4.9 - Explain how firewall, DNS, load balancers, and reverse proxy in application deployment:**

* **Firewalls 🡪** are devices which block incoming or outgoing packets depending on the deployment. In application development the following should be kept in mind:
  + Firewalls should keep any outside access to the untested application from happening
  + Firewalls need to be configured in such a way that the application can be appropriately tested. For example, if the application needs to access a development version of a database firewall rules will need to allow that.
  + The environment should be as close a replica of production as possible in order to catch any firewall-related configuration issues as quickly as possible.
* **Load Balancers 🡪** They balance the flow of traffic between several targets/servers, therefore load balancer make decisions based on the following methods:
  + **Persistent session 🡪** if an application requires a user to be logged in then the load balancer will send request to several servers to handle these requests from users.
  + **Round Robin 🡪** The server simply sends the request to the next server on the list. Thus, looping through the entire list or round robin.
  + **Least Connection 🡪** Sends the request to the least busy server or has minimal load.
  + **IP Hash** **🡪** The decision is based on the hash (an encoded value based on IP of the request)
  + **Blue-Green Deployment 🡪** Sends traffic to the new environment (BLUE) rather than making changes to the production environment (GREEN).
  + **Canary Deployment 🡪** The load on the blue environment is increased slowly until a problem is hit, the load then goes back to the old environment.
* **DNS: SAME AS ALWAYS**
* **Reverse Proxy:** A reverse proxy works to make that the response for request are coming from the same server. Therefore, this hides the identity of which server the request is processed by.

**4.12 - Identify the principles of DevOps practices:**

Basic stuff can be memorized but not really needed same principles as read before.

**4.5 - Construct a Python unit test:**

See python code in the repo.

**5.0 - Infrastructure and Automation**

**5.1 - Describe the value of model driven programmability for infrastructure automation:**

* Delivers a unique way to consolidating data gathering from devices.
* Provides a standards-based programmatic method of writing configuration, statistics, and operational data.
* Enables programmability.

**5.2 - Compare controller-level to device-level management:**

**Controllers 🡪** Solves the scalability problem via a centralized approach.

* All interactions are via the controller.
* Utilizes REST API.
* Enables autonomy 🡪 holistic and uniform view of the network.

**Device-level 🡪** Provides physical access and device level configuration

* Time consuming and error prone.
* Requires knowhow of several types of CLI interfaces.
* Restricted to local/physical or access domains.

**5.3 - Describe the use and roles of network simulation and test tools (such as VIRL and pyATS) 🡪 Basic**

**5.4 - Describe the components and benefits of CI/CD pipeline in infrastructure automation:**

* See 4.4 has detail for both sections

**5.5 - Describe principles of infrastructure as code:**

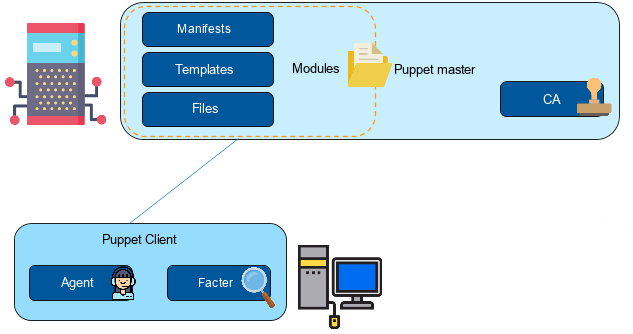
* **Declarative 🡪** With the declarative approach, the desired state of the system is defined and then the system executes all the steps that need to happen to attain the desired state.
* **Imperative 🡪** The imperative approach defines a set of commands that have to be executed in a certain order for the system to achieve the desired state.
* **GitOPs 🡪** Gather info on this

**5.6 - Describe the capabilities of automation tools such as Ansible, Puppet, Chef, and Cisco NSO:**

**Ansible 🡪** Configuration management and orchestration tool, agentless operations where ansible logins via SSH, NETCONF, REST API copy’s code executes with cleanup after.

* **Control Node 🡪**  Ansible Machine.
* **Managed Node 🡪** Network/Server devices that are being managed by ansible.
* **Task 🡪** List of units of actions.
* **Playbook 🡪** YAML files which contains ordered sets of tasks which run in sequence.
* **Inventory OR Host File 🡪** List of all managed nodes grouped together by function/location or etc. However large files have groups\_vars and host\_vars which house specific variable info.
* **Modules 🡪** Python Modules
* **NOTE 🡪 Practice Ansible Further**

**Puppet 🡺** Configuration management and automaton tool based on a client – server architecture utilizing an agent on clients to config/report changes. Additionally, it uses a certificate authority for every client with SSH in a declarative manner where a state is defined, and the process achieves this state**. NOTE: Puppet uses the language DSL**.



**Components of Puppet 🡪**

* **Certificate Authority 🡪** used for internal signing of certificates for clients.
* **Modules 🡪** a collection of manifests, templates, and files:
  + **Manifests 🡪** the code for the client configuration.
  + **Templates 🡪** combines code and data to render a final document. This is mostly generated automatically.
  + **Files 🡪** These are files included/required for process to complete such as an index.html file for an apache server.

