o

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 247464-web-01 | ubuntu | 34.229.69.114 | running | Actions Toggle Dropdown |
| 247464-web-02 | ubuntu | 100.26.122.201 | running | Actions Toggle Dropdown |
| 247464-lb-01 | ubuntu | 54.173.67.36 | running |  |

|  |  |
| --- | --- |
| **0x12-javascript\_warmup** | |
| **process.argv**: is a built-in Node.js variable that stores the command-line arguments passed when running a Node.js script. The first element of process.argv is the path to the Node.js executable, and the second element is the path to the script being executed. The actual arguments start from the third element, so we use .slice(2) to extract only the arguments.   * Take a good look at this code   *#!/usr/bin/node*  const { argv } *=* require('process');  *if* (argv.length *<=* 3) {  console.log(0);  } *else* {  const newArg *=* argv.slice(2).map(row => parseInt(row));  *// console.log(newArg);*  const sortedArg *=* newArg.sort((a, b) => a *-* b);  const secondMax *=* sortedArg[sortedArg.length *-* 2];  *// console.log(sortedArg);*  console.log(secondMax);  }  Sorting:   * To create object instance we can use either of this * ./greeting.js john tom ~ node greeting.js john tom   node is the path to the Node.js executable.  greeting.js is the name of the Node.js script.  John is the first command line argument passed to the script.  tom is the second command line argument passed to the script.  The process.argv array will contain the following elements:  process.argv[0]: /usr/bin/node  process.argv[1]: greeting.js  process.argv[2]: John  process.argv[3]: tom | Sorting:  const items *=* [ { name: "apple", price: 2 }, { name: "banana", price: 1 }, { name: "grape", price: 3 }];  items.sort((a, b) => a.price *-* b.price);  console.log(items);  const words *=* ["apple", "banana", "grape", "cherry"];  words.sort((a, b) => a.length *-* b.length);  console.log(words);  *// Output: ["grape", "apple", "cherry", "banana"]*  const myObject *=* {  type: 'object',     value: 12   };  myObject['incr'] *=* function () {  myObject.value *=* myObject.value *+* 1;   }  ============================================  const myObject *=* {  type: 'object',  value: 12  };  myObject['incr'] *=* function () {  myObject.value *=* myObject.value *+* 1;  }  Or we can directly create it inside the object decleration  Const myObject = { type: 'object', value: 12,  Incr: function() { this.value = this.value + 1; }  }  Example. Download using requsest and write the downloaded content to a file. const fs = require('fs');  const request = require('request');  const url = process.argv[2];  const filePath = process.argv[3];  request(url, function (error, response, body) {  if (error) {  console.log(error);  return;  }  fs.writeFile(filePath, body, function (error) {  if (error) {  console.log(error);  return;  }  console.log('File downloaded successfully!');  });  }); |

|  |  |  |
| --- | --- | --- |
| 0x13-javascript\_objects\_scopes\_closures | | |
| Exporting:   1. Separate class definition and Export:   class Rectangle { }  module.exports = { Rectangel } | 1. Inline Export   module.exports = class Rectangle {} | **module.exports = logme:**  function logme(item) {  console.log(item);  }  module.exports = logme;   * This approach directly assigns the logme function to the entire module.exports object. * It replaces the entire exports object with the logme function. * If you want to export only the logme function and nothing else, this is a straightforward and commonly used method.   **exports.logme = logme:**  javascript  function logme(item) { console.log(item); }  exports.logme = logme;   * This approach adds a new property logme to the exports object and assigns the logme function to it. * It is essentially an alternative syntax­­ to module.exports. * This approach is useful when you want to export multiple values (functions, objects, etc.) from a single module.   Concatinate files:  Const fs = require(‘process’)  // Read the contents of file1  const file1Content = fs.readFileSync(file1Path, 'utf-8');  // Read the contents of file2  const file2Content = fs.readFileSync(file2Path, 'utf-8');  fs.writeFileSync(destination, sourceContent *+* secondSourceContent, 'utf-8'); |
