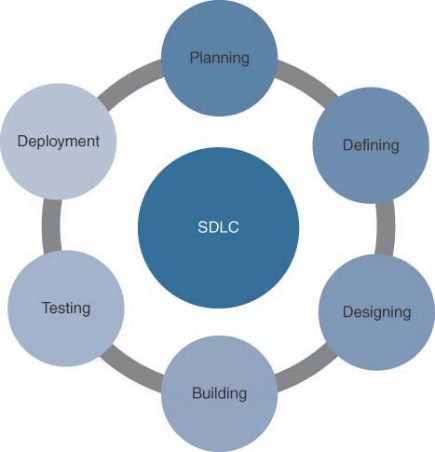
**1.0 - Software Development and Design (1.1 – 1.8)**

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

A Software Design Lifecycle is used to prevent problems from slapdash approaches providing sanity via guidelines on building software packages. The stages of a SDLC are as follows:



The SDLC are as follows:

**Waterfall:**

* This methodology is very sequential and linear, based on a clear scope from the start and tackling each step-in order. This means that as each one of the eight Waterfall stages are reached, the developers move on to the next step.
* The narrow and linear nature of Waterfall makes it great for simple, unchanging projects. It is easy to manage, with each step having specific deliverables and clearly delineated start and finish points.
* The idea is that development does not start until all the scoping out actions have been reached and this helps projects to deliver on time.
* Waterfall processes struggle with change. They struggle because they are so sequential and usually, after the completion of a phase, it is challenging and costly to go back and change anything. Also, because development does not start until the first few phases are finished, there won’t be any working software for stakeholders to evaluate until late in the process.

Diagram, text

Description automatically generated

Advantages

* With a waterfall method, the client knows what to expect. They will have a reasonable idea of the size, cost, and timeline for the project. They will have a definite idea of what their program will do in the end
* In staff turnover matters, waterfall’s reliable documentation allows for minimal project impact.
* In waterfall projects, progress is more easily measured, as the full scope of the work is known in advance.

Disadvantages

* Waterfall methods rely massively on initial requirements. However, if these requirements are wrong in any manner, the project is, in technical terms, doomed
* The entire product is only tested at the end. If bugs are written early in the code, but discovered late, their existence may have affected how another code was written. This could also lead to the ‘doom’ scenario.

When to use it:

* 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 organisation isn’t in a mature enough state to adopt an agile approach

**Agile:**

* Agile approaches are designed with flexibility at heart. Developers start with a simplistic project design, and then begin to work on small modules. The work on these is performed in sprints (weekly or monthly). At the end of each sprint, priorities of the project are evaluated, and tests are run. These weekly or monthly sprints allow for bugs to be identified, and client feedback to be incorporated into the design before the next sprint is run.
* The goal of each iteration is to create a working product, which can be displayed to stakeholders. Feedback can then be included in the next iteration or future ones.
* It is excellent for projects where the end-goal may not be clearly defined from the start and can evolve as it develops. With so many cycles, clients can give continuous user-based feedback.
* Agile Methods:
  + Agile Scrum: focuses on small, self-organizing teams that meet daily for short periods and work in iterative sprints, constantly adapting deliverables to meet changing requirements.
  + Lean: Below
  + Extreme Programming (XP): Is more prescriptive about software engineering best-practices, and more deliberately addresses the specific kinds of quality-of-life issues.
  + Feature-Driven Development (FDD): An overall model, broken out, planned, designed, and built feature-by-feature.
* Agile Sprints:
  + Functions on quick iterations of an SDLC which are called sprints an range from 2 to 4 weeks and at the end of the sprint the features or deliverables decided beforehand are considered “Deliverable”.
* Agile Backlog:
  + This is a list of needed/wanted features in a prioritized list developed by the product owner which are discovered in the “Requirement & Analysis Phase”.
* Agile User Stories:
  + When a feature gets to the top of the backlog list it is broken down into smaller tasks or “User Stories”. Which are small enough to be completed within a single sprint cycle. A template is as follows:
    - As a <user|role>, I would like to <action>, so that <value|benefit>
* Agile SCRUM Teams:
  + Are a small team of people which are cross-functional, collaborative, self-managed and self-empowered, and the whole team is held accountable for the completion of the user story.
    - Standup Meetings: These are 15 mints daily meetups held by the Scrum Master whose job is to report and/or help remove obstacles.

Advantages

* The testing conducted at the end of each sprint ensures that bugs are caught and taken care of in the development cycle. They won’t be found at the end.
* Because the Agile methods allow you to make changes, it’s easier to keep your product up-to-date with the latest technological advancements.

Disadvantages

* As the original project does not have a definitive plan, the final product can be different than what was initially intended.

When to use it

* 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**:

* By only focussing on what’s necessary, Lean methods are often able to speed up the process, reducing project time and cost by reducing wasteful notions.
* With Waterfall and Agile methods, the client/stakeholder/end user receives the product at the end of the project. However, with a Lean methodology, you are mostly focusing on building a tiny subset of features and getting that Minimal Viable Product (MVP) to your clients and the market a lot sooner than with Waterfall and Agile methods.
* The quicker delivery helps Lean remove potential waste as you will quickly identify if you are building the right product or not. If you are not, then you can abandon ship sooner thus saving two precious project resources: time and money.
* Lean Principles:
  + Eliminate Waste: which is anything that does not add value to the customer as defined by the customer.
  + Amplify learning: which is short frequent working software development which helps developers learn and fine tune software components effectively.
  + Decide as late as possible: when uncertain decide to delay the decision and make it later based on more developed facts.
  + Deliver as fast as possible: Self-explanatory.
  + Empower the team: Allowing the team to make decision based on their expertise.
  + Build integrity: The deliverable maintains its usefulness to the customer.
  + Optimize the whole: ensure decisions made are positive and does not have unintended consequence.

Disadvantages

* For the Lean method to work, everyone involved in development should follow the process, from project manager to developer. If anyone of them does not, there can be a big impact on productivity.

Advantages

* Lean uses fewer resources to build an MVP which considerably reduces the overall costs.
* Lean helps company’s management to focus on smart work rather than hard work by making strategic plans for the future.
* Elimination of waste: If something doesn’t add value to the final product, get rid of it. There is no room for wasted work.
* Just-in-time: Don’t build something until the customer is ready to buy it. Excess inventory wastes resources.
* Continuous improvement (Kizan): Always improve your processes with lessons learned and communication.

When to use it

* Lean methods often work best in small projects with a short time frame, as Lean teams are usually quite small, and bigger projects require multiple teams. While not impossible, this can be difficult to manage effectively on a larger, more complex project.