**Workings of Puppet 🡪**

* **Authentication 🡪** 
  + Client sends a certificate with id to server.
  + Server signs the certificates and sends it back to the client. (Authentication Complete)\
* **Facts Gathering 🡪**
  + Agents on the clients gather and sends facts to the server.
  + The Master/Server analysis’s these facts and complies a manifest into catalogs where a catalog is a complied/consolidated format of the module.
* **Deployment 🡪**
  + The generated catalogs are sent to the agents on the clients which are installed by said agent.
  + The Agents then send a state of the machine back to the master.

**Chef 🡺** Configuration management like puppet based on a client – server architecture utilizing a declarative model and client agents.

**Chef Components 🡪**

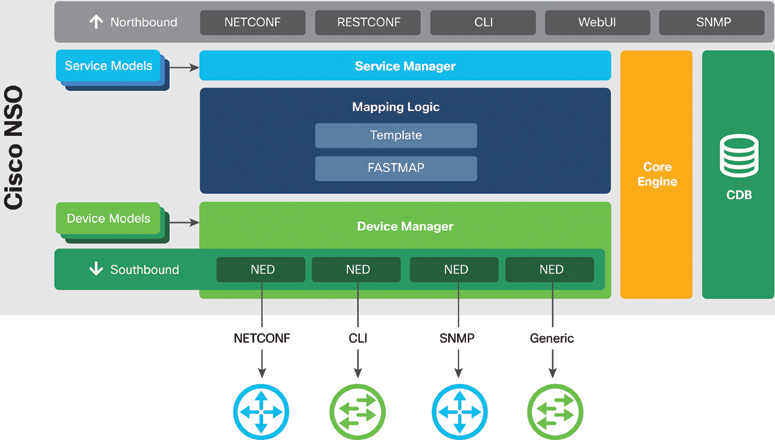
* “**Recipes” 🡪** Ruby Code,
* **“Cookbooks” 🡪** Collection of code,
* **“Knife” 🡪** Executes the code,
* **“Onai” 🡪** Data Gatherer,
* **“Chef client” 🡪** Agent.

**Working of Chef 🡪**

1. Ohai gathers and sends gathered facts to the server.
2. The Workstation accesses these facts and allots a cookbook for a desired state.
3. The Workstation creates cookbooks and recipes which is send to the server.
4. NOTE: if a node fails the server is notified and a cookbook is sent back to rectify this.

**Cisco Network Service Orchestrator (NSO) 🡺** is a multi-vendor supported service orchestration platform which utilizes NETCONF and YANG data models.

**Components of NSO 🡪**



* **Service Manager 🡪** provides an interface for admin to define services as needed to be implemented in the network.
* **Northbound API 🡪** provides a universal API.
* **Device Manager 🡪** provides a layer for device configuration scenarios.
* **Configuration DB or CDB 🡪** houses all device/service configurations.
* **Mapping Logic Layer 🡪** acts as a translation layer for imbedded infrastructure models.
* **Core Engine 🡪** Handles all the major operations such as transactions, high-availability replication, upgrades and downgrades, role-based access control, and rollback management.
* **NOTE: Add more information here potentially**.

**5.8 - Identify the workflow being automated by an Ansible playbook (management packages, user management related to services, basic service configuration, and start/stop):**

ansible.cfg & hosts -> /etc/ansible/

hosts ->

* Define hosts
  + eg:
    - [nsox]
    - 172.16.30.101
* childrens:
  + [childgroup2]
  + host1
  + host2
  + [childgroup1]
  + host2
  + host3
  + [parent1:children]
  + childgroup1
  + childgroup2

group\_vars ->

* + dir for groups
  + yaml structure

host\_vars ->

* + dir for hosts
  + yaml structure

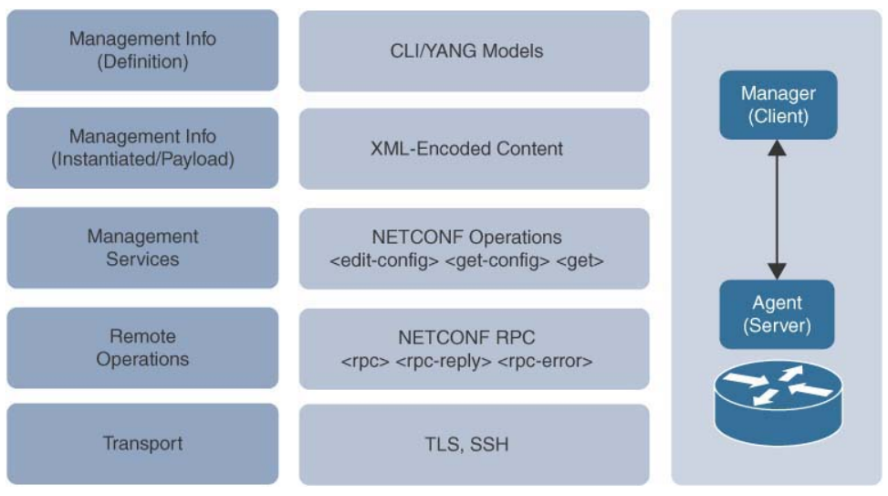
How to run playbooks

- ansible-playbook

**5.9 - Identify the workflow being automated by a bash script (such as file management, app install, user management, directory navigation):**