| * print char n times let char = ‘x’ console.log(char.repeate(5)) * num.toString(base)**:** This part of the function calls the toString() method on the input number num and passes the base as an argument. **The** base **argument specifies the numerical base to use for the string representation.** For example, base can be 2 for binary, 8 for octal, 10 for decimal, 16 for hexadecimal, and so on. * Iterate across a dictionary in javascript  1. for (const [key, value] of Object.entries(dictionary)) {   console.log(`${key}: ${value}`);  }   1. Object.entries(dictionary).forEach(([key, value]) => {   console.log(`${key}: ${value}`);  **Using Template Literals:**  javascript  console.log(`${key}: ${value}`);  This uses template literals, which is a modern JavaScript feature that allows you to embed expressions within strings using ${...}. It provides a cleaner and more readable way to interpolate variables into strings.   * check if key exists in a dictionary   const myObject = { key1: 'value1', key2: 'value2'};  if (myObject.hasOwnProperty(keyToCheck))  console.log(‘key exists”);  the above can create error by ESlinter so use this instead  if (Object.prototype.hasOwnProperty.call(myObject, keyToCheck)) {  console.log(`${keyToCheck} exists in the object.`); | |

|  |  |
| --- | --- |
| 0x14-javascript-web\_scraping | |
| **. – file operations:**  Const fs = require(‘fs’) -> makes node js’ file operation available for JavaScript .  Methods available in fs.  **Fs.readFile(file\_path, option, callback);**   * ***option*** (option encoding, option flag (r for read , w for write) * ***callback*** (err, data) => { task } ->is the function that will be executed once the read operation is complete. * Nb: err = null -> if error occurred while reading   else the read data will be stored in the variable data  Fs.writeFile(file\_path, data, encoding, callback)   * **Data** -> data to be written to the file * **Callback** -> function to be executed after the write operation has ended * **Enconding** -> the type of encoding to be used   **Request**   1. **Request(url, option, callback);**   Ex: Request(‘[www.api../id](http://www.api../id)’, function(err, response, data) { task } );  Or request(‘www.api../id’, GET, (err, ‘response’, body) => { task } );   * **Option**: get, post, put, * **Callback**: task to be done after the request is made * **Response:**   Here's an example of a simplified representation of the response format:  Javascript:  const response = {  statusCode: 200, // Status code (e.g., 200 for OK)  headers: {  'content-type': 'application/json', // Example content-type  'content-length': '1234', // Example content length  // Other headers...  },  body: '{ "key": "value" }' // Example response body as a JSON string  }; | const file = require('fs');  File.writeFile(file, data, encoding, function(err) { });  File.writeFile(file, data, encoding, (err)=> {task});  File.readFile(path, encoding, function(err, data) { task });  file.readFile(path, encoding, (err, data)=> {task});  **Array and objects in JavaScript**   * Loop through array:   const colors = ['red', 'green', 'blue', 'yellow'];  colors.forEach((color, index) => {  console.log(`Color at index ${index}: ${color}`);  }); |

|  |  |
| --- | --- |
| **0x0F-python-object\_relational\_mapping** | |
| * **JOIN** **:** separator.joint(iterable) * **a = [0,1,2,3,4,5,6,7,8,9]** * print(“, “.join(a for I in a if a%2 == 0)) * output: 2, 4, 6, 8   pool\_pre\_ping=True, SQLAlchemy will periodically send a small test query (a "ping") to each connection in the pool to check if it's still alive and connected. If a connection is found to be disconnected or not working properly, it will be fixed or replaced before you try to use it for your actual queries. This helps ensure that your program doesn't run into unexpected errors due to connection issues.   1. **Shebang Line (#!/usr/bin/python3):** This is a special line that tells the operating system to use the Python 3 interpreter to execute this script. 2. **Comment Block:**    * The comment block provides a description of the script's purpose and usage.    * The script is meant to list all State objects from a database named hbtn\_0e\_6\_usa.    * The expected usage format is to run the script with three command-line arguments: MySQL username, MySQL password, and the name of the database. 3. **Import Statements:**    * import sys: Imports the sys module, which provides access to command-line arguments.    * from sqlalchemy import create\_engine: Imports the create\_engine function from the SQLAlchemy library, which is used to establish a connection to the database.    * from sqlalchemy.orm import sessionmaker: Imports the sessionmaker class from SQLAlchemy's ORM (Object-Relational Mapping) module, which helps manage database sessions.    * from model\_state import State: Imports the State class from the model\_state module. This class should represent the structure of the State table in the database.   **Create relationship:** | 1. **Main Block (if \_\_name\_\_ == "\_\_main\_\_":):**    * This block contains the main functionality of the script, which is executed when the script is run directly (not imported as a module). 