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

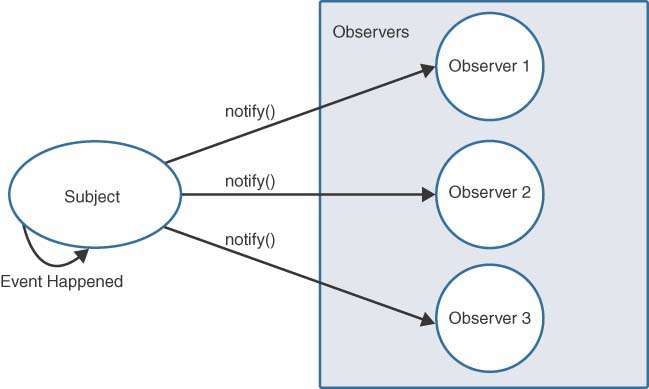
Design patterns are a set of rules to write code, it delivers a map or guidelines to develop code.

Singleton DP:

* Pattern that restricts classes to only being instantiated once during the execution of the program.
* Meaning that the program creates only one instance of a given class for the entirety of the program's execution.
* This is used for example establishing connections to Databases or Servers which are one time only.
* To access values in this patten a “get\_instance()” method is used.

Observer DP:

* A pattern where dependent objects are updated or modified by one or more subject objects.
* The benefit of the observer design pattern is that observers can get real time data from the subject when a change occurs.
* Two classes: subject and Observer
* Subject: attach, detach and modify, it refers to the object state being observed—in other words, the data that is to be synchronized.
* Observer: update and allows the component that registers with the subject to allow the subject to be aware of the observer and how to communicate to it.
  + Only have one function defined.



MVC (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
  + View: Component that displays data
  + Controller: Component that handles logic and directs model and View

Diagram

Description automatically generated

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

Functions vs methods

* Function a block of code assigned to a symbol.
* Methods are functions defined as a part of a class invoked by instance of that class.

OOPS

* classes and objects

Modules

* import

Refactoring the Code:

* Cleaning the code without adding more functionality

Inheritance

* Inheritance allows us to define a class that inherits all the methods and properties from another class. Parent class is the class being inherited from, also called base class. Child class is the class that inherits from another class, also called derived class.
* Basically, to inherent from an existing class the second class should use the first-class's name as a parameter. Therefore, when the second class is created it uses the methods defined in the parent class.

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

Plain Text:

* + Human readable
* + Open and extensible
* + Self-describing
* + Long lifespan
* - Computer Readable
* Example: READ ME files

XML (Extensible Markup Language):

Comparative:

* + Legacy Support
* + Supported by SOAP which is web-based protocol.
* - Long

Structure:

**tags**

- < Person Id="2"></Person>

root tag

**prolog**

- starting tag <?xml version="1.0" encoding="UTF-8" ?>

**attributes**

- < Person Id="2"> here Id is attribute

**elements**

- < Person Id="2">{{elements}}</Person>

**empty elements**

- <EmptyElement><EmptyElement/> OR Simply <EmptyElement/>

**Comments**

* <!-- Comment -->

XML Document Object Model

tree like structure

- siblings, parents, childs

JSON:

Comparative:

* + popular
* + lightweight
* + native to JS
* + compressed to a single line, whitespace does not have any meaning.
* - short
* - structure matters in terms of { } [ ] , .

Structure (Example):

* Nested Objects such as dictionaries, lists, booleans.

YAML:

Comparative:

* + human friendly
* + compact but not as much as JSON as the files structure matters
* + ideal for configuration
* - whitespace matters

Structure:

* Indentation level
* no inverted commas needed though accepted.
* starts with --- ends with ...

**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:**

Python Parsing Data:

* open () function is used to open files with anyone of the following:
  + r: Open for reading (default)
  + w: Open for writing, truncating the file first.
  + x: Open for exclusive creation, failing if the file already exists.
  + a: Open for writing, appending to the end of the file if it exists.
  + b: Open in binary mode
  + t: Open in text mode (default)
  + +: Open for updating (reading and writing)
* CSV:
  + Read:
    - with open (“file.csv”, “r”) as data:
      * csv\_list = csv.reader(data)
  + Write:
    - with open (“file.csv”, “w”) as data:
      * data\_writer = csv.writer(data, OPTIONS)
      * data.writer.writerow (DATA)
* JSON:
  + load(): This allows you to import native JSON and convert it to a Python dictionary from a file.
  + loads(): This will import JSON data from a string for parsing and manipulating within your program.
  + dump(): This is used to write JSON data from Python objects to a file.
  + dumps(): This allows you to take JSON dictionary data and convert it into a serialized string for parsing and manipulating within Python.
* XML:
  + Load file same as JSON but converted to a data structure via xml\_dict= xmltodict.parse(xml\_example).
  + xml.etree.ElementTree
  + lxml etree
  + xmltodict
* YAML:
  + File is read via the open() function and parsed via the .read method. However, the data itself is parsed and converted via the yaml.load () method.

Python Error Handling:

* The sequence of error handling statements are as follows:
  + Try
  + Except
  + Else
  + Finally
  + TEEF

Python Modules:

* **pprint**: The pretty print module is a more intelligent print function that makes it much easier to display text and data by, for example, aligning data for better readability. Use the following command to import this module.
* **sys**: This module allows you to interact with the Python interpreter and manipulate and view values. Use the following command to import this module.
* **os**: This module gives you access to the underlying operating system environment and file system. It allows you to open files and interact with OS variables. Use the following command to import this module.
* **datetime**: This module allows you to create, format, and work with calendar dates and time. It also enables timestamps and other useful additions to logging and data. Use the following command to import this module.
* **time**: This module allows you to add time-based delays and clock capabilities to your Python apps. Use the following command to import this module.
* **xmltodict**: This module translates XML-formatted files into native Python dictionaries (key/value pairs) and back to XML, if needed. Use the following command to install this module.
* **csv**: This is a standard library module for understanding CSV files. It is useful for exporting Excel spreadsheets into a format that you can then import into Python as a data source. It can, for example, read in a CSV file and use it as a Python list data type. Use the following command to import this module.
* **json**: This is a standard library module for reading JSON-formatted data sources and easily converting them to dictionaries. Use the following command to import this module.
* **PyYAML**: This module converts YAML files to Python objects that can be converted to Python dictionaries or lists. Use the following command to install this module.
* **pyang**: This isn’t a typical module you import into a Python program. It’s a utility written in Python that you can use to verify your YANG models, create YANG code, and transform YANG models into other data structures, such as XSD (XML Schema Definition). Use the following command to install this module.
* **requests**: This is a full library to interact with HTTP services and used extensively to interact with REST APIs. Use the following command to install this module.
* **ncclient**: This Python library helps with client-side scripting and application integration for the NETCONF protocol. Use the following command to install this module.
* **netmiko**: This connection-handling library makes it easier to initiate SSH connections to network devices. This module is intended to help bridge the programmability gap between devices with APIs and those without APIs that still rely on command-line interfaces and commands.
* **pysnmp**: This is a Python implementation of an SNMP engine for network management. It allows you to interact with older infrastructure components without APIs but that do support SNMP for management.
* **napalm**: napalm (Network Automation and Programmability Abstraction Layer with Multivendor Support) is a Python module that provides functionality that works in a multivendor fashion.
* **nornir:** This is an extendable, multithreaded framework with inventory management to work with large numbers of network devices.
* **unittest:** This standard library testing module is used to test the functionality of Python code.
* **pyats:** This module is a testing framework for code validation provided by Cisco.