**Basic Bash Scripting Execution flow**

**5.10 - Interpret the results of a RESTCONF or NETCONF query**

****

**NETCONF Operations:**

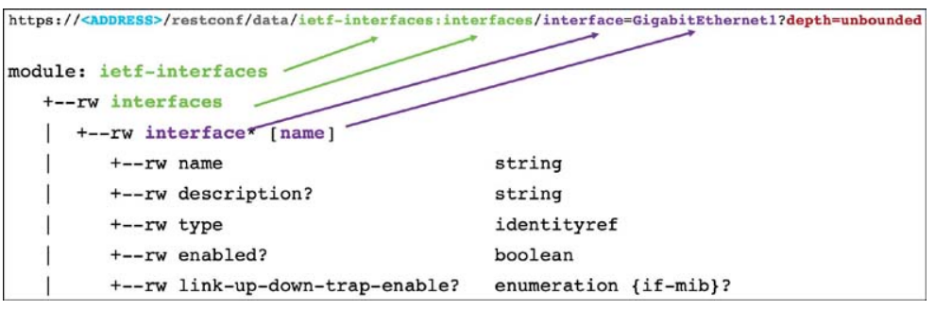
* **<get>** Retrieve running configuration and device state information
* **<get-config>** Retrieve all or part of the specified configuration data store
* **<edit-config>** Load all or part of a configuration to the specified configuration data store
* **<copy-config>** Replace an entire configuration data store with another
* **<delete-config>** Delete a configuration data store
* **<commit>** Copy the candidate data store to the running data store
* **<lock> / <unlock>** Lock or unlock the entire configuration data store system
* **<close-session>** Gracefully terminate the NETCONF session
* **<kill-session>** Forcibly terminate the NETCONF session

**Three types of configuration data stores are defined in NETCONF:**

* **running:** This data store holds the complete configuration currently active on the network device. Only one running data store can exist on a device, and it is always present.
* **candidate:** This data store acts as a workplace for creating and manipulating configuration data.
* **startup:** This data store contains the configuration data that is loaded when the device boots up and comes online.

**RESTCONF to NETCONF Operations:**

* **GET <get>, <get-config>**
* **POST <edit-config> (operation=“create”)**
* **PUT <edit-config> (operation=“create/replace”)**
* **PATCH <edit-config> (operation=“merge”)**
* **DELETE <edit-config> (operation=“delete”)**

****

**5.11 - Interpret basic YANG models:**

**YANG Built-in Data Types 🡺** binary, bits, boolean, decimal64, empty, enumeration, int& unit 8/16/32/64, string.

**Statements 🡺**

* **Typedef 🡪** defines derived types from base type.
* **Inet-yang-types 🡪** data types are organized into these modules.
* **import 🡪**  this statement is used to make all these data types available exp. import ‘inet-yang-types’

**YANG four types od nodes for data modeling 🡺**

* **Leaf nodes 🡪** has one value of a specific type.
* **Leaf-list nodes 🡪** a series of leaf’s nodes of a specific type.
* **Container nodes 🡪** used to group related nodes into a subtree.
* **List nodes 🡪** defines a sequence of list entries.

**Components of a YANG Model 🡺**

**Graphical user interface, text, application

Description automatically generated**

**Header 🡪** contains the namespace & prefix and more.

**Import & Include 🡪** import is used to refer to an external module whereas include is used to pull the main body of the module. A module can be split into multiple files.

**Type of definitions 🡪** this section defines any extra data types needed to build the model.

**Configuration & Operational Data… 🡪**  this include all the config for the module.

**Action RPC… 🡪** this is an optional to define custom RPC notifications.

**5.12 - Interpret a unified diff**

**Text

Description automatically generated**

* The ----- and +++ 🡺 the files where the data was removed and where it was added.
* @@ -42, 9 🡪 from file with 9 lines starting from 42… @@ +42, 8 🡪 to file with 8 lines starting from 42.

**Find better examples 🡪 20-3 demo-interpreting a unified diff**

**5.13 and 5.14**

**Refer to do**

**NOTE: For 5.7, 5.8, 5.9, 5.10, 5.11 = Practical and links with all of 3.0, SEE CODE**

**6.0 - Network Fundamentals**

**Review CCNA/CCNP Notes and view section 6 in “Devnet Associate Notes File”**

**0.0 – Extra Notes**

Descriptions of the functions:

cli() Returns the raw output of CLI commands, including control and special characters.

clip() Prints the output of the CLI command directly to stdout and returns nothing to Python.

clid() Returns a dictionary of attribute-names/values for CLI commands that support JSON

**Custom Tokens 🡪** A custom token allows a user to enter his or her username and password once and receive a unique auto-generated and encrypted token.