2. **Create Database Engine:**    * The create\_engine function is used to create a database engine.    * It takes a connection URL as an argument. This URL is constructed using the MySQL username, password, and database name provided as command-line arguments.    * pool\_pre\_ping=True ensures that the database connections are checked for validity before use. If a connection is stale, it will be refreshed. 3. **Create Session Factory:**    * Session = sessionmaker(bind=engine): Creates a session factory that's bound to the previously created database engine (engine).    * session = Session(): Creates a session using the session factory. A session is a context for querying and manipulating the database. 4. **Query and Print State Objects:**    * session.query(State): This creates a query object that will retrieve data from the State table.    * .order\_by(State.id): Orders the results by the id column.    * The for loop iterates through the query results (State objects).    * The script prints the id and name attributes of each state using the print function.   This script essentially connects to a MySQL database, retrieves State objects from the State table, and prints their information to the console. It's a basic example of how SQLAlchemy can be used to interact with a database using Python.  **SqlAlchemy**  **Create new data:**  Step 1: new\_state *=* State(name*=*"Louisiana")  Step 2: Session.add(new\_state)  Step 3: session.commit() |
|  |  |

|  |  |
| --- | --- |
|  |  |
| /dev/null: is a special file in Unix-like operating systems that serves as a data sink. It's often referred to as the "null device" or "bit bucket." When data is written to /dev/null, it effectively disappears and is discarded without being stored anywhere. This is useful for situations where you want to discard output or data without affecting the system or consuming unnecessary resources.  update()method is indeed a built-in method for dictionaries that allows you to update the contents of one dictionary with the contents of another dictionary or with an iterable of key-value pairs.  dict1 = {"a": 1, "b": 2} dict2 = {"b": 3, "c": 4}  dict1.update(dict2)  print(dict1) # Output: {'a': 1, 'b': 3, 'c': 4}  the above will change the dictionary content. because it merges them in place  so to create a new dictionary without affecting the new one. use DICTIONARY UNPACKING  merged\_dictionary = {\*\*dict1, \*\*dict2)  hasattr() function in Python is used to determine whether an object has a given attribute or not. It takes two arguments: the first argument is the object you want to check, and the second argument is a string representing the name of the attribute you're.  setattr() function in Python is used to set the value of a named attribute on an object. It takes three arguments: the first argument is the object on which you want to set the attribute, the second argument is a string representing the name of the attribute, and the third argument is the value you want to assign to that attribute.  fromisoformat(value)**:** This is a method of the datetime class. When you call datetime.fromisoformat(value), it parses the given ISO 8601 formatted string and returns a new datetime object that represents the date and time specified in the string.  iso\_date\_string = "2023-08-19T14:30:00" (ISO 8610)  parsed\_datetime = datetime.fromisoformat(iso\_date\_string) | *1 -- Create Database*  *CREATE* *DATABASE* IF *NOT* *EXISTS* hbnb\_dev\_db;  *-- Create new user*  *CREATE* *USER* IF *NOT* *EXISTS* 'hbnb\_dev'@'localhost' IDENTIFIED *BY* 'hbnb\_dev\_pwd';  *-- Grant priviledge*  GRANT CREATE, ALTER, DROP, INSERT, UPDATE, DELETE, SELECT, REFERENCES, RELOAD on \*.\* TO 'sammy'@'localhost' WITH GRANT OPTION;  *GRANT* ALL PRIVILEGES *ON* `hbnb\_dev\_db`.*\** *TO* 'hbnb\_dev'@'localhost';  *-- GRANT SELECT ON performance\_schema TO hbnb\_dev;*  *GRANT* *SELECT* *ON* `performance\_schema`.*\** *TO* 'hbnb\_dev'@'localhost';  *--*  FLUSH PRIVILEGES; |

|  |  |
| --- | --- |
| **0x0B ssh** | |
| 1. create new ssh key: ssh-keygen –t rsa –b 4096 -f school –N betty 2. log into server: ssh -i ~/.ssh/school [ubuntu@52.207.208.97](mailto:ubuntu@52.207.208.97) |  |

|  |  |
| --- | --- |
| Web server | |