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

* **Code Review/Reducing Errors**: Version control systems allow you to compare files, identify differences, and merge the changes if needed prior to committing any code.
* **Identifiability**: It is possible to link documents to decisions, contributions, contributors, and time. All documents should also be identifiable by filename, author, and date.
* **Clarity**: Multiple versions of documents can be distinguished, and the latest version identified.
* **Reduced Duplication**: Helps to reduce potentially out of date/misleading copies.
* **Collaboration capabilities**: Multiple people can work on a project (a set of files) at the same time without overriding each other's changes.
* **Accountability and visibility**: Know who made what changes when they made it and (hopefully) why.
* **Isolation for a work environment**: Build new features independently without affecting the existing software.
* **Safety with backup and restore**: Files can be reverted when a mistake is made.
* **Work anywhere**: Files are stored in a repository, so any device can have a working copy.

Version Control Systems:

* **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**: A process where the code is check line by line by a reviewer in a series of meetings until a consensus is reached on the state of the code.
* **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.

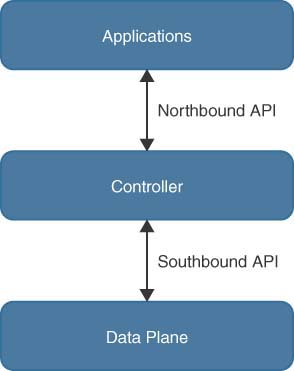
Testing:

* **Functional Testing**: This testing seeks to determine if the developed software works correctly based on the functions it preforms such as database connections, correct-input acceptance and so on.
* **Non-Functional Testing**: This testing examines usability, performance, security, resiliency, compliance, localization, and many other issues.
* **Unit-Testing:** Detailed functional testing where small pieces of code are developed and tested.
* **Integration Testing**: This means to build specific pieces of code within the software to support testing, such as functions which test other functions and methods which can be run by libraries 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**

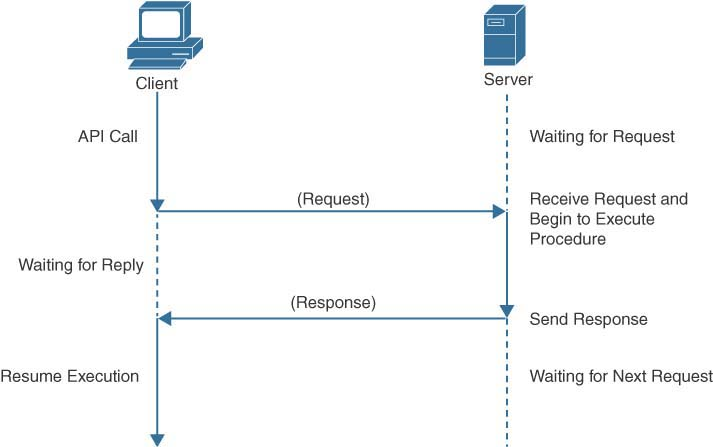
General Information:

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

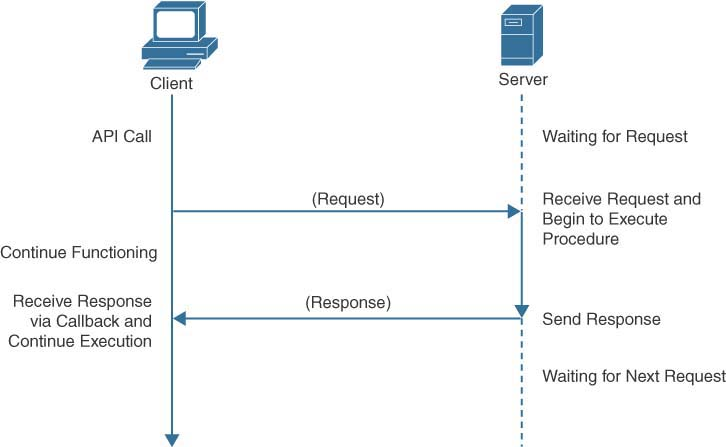


**2.8 - Compare common API styles (REST, RPC, synchronous, and asynchronous**

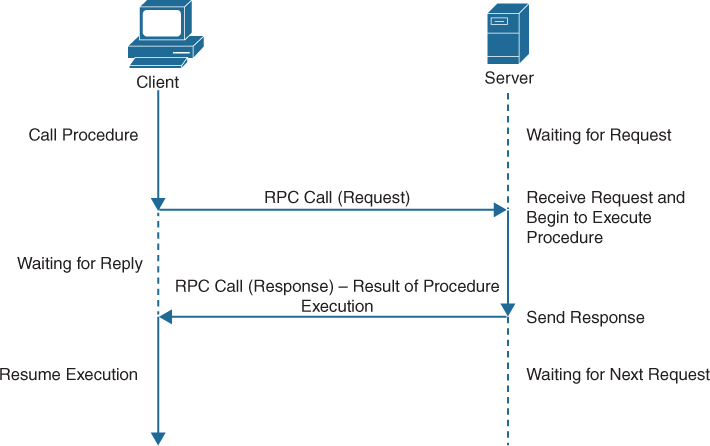
* Synchronous: If an API call is synchronous, it means that code execution will block (or wait) for the API call to return before continuing.



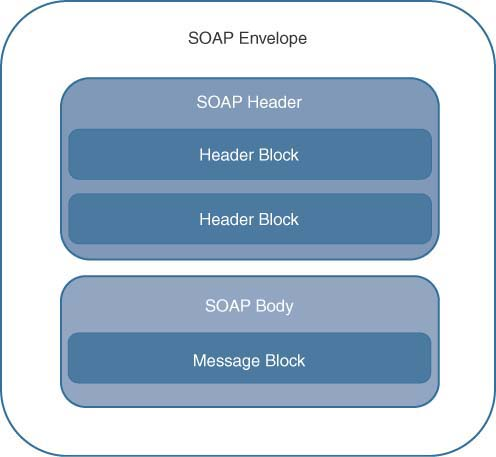
* Asynchronous: Asynchronous calls do not block (or wait) for the API call to return from the server. Execution continues in your program, and when the call returns from the server, a "callback" function is executed.

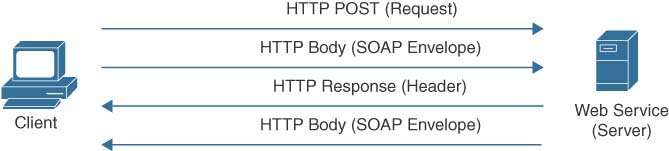


* Remote Procedure Calls (RPC): allows for remote execution of code on a remote node in a network where the behavior is as if the code was executed locally. This functions as a “Synchronous” call where the client is blocked until the reply from the server is received. RPC uses XML-RPC



* 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: is a root message which defines the namespace and highlights the document as a SOAP message.
  + Header: is an optional element but if present must if the first child of the envelope element.
  + Body: contains the data to be transported which must be in XML format and its own namespace.
  + Fault: is an optional element but if present is a child of the body. The fault provides error and/or status information.
    - SOAP Fault Codes in Book Chapter 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. Additionally, REST consists of the following constraints:
  + 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 interface must adhere to the following principle:
    - 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.

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

* REST API Requests: Are essentially HHTP request which follow the REST principles and are made of 4 components:
  + Uniform Resource Identifier/locator (URI): identifies which resource for a REST API and is usually a part of the URI and consists of the following:
  + 
  + HTTP Methods: REST uses HTTP CRUD verbs which are:
    - Table

      Description automatically generated
  + Header: REST uses standard HTTP headers to communicate additional information between client and servers which are formatted as name-value pairs separated by colons ( : ). Common headers:
    - Request Header: include additional information that does not relate to the content of the message.
    - Entity Header: include additional information which describes the content of the body.
  + Body: this contains the APR request data pertaining the resources that the client wants to manipulate. Example Methods POST PUT and PATCH.
* REST API Response: Are HTTP responses which communicate the result of a client’s HTTP request. This may contain data which was requested or even inform the client about a problem. Responses have 3 major components:
  + Section 2.6 Identify the parts of an HTTP response (response code, headers, body)

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

* Authentication: is the act of verifying the user’s identity.
* Authorization: is the process of proving the user has permission to perform a given action or access.
* REST API Authentication Mechanism:  
  + Basic Authentication: This is basic clear text authentication which is included inside of the HTTP Header encoded in base64.
  + Bearer or Token Authentication: Use 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: Are unique alphanumeric strings generated via a server assigned to a user which is obtained usually through login credentials. These keys should be used with HTTPS and consists of the two types:
    - Public API Keys: Are shareable and enables the user to access a subnet of data via query strings.
    - Private Keys: Are kept secret.

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

* See Python Code in Section 2.0

**2.3 - Identify the constraints when consuming APIs:**

* Representational State Transfer (REST): is an architecture for passing information between two parties via several protocols with HTTP/HTTPS being the most common. Additionally, REST consists of the following constraints:
  + 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 interface must adhere to the following principle:
    - 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.

**2.2 - 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 be running at all times 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.

**2.5 - Troubleshoot a problem given the HTTP response code, request and API documentation:**

* Client-Side Errors:
  + Invalid URI: {Invalid URL exception}
  + Wrong Domain Name: {Error 8 nodename nor server provided not known}
  + Certificates: {requests.exceptions.SSLError}
* Server-Side Errors:
  + Server Down/Issues: Long delays before {Error 60 Operation timed out}
* HTTP-Status Codes: HTTP/1.1 standard has 5 categories which are:
  + 1xx: Informational: Request received, continuing to process.
  + 2xx: Success: The action was successfully received, understood, and accepted.
  + 3xx: Redirection: Further action must be taken in order to complete the request.
  + 4xx: Client Error: The request contains bad syntax or cannot be fulfilled.
    - 400 – Bad Request: Request could not be understood.
    - 401 – Unauthorized: Authentication failed.
    - 403 – Forbidden: Authentication passed but authorization failed.
    - 404 – Not Found: Server not found.
    - 405 – Method Not Allowed: Request was recognized by the server method requested was rejected.
    - 406 – Not Acceptable: The target resource does not have a way to represent the requested resource.
    - 407 – Proxy Authentication Required: An authentication is requited with the proxy used.
    - 409 - The request could not be completed due to a conflict with the current state of the target resource.
    - 415 – Unsupported Media Type: Media sent by the client is not supported.
  + 5xx: Server Error: The server failed to fulfill an apparently valid request.
    - 500 – Internal Server Error: Server encountered an unexpected error.
    - 501 – Not Implemented: Server does not support the function.
    - 502 – Bad Gateway: Received an invalid response from an inbound server.
    - 503 – Service Unavailable: Server is unable to handle the request.
    - 504 – Gateway Timeout: The gateway did not receive a timely response.

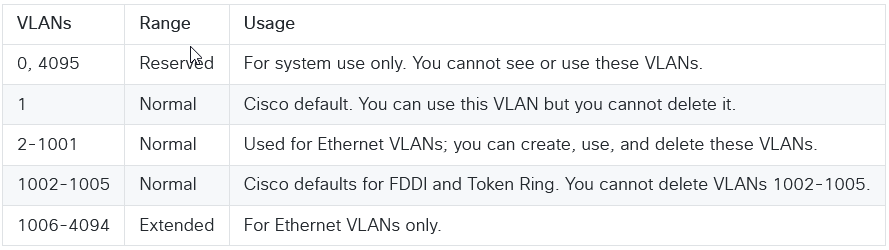
**2.6 - Identify the parts of an HTTP response (response code, headers, body):**

|  |  |
| --- | --- |
| **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** |  |

**6.0 - Network Fundamentals:**

Review CCNP Notes and CCNA Notes but below are general notes and areas which are included.