| Different types of DNS records are used to store various types of information associated with a domain. Here's an explanation of the DNS record types you mentioned:   1. A Record (Address Record): An A record is one of the most common types of DNS records. It maps a domain name to an IPv4 address. if you enter "[www.example.com](http://www.example.com)" in your browser, the A record will provide the IP address needed to connect to the server hosting the website. 2. CNAME Record (Canonical Name Record): A CNAME record is used to create an alias or a nickname for a domain. It points a domain name to another domain name, rather than directly to an IP address. if you have both "blog.example.com" and "[www.example.com](http://www.example.com)," and you want them both to point to the same server, you can set up a CNAME record that makes "blog.example.com" a CNAME for "[www.example.com](http://www.example.com)." 3. TXT Record (Text Record): A TXT record is a type of DNS record that allows you to associate arbitrary text with a domain. These records are often used for various purposes such as verifying domain ownership, providing additional information, or setting up SPF (Sender Policy Framework) records for email authentication. They can contain human-readable text or machine-readable data. For instance, SPF records use TXT records to specify which mail servers are allowed to send email on behalf of a domain. 4. MX Record (Mail Exchange Record): MX records are used specifically for routing email messages. They point to the mail servers responsible for receiving emails addressed to a specific domain. When you send an email, your email client queries the recipient's domain's MX records to determine where to deliver the email. These records specify the mail server's hostnames and their priority levels (in case there are multiple mail servers). The email is then delivered to the highest priority mail server available.   In summary, these DNS record types play crucial roles in connecting domain names to the appropriate resources on the internet. The A record connects domains to IPv4 addresses, CNAME records create aliases, TXT records store text-based information, and MX records route email messages to the correct mail servers.  **sed command explained**  Certainly! The sed command is a stream editor used to perform basic text transformations on an input stream (a file or input from a pipeline). In this code, sed is being used to modify the Nginx configuration file by searching for a specific pattern and replacing it with another.  Here's a breakdown of the sed command:   * sudo: This command is used to run the subsequent command with superuser privileges. Superuser privileges are required to modify system files like the Nginx configuration. * sed: This is the stream editor command itself. * -i: This option tells sed to edit the file in place. It means that the changes made by sed will be applied directly to the file specified, rather than just being printed to the terminal. * 's/listen 80;/listen 80 default\_server;/g': This part of the command is the actual substitution pattern.   + s/: The s indicates that a substitution is being performed.   + listen 80;: This is the pattern to search for in the file. In this case, it's looking for the text "listen 80;". The 80 here represents the port number 80.   + /: This is a delimiter that separates the pattern to search for from the replacement text.   + listen 80 default\_server;: This is the replacement text. It replaces the "listen 80;" pattern with "listen 80 default\_server;". This change designates port 80 as the default listening port for Nginx.   + /: Another delimiter, separating the replacement text from the flags.   + g: This flag stands for "global" and indicates that the substitution should be applied globally throughout the file. Without this flag, only the first occurrence of the pattern on each line would be replaced. * /etc/nginx/sites-available/default: This is the path to the Nginx configuration file that the sed command should operate on. In this case, it's the default Nginx configuration file that comes with the installation.   So, when the sed command is executed with this pattern, it searches for all occurrences of "listen 80;" in the specified file and replaces them with "listen 80 default\_server;", effectively configuring Nginx to listen on port 80 as the default server.  The command sed -i 's/\r$//' 101-starwars\_characters.js will replace all \r characters at the end of each line in the file 101-starwars\_characters.js with nothing. NB: $ specifies \r at the end of a line The sed command is a stream editor, which means that it reads a stream of data and performs operations on it before writing the modified stream of data back to the output. The -i flag tells sed to edit the file in place, rather than writing the modified data to a new file.  The s/ command tells sed to substitute a pattern with a replacement string. In this case, the pattern is \r$, which matches a \r character at the end of a line. The replacement string is //, which is the empty string.  