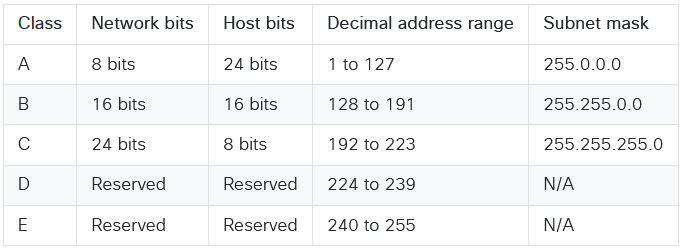
**6.1 - Describe the purpose and usage of MAC addresses and VLANs:**

* Ethernet: The basic structure of a ethernet frame is as follows:
  + 
  + Preamble: 7 Bytes and used to synchronize the signals of the communicating computers.
  + SFD (Start of frame delimiter): 1 Byte and used to mark the end of the preamble.
  + DA: 6 Bytes and houses the destination MAC address
  + SA: 6 Bytes and houses the Source MAC address
  + Type: Houses code that identifies the network layer protocol
  + Data and pad: Max 64 Bytes and houses data/padding if required.
  + FCS: The Frame Check Sequence checks if the data has been transmitted without corruption.
* MAC Address:
  + Layer two identifiers
  + 12 hexadecimals
  + 24-bit OUI and 24-bit Unique identifiers (Total MAC address = 48-Bits or 6 Bytes)
  + Types of communication:
    - Unicast = 1: 1
    - Broadcast = 1: ALL
    - Multicast = 1: MANY
* Switching: Basic From CCNA
* VLANs: Virtual Local Area Network
  + Breaks the broadcast domain
  + Layer 2 feature
  + Used IEEE 802.1Q standard
  + 

**6.2 - Describe the purpose and usage of IP addresses, routes, subnet mask / prefix, and gateways:**

CCNA Notes ---> File = Video NOTES (Layer 3 Technologies and Some Review of Layer 2

BELOW IS SUMNMARY

* IP addresses:
  + Layer 3 identifier
  + IPv4 = 32 Bits or 4 Octets 8 bits each
  + 
  + IPv6 = 128 Bits or 16-bit Hexadecimal
    - No address Class
    - No subnetting
    - NOTE: Review CCNA Notes at Video NOTES (WAN & Other Protocols & IPv6) “Page 37 to 44”
* Routes
  + Gain information of distant networks
* Subnet mask / prefix
  + Identifies a network.
  + Network and Host network
* Gateways
  + where to transfer the packets.
  + Default routes.

**6.3 - Describe the function of common networking components (such as switches, routers, firewalls, and load balancers):**

* Switches:
  + L2 Switching
  + Switch Operations:
    - **Cut-Through Switching Mode**: Started to forwards after reading the destination MAC address even before the complete frame arrives.
    - **Store-and-Forward Switching Mode:** Forwards Frame after receiving the complete frame and checking the CRC.
* Routers:
  + L3 Routing
  + Routing Operations:
    - **Process switching:** Each packet is looked up in the routing table = SLOW.
    - **Fast switching:** Utilizes a cache for known destination = Faster
    - **Cisco Express Forwarding (CEF):** Builds the cache based on the routing table inside of building it dynamically = Fastest.
* Firewall:
  + Default is to block all packets
  + Type’s:
    - **Stateless packet-filtering:** Functions on permit or deny ACLs.
    - **Stateful packet-filtering:** Same as stateless packet-filtering firewall but also keeps track of the connection state.
    - **Application Layer Packet-Filtering:** Deep inspection of the packet occurs all the way up to the OSI model’s Layer 7.
    - New Gen Fire Wall (L5)
* Load balancers:
  + Balance data load between servers/devices
  + Provides Networks With:
    - Device redundancy — High availability, which allows you to set up a peer load balancer device in the configuration so that if one load balancer becomes inoperative, the other load balancer can take its place immediately.
    - Scalability — Virtualization, which allows running the load balancers as independent virtual devices, each with its own resource allocation.
    - Security — Access control lists, which restrict access from certain clients or to certain network resources.
  + Provide Applications With:
    - High services availability — High-performance server load balancing, which allows to distribute client requests among physical servers and server farms, and provide health monitoring at the server and server farm levels through implicit and explicit health probes.
    - Scalability — Virtualization, which allows the use of advanced load-balancing algorithms (predictors) to distribute client requests among the virtual devices configured in the load balancer. Each virtual device includes multiple virtual servers. Each server forwards client requests to one of the server farms. Each server farm can contain multiple physical servers.
    - Services-level security — Allows to establish and maintain a Secure Sockets Layer (SSL) session between the load balancer and its peer, which provides secure data transactions between clients and servers.

**6.4 - Interpret a basic network topology diagram with elements such as switches, routers, firewalls, load balancers, and port values**

**CCNA Notes otherwise straight forward**

**6.5 - Describe the function of management, data, and control planes in a network device**

* Management Plane:
  + Managing network devices
  + SNMP, NETCONF, SSHn
* Data Plane:
  + Actually, do forwarding
* Control Plane:
  + How to forward
  + Routing protocols
  + MAC address table

**6.6 - Describe the functionality of these IP Services: DHCP, DNS, NAT, SNMP, NTP:**

* DHCP (Dynamic Host Configuration Protocol):
  + Handing out IPs to the devices in the network
  + Uses UDP
  + Port 67: Used by the server
  + Port 68: Used by the client
* DNS (Domain Name Server):
  + Conversion of domain name to IP addresses
  + Used UDP
  + Port 53
  + Records:
    - A: Known as a DNS host record, stores a hostname and its corresponding IPv4 address
    - AAAA: Stores a hostname and its corresponding IPv6 address.
    - CNAME: Can be used to alias a hostname to another hostname. When a DNS client requests a record that contains a CNAME, which points to another hostname.
    - MX Records: Specifies an SMTP email server for the domain, used to route outgoing emails to an email server.
    - NS Records: Specifies that a DNS Zone.
* NAT (Network Address Translation):
* Converts local IPs to routable public IP addresses
* SNMP (Simple Network Management protocol):
  + Used for management of network devices
  + Port: 161
* NTP (Network Time Protocol):
  + Used for clock sync b/w systems
  + Stratum levels
  + Uses UDP
  + Port 123

**6.7 - Recognize common protocol port values (such as, SSH, Telnet, HTTP, HTTPS, and NETCONF):**

* SSH: 22
* Telnet: 23 or 992
* DNS: 53
* HTTP: 80
* HTTPS: 443
* NETCONF: 830
* RESTCONF: 8008, 8080, 8888

**6.8 - Identify cause of application connectivity issues (NAT problem, Transport Port blocked, proxy, and VPN):**

NAT problem, Transport Port blocked, proxy, and VPN

Firewalls, IPS, ACLs

MTUs

Loopback Interfaces

Insecure Protocols (HTTP, Telnet)

**6.9 - Explain the impacts of network constraints on applications: Revise CCNP/CCNA Notes for IP Network Issues**

**4.0 - Application Deployment and Security:**

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

There are several application deployment methods with unique advantages and disadvantages these are as follows:

* **Bare Metal:** Functions by deploying the application directly on a physical device.
  + **Advantages:** 
    - Specific/unique physical hardware access.
    - Applications have access to high performance computing.
  + **Disadvantages:**
    - Less Flexible
    - Multiple applications do not have separation.
  + **Usage Cases:**
    - Base infrastructure for hypervisors.
* **Virtual Machines:** A Virtual environment hosted on a hypervisor.
  + **Advantages:**
    - Provides application separation.
    - Effective usage of available resources.
    - Easy Backup.
  + **Disadvantages:**
    - Maybe Complicated.
    - Licensing may be required for Hypervisor.
  + **Usage Cases:**
    - 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:**
    - Solve versioning issues.
    - Effective usage of hardware space as only one OS is needed.
    - Support cloud native infrastructure.
    - Application Stateless.
  + **Disadvantages:**
    - 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:**
    - Quick deployment.
    - Elastic as the infra is handled by a provider.
    - Only pay for what is used.
  + **Disadvantages:**
    - Provider reliant.

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

There are serval methods of deploying infrastructure some of the most common and adopted are as follows:

* **On-Premises:** This refers to the traditional DC where all equipment is housed on site.
  + **Advantages:**
    - Security is completely controlled by the business.
    - Devices can be accessed manually if needed.
  + **Disadvantages:**
    - Rigid.
    - Takes time to scale, modify or change.
* **Private Cloud or Infrastructure as a Service:** Is 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:**
    - 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.
  + **Disadvantages:**
    - Complicated.
* **Public Cloud:** Provisioned by providers for open utilization by the public at large and exists solely on the premises of the provider.
  + **Advantages:**
    - Easily accessible.
    - Pay as you go model.
    - Flexible.
    - No responsibility of hardware.
  + **Disadvantages:**
    - Content is stored on third party servers.
    - Security is depended on the provider.
* **Hybrid Cloud:** Is a combination of both the public and private clouds linted/connected via the internet or WAN network. Usually consists of both On-Prem and Off-Prem hardware/resources.
  + **Advantages:**
    - Delivers large scalability and Flexibility.
  + **Disadvantages:**
    - Security is a concern.
    - NOTE: Combined of Public and Private Clouds.
* **Community Cloud:** This is a cloud deployment dedicated for usage by a community or organizations or business 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.

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

* Moving the data processing and storage closer to the user
* CDN : Content Delivery Network 🡪 is a highly distributed platform of servers that helps minimize delays in loading web page content by reducing the physical distance between the server and the user. Example of Edge Compute
* Pros:
  + Speed/Latency
  + Privacy
  + Resiliency
  + Scalability
* Cons:
  + Resource Requirements
  + Infrastructure Complexity
  + Distribution of Security, Knowledge, etc

**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 two 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.
  + **An example pipeline may have for CI:**
    - Code Compilation
    - Unit Test Execution
    - Static Code Analysis
    - Integration testing
    - Packaging and versioning
    - Publishing the version package to Docker Hub or other package repositories
  + **Components:**
    - 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.
* **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.
  + **An example pipeline may have for CD**:
    - 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
  + **Processes:**
    - Utilizes Test Driven Development.
    - If a bug is discovered all development stops until it is fixed/deployed.
* **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.
  + **Components:**
    - 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.

**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 then 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.

**4.6 - 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):**

Securing Data:

* Encrypting Data is to encrypt the data in storage so that its un-understandable without proper credentials.
  + Two-Way Encryption: is where one encryption key can be used to both encrypt and decrypt data.
  + One-Way Encryption: is where the encryption key is used to only encrypt data.
* Securing in Transport:
  + SSH: provides for authentication and encryption of messages between the source and target machines, 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 from 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

**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:**

* Replaces traditional vendor specific CLIs.
* Delivers a unified way to consolidating data gathered 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: They solve the scalability problem via a centralized approach.
  + All interactions are via the controller to device.
  + Provides an abstraction layers eliminating vendor locking.
  + Uses REST API by may differ.
  + Provides atomicity = holistic and uniform view of the network.
  + Supports multiple types of devices and equipment.
* Device-Level: provides physical access and device level configuration
  + Time consuming and can be error prone.
  + Requires knowhow of server 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 stuff

**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:**

* Same stuff
* 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:

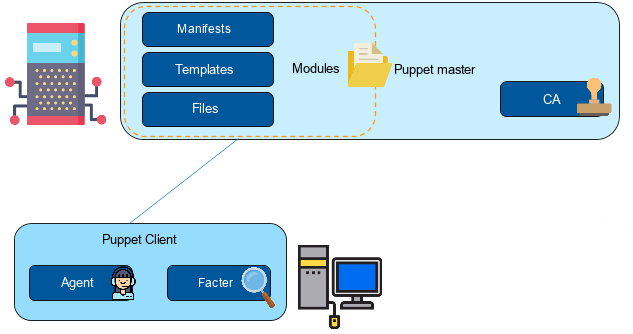
**5.6 - Describe the capabilities of automation tools such as Ansible, Puppet, Chef, and Cisco NSO:**

Ansible: Config 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 Nodes: Network/Server devices that are being managed by ansible.
* Task: list of units of actions.
* Playbook: YAML file which contains ordered set 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

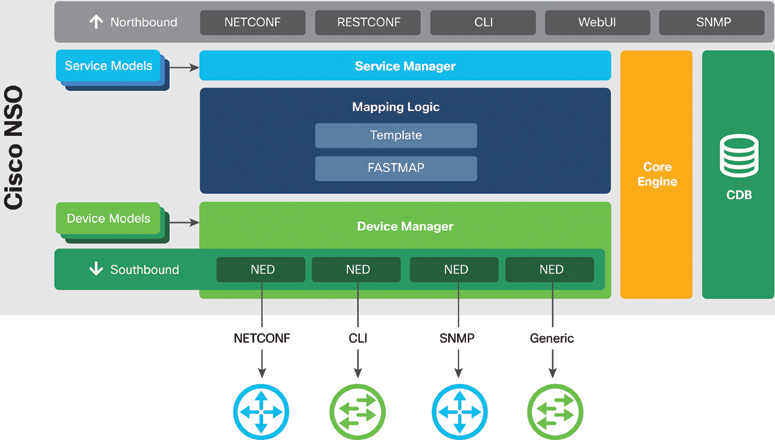
Puppet: Config management and automaton tool based on a client – server architecture utilizing an agent on clients to config/report changes. Additionally, it uses a certificate authority for every client with SSH in a declarative manner where a state is defined, and the process achieves this state. NOTE: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

* 
* Components:
  + CA: is used for internal signing of certificates for clients
  + Modules: is a collection of manifests, templates, and files:
    - Manifests: is the code for the client configuration.
    - Templates: combines code and data to render a final document. This is most generated automatically.
    - Files: These are files included/required for process to complete such as an index.html file for an apache server.
* Workings:
  + 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: config management like puppet based on a client – server architecture utilizing a declarative model and client agents. “Recipes” = Ruby Code, “Cookbooks” = Collection of code, “Knife” = Executes the code, “Onai” = Data Gatherer, “Chef client” = Agent.

* Workings:
  + Ohai gathers and sends gathered facts to the server.
  + The Workstation accesses these facts and allots a cookbook for a desired state.
  + The Workstation creates cookbooks and recipes which is send to the server.
  + 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. Its components include:



* 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 AND 5.11 - Interpret basic YANG models:**

**REFER to 3.8**

**5.12 - Interpret a unified diff**

**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**

**3.0 Cisco Platforms and Development**

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

Basic Practice 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:**

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

**Unified Computing Manager:**

* Cisco Integrated Management Controller (CIMC) = Single Physical Server
* Cisco UCSM = 106 Servers
* Cisco UCS Central = 10,000 Servers
* Cisco UCS Service Profiles: uses this to assign identity to a server
  + Each UCS can only have a single Service Profile at a time.
  + Embedded Management: meaning it is the management server itself.
  + Unified Fabric: self-explanatory
  + Converged Management: manages servers themselves.
* Cisco UCS Manager Servers:
  + Blade Chassis (B-Series), (C-Series). Both are connected to a redundant pair of switches are called UCS Fabric Interconnects (FIs).
* Cisco UCS Management Information Model:
  + All physical and logical components are represented in a hierarchical management information model (MIM) or MIT, which is a tree structure with nodes where each node represents a managed object (MO) which is = to the state.
  + The tree begins with the top called a ‘SYS’ and contains parents and child nodes. And has the same concept of Distinguished names and Relative Names.
  + This model is stored in the “Data Management Engine (DME)”.
* Distinguished Name:
  + <dn="sys/chassis-5/blade-2/adaptor-1"/>
* Relative Name:
  + <dn="sys/chassis-5/blade-2/adaptor-1/host-eth-2"/>
* Authentication Methods:
  + aaaLogin: Initial method for logging in and retrieving an authentication cookie.
  + aaaRefresh: Refreshes the current authentication cookie.
  + aaaLogout: Exits the current session and deactivates the authentication cookie.
  + DEFAULT PORT = 443 & 80.
  + NOTE: Ansible works with UCS Manager & Cisco Integrated Management Controller (CIMC)
  + Programing Frameworks = PowerTools and UCS Python SDK

**Cisco Unified Computing Director:**

* This extends the unification of computing and networking layers through Cisco UCS to provide visibility and management of data center infrastructure components.
  + Create, clone, and deploy services and templates.
  + Monitor organizational usage, trends etc.
  + Deploy and add capacity to converged infrastructure.
  + Manage secure multi-tenant environments.
* Automation and Orchestration:
  + Supports automated workflows with wide array of tasks and use cases.
    - Example:
      * VM provisioning.
      * Network resource configuration and management.
      * Storage Resources config
      * Tenant onboarding
      * Application Infrastructure
* Features and Benefits:
  + Central Management.
  + Self-Service
  + Adaptive Provisioning
  + Dynamic Capacity Management
  + Multiple Hypervisor Support
  + Computing Management
  + Network Management, Storage Management, Dashboards
* Supports Rest API see Code for details
* NOTE: API v1 = JSON or XML, API v2 = XML only.

**Cisco Intersight:**

This is a Software as a Service (SaaS) system management platform capable of managing infrastructure at the edge and remote locations which include data centers. Provides scalability and the REST API allows access to the “Management Information Model” (MIM).

* 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.

Intersight API Keys:

Intersight API keys are categorized into two, an API key ID and a secret key. The API Key ID is a multi-character string always visible after initial key creation. The secret key is an RSA Private Key only available at API key creation. To create API keys in Intersight, you must login at https://intersight.com and perform the following steps:

* Click the Settings icon.
* Click API Keys in the left-hand navigation pane.
* Click Generate API Key.
* Enter a Description for the key.
* Click Generate.
* Click the Save Secret Key to text file icon.
  + A "SecretKey.txt" file is downloaded to your default downloads location.

**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 OR (Unified CM) OR (CUCM):

* Is an IP based comms solution.
* Supports Voice, Video, and data in a single on-prem box with call control and session management.
* Supports call handling such as hold, transfer etc.
* Stores data in an internal database with a Publisher 🡪 Subscriber cluster architecture.

Advance Feature:

* AXL: SOAP API (XML requests) supported by CUCM.
* UDS: Enables end users to update their own personal settings stored on CUCM.
* Etc. if needed look at the website or downloaded files.

Administrative XML Layer:

is an XML/SOAP based interface that provides a mechanism for inserting, retrieving, updating, and removing data from the Unified Communication configuration database.

Graphical user interface, chart

Description automatically generated

This supports every operation that is supported via the web GUI.

* Uses XML and the communication includes passing XML data through and from the server.
* This also supports SQL queries ExecuteSQLupdate or ExecuteSQLquery.

Graphical user interface, text, application, email

Description automatically generated

NOTE: one notable advance feature is Change Notification Feature. You can use this SOAP request repeatedly to see what changes have been made to the system since the last time you ran the request.

User Data Services:

Is a REST-Based API which provides a mechanism for inserting, retrieving, updating, and removing data from the UC config database.

Workings:

* This is a REST based interfaces which sends and receives XML-formatted data. It supports GET, POST, PUT, DELETE.
* It supports Single Sign-on SSO and Basic Authentication for authentication.

Graphical user interface, text, application, email

Description automatically generated

Cisco Finesse:

This is a browser-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.

Finesse Details:

* Contact Center = a centralized location where a company handles service request.
* Inbound tasks = request or communication from the customer.
* Outbound tasks = interaction from the rep to the customer.
* Contact Center Systems uses skill-based routing for call distributor and provides monitoring of agents so that automatic call distributor (ACD) knows who is available to receive tasks. Such as: Not Ready, Ready, Talking, Work, Logout.
* Cisco Finesse API:
  + Supports Both HTTP and HTTPS
  + The API is broadly classified into the:
    - User, Dialog, Queue, Team, CLientLog, Task Routing APIs, Single Sign-On, TeamMessage.
  + Supports Base64-encoded authentication.
  + Sigle Sign-on token.
  + Get, PUT, POST, DELETE
  + Supports a Java Script Library and Python.
  + Use Cases 🡪 DevNet website or downloaded files.

Cisco Webex Teams API:

Authentication:

* Personal Access Token: These are which a single person can use to interact with the REST API. Additionally this token has a lifespan of 12 hours.
* Integrations: These are actions on behalf of other users to obtain an OAuth authorization and supports the OAuth 2 standard. The steps for registrations and the flow request are in the book use if needed.
  + Access Scopes:
    - This defines the level of access that an integration requires. Table 10-2 in the book details API Definitions, memorization of this is not needed most likely.

API Targets:

* There are multiple targets or section for the API calls to hit these include:
  + 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).

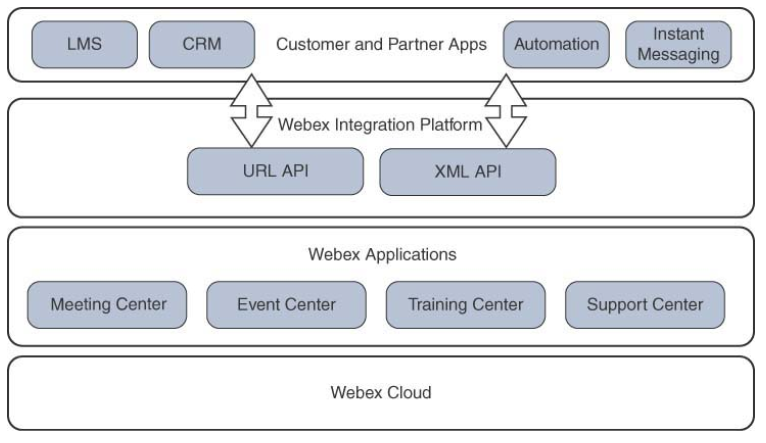
Webex Teams SDK:

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

Cisco Webex Meetings APIs:

This APIs allows for incorporation of meetings into other applications

* URL API:
  + Is a lightweight HTTP/HTTPS-based mechanism which provides browser-based external hooks into the Webex Services. It is used in enterprise portal integrations to support basic interactions such as single sign-on (SSO), scheduling 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:

* There are 3 ways to interact with the Webex Meeting via API:
  + Admin Account
  + User account
  + Webex Meetings Integration

NOTE: Examples of these working is in the code section

Cisco Devices xAPI:

Is an collaboration endpoint for both on-premises registered video conferencing devices (Unified CM and VCS) and devices registered to Cisco’s cloud service (Cisco Webex Devices). xAPI consists of four major groups:

* Commands, Configurations, Status, Events

xAPI Authentication:

* Uses HTTP basic access authentication as a user with the ADMIN role
* Unauthenticated 🡪 401
* 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:

This is an endpoint security solution which support several platforms with a cloud-managed tools delivered via desktops clients, mobile devices, and server-based endpoints. AMP supports:

* Prevention:
  + Identifies malware files
  + Uses a global threat intelligence and can block file-based or non-file-based malware etc.
* Detection:
  + Continuous monitoring and record keeping
  + Ensures visibility into endpoints file activity.
  + AMP Cloud offer 🡪 lookups, signaturing, machine learning etc.
  + TETRA 🡪 antivirus as part of the AMP connector for windows.
  + ClamAV 🡪 similar engine for macOS and Linux.
* Response and Automation:
  + Automated investigation and remediation.
  + Provides 🡪 sandboxing & isolation.
  + Reporting

AMP Endpoints API:

* Authentication:
  + Supports 🡪 Client ID via API Key.
  + Supports 🡪 Basic HTTP authentication Base 64-encoded string.
* API Rate Limits:
  + Three X- headers provide information about rate limiting with the AMP for Endpoints API:
    - X-Rate-Limit-Limit 🡪 Number of total allowed requests in the current period.
    - X-Rate-Limit-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}
* NOTE: EXMPLES IN THE CODE

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 APIs:

* Compatibility:
  + FDM / FTD-API / CDO 🡪 can co-exist.
  + FMC 🡪 for advance cases
* Architecture:
  + FMC can run on 🡪 vSphere or AWS
  + Firepower Management Center (FMC) is multi-device manager for large Enterprise deployments with the need for deep correlation and analytics capabilities.
  + Firepower Device Manager (FDM) is a "single" device manager for small/medium customers. It 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 single IP.
    - Payload Size 🡪 20480 bytes.
  + FTM:
    - API’s upper limit value is 1000.
* NOTE 🡪 EXAMPLE IN THE CODE.

Cisco Identity Service Engine (ISE):

ISE provides a rule-based engine for enabling policy-based network access to users and devices.

* Architecture:
  + Consists of nodes with defined node types these are as follows:
    - Admin Nodes: 🡪 preforms all admin operations
    - Policy Service Node: 🡪 provides network access, posture, guest access, client provisioning, and profiling services.
    - Monitoring Node: 🡪 monitors persona, log collector.
    - 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 registered devices.

ISE REST APIs:

* Consists of two 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
* NOTE: Examples in the code.

Cisco Threat Grid:

This is a malware analysis platform which can be deployed as an appliance or as a service in the cloud. It can also be integrated with other services like AMP, Firepower, Meraki and more.

Threat Grid APIs:

* Format:
  + All calls are made to 🡪 <https://panacea.threatgrid.com/api/>
    - Example 🡪 [https://panacea.threatgrid.com/api/<ver>/<api-endpoint>?q=<query>&api\_key=apikey](https://panacea.threatgrid.com/api/%3cver%3e/%3capi-endpoint%3e?q=%3cquery%3e&api_key=apikey)
* Authentication:
  + Uses API keys which is received via the UI the sets are as:
    - 1. Go to UI page
    - 2. Navigate to manage users.
    - 3. User details page 🡪 copy API key
* Example in CODE

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.