Sudo chown $USER:$USER /var/www/html/index.html   1. **$USER:**    * $USER is a shell variable that represents the username of the currently logged-in user. When you run a command containing $USER, the shell substitutes it with your actual username.    * For example, if your username is "john," then $USER becomes "john" when you run a command. 2. **: (colon):**    * In Unix-like systems, the colon : is used as a delimiter to separate the owner from the group when specifying ownership for a file or directory.    * When used in the context of chown or other ownership-related commands, the colon is used to separate the owner and group specifications. 3. **$USER:$USER (in the context of chown):**    * When you see $USER:$USER in a command like chown, it means that you are setting both the owner and the group of a file or directory to the same user, which is your current username (the one represented by $USER).    * In other words, the file or directory will have your username as both the owner and the group after running this command.   Here's why setting ownership to $USER:$USER can be useful:   * It simplifies file permissions management by ensuring that both the owner and the group are the same, which can be convenient for single-user systems or when you want to grant yourself full control over a file or directory. * It allows you to work with files without needing to specify different user and group names separately. * It's often used when you need to make sure that the currently logged-in user has full access to a file or directory.   In multi-user environments or when following best practices for security | . sudo chown $USER:$USER /var/www/html/index.html  "Change the ownership of the file '/var/www/html/index.html' to the currently logged-in user and their associated group."  By changing the ownership of the file, you're granting yourself (the logged-in user) control over that file, which means you can read from and write to it, as well as modify its permissions. This command is often used when you want to make changes to files in system directories that require administrative privileges.   * Sed : This is the Stream Editor, a command-line utility that is used for parsing and transforming text.   sed OPTIONS 'COMMAND' FILE   * OPTIONS can include flags like -i for in-place editing. * 'COMMAND' is the sed command that specifies the changes you want to make. * FILE is the path to the file you want to modify.   In the provided sed command, the a command is used to append text after a specific pattern match in the file. This is a powerful way to automate changes in configuration files without manual editing. However, be cautious when using such commands to avoid unintended consequences or breaking your configuration. Always have backups of important files before making changes.  sed -i '/server\_name \_/a location /redirect\_me { rewrite ^ https://www.youtube.com/watch?v=QH2-TGUlwu4? permanent; }' /etc/nginx/sites-available/default  /etc/nginx/sites-available/default). Let's break down the code step by step:   1. sed: This is the command-line utility for text stream processing. It is commonly used for text manipulation and substitution. 2. -i: This is a command-line option for sed, which stands for "in-place edit." When used with sed, it tells the utility to edit the file directly and save the changes back to the file. 3. '/server\_name \_/a location /redirect\_me { rewrite ^ https://www.youtube.com/watch?v=QH2-TGUlwu4? permanent; }': This is a sed command that specifies what changes to make in the file. Let's break down the parts of this sed command:    * '/server\_name \_/a: This part of the sed command specifies a pattern to search for within the file. In this case, it's searching for the text "server\_name \_" within the file.    * location /redirect\_me { rewrite ^ https://www.youtube.com/watch?v=QH2-TGUlwu4? permanent; }': This is the text that will be added after the line matching the pattern. It's essentially appending a new location block to the Nginx configuration file. This location block is used for URL redirection, sending requests to a YouTube video with a permanent (301) redirect. 4. /etc/nginx/sites-available/default: This is the path to the Nginx configuration file that you want to edit. In this case, it's the default configuration file typically found on many Linux systems at this location.   So, when you run this sed command, it will look for a line in the Nginx configuration file that contains "server\_name \_" (which typically represents the default server block for Nginx) and then append the specified location block after that line. This addition can be useful for setting up URL redirection rules in an Nginx server block. Make sure to use this command carefully, as it directly modifies your Nginx configuration file.  The /etc/nginx/ directory typically contains several important configuration files and subdirectories related to the Nginx web server. Here are some of the key files and directories you might encounter in the /etc/nginx/ folder:   1. **nginx.conf:** This is the main Nginx configuration file. It defines global settings and includes other configuration files. It's the entry point for Nginx's configuration. 2. **sites-available:** This directory typically contains individual server block configuration files for different websites or applications hosted on your Nginx server. These configuration files are usually not enabled by default. 3. **sites-enabled:** This directory contains symbolic links to the server block configuration files located in sites-available. When a configuration file is symlinked here, it means the associated website or application is enabled and will be served by Nginx. 4. **conf.d:** This directory can hold additional configuration files for Nginx. It's often used for modular configurations or to separate specific settings, making it easier to manage large configurations. 5. **snippets:** This directory can contain configuration snippets that are reused across multiple server blocks or configuration files. Snippets are included in Nginx configurations using the include directive. 6. **mime.types:** This file defines MIME types for various file extensions. It's used to specify how the web server should interpret different types of files. 7. **fastcgi\_params:** This file contains default FastCGI parameters that can be included in Nginx server blocks when configuring FastCGI applications. 8. **proxy\_params:** Similar to fastcgi\_params, this file contains default proxy-related parameters that can be included when configuring proxy pass settings for reverse proxying. 9. **koi-utf:** This file and the related koi-win file are used for character set conversions. 10. **scgi\_params:** If you're using Nginx with SCGI (Simple Common Gateway Interface), this file contains default SCGI parameters. 11. **uwsgi\_params:** If you're using Nginx with uWSGI (a web server gateway interface), this file contains default uWSGI parameters. 12. **nginx.conf.default:** On some systems, you might find a default Nginx configuration file that is provided as a backup or reference.   Remember that the exact files and directories in /etc/nginx/ can vary depending on your specific system and how Nginx was installed. The organization of these files can also be customized to some extent in the Nginx configuration. Typically, you'll spend most of your time working with the sites-available and sites-enabled directories when configuring virtual hosts for websites hosted on your Nginx server.  In the context of your Nginx configuration, when you use the return 301 or rewrite ... permanent directive, you're setting up a permanent redirection, which is generally preferred when you want to ensure that visitors and search engines recognize the change in URL location.  Here's a breakdown of the differences:   1. **Directive**:    * The first command uses the rewrite directive.    * The second command uses the return directive with a 301 status code. 2. **Usage**:    * Both directives are commonly used for URL redirection in Nginx. 3. **Redirection Type**:    * Both commands will result in a "301 Moved permanently" redirection, which informs clients (browsers and search engines) that the requested content has moved to a new location. 4. **Syntax**:    * The syntax for specifying the redirection target is slightly different between the two directives. In the rewrite directive, the target URL is specified after the ^ symbol. In the return directive, the target URL is provided directly.Top of FormBottom of Form |

|  |  |
| --- | --- |
|  | |
| * afp://www.abc.com : afp = Apple Filing Protocol (AFP) is a network protocol developed by Apple Inc that uses port number 548. for sharing files, folders, and other resources between computers in a network, particularly within Apple's ecosystem   **Curl flags**   -X: Used to specify the HTTP request method. -X DELETE: This flag specifies the HTTP request method, in this case, DELETE. The -X option is used to set the HTTP method to be used in the request.   -s: Used for silent or quiet mode, suppressing unnecessary output.   -d(--data): Used to set data in the request body, often for POST or PUT requests.  Ex: curl -H "User-Agent: MyUserAgent" -H "Authorization: Bearer token123" https://example.com   H (--header): Used to set an HTTP header and its value.  curl -H "User-Agent: MyUserAgent" -H "Authorization: Bearer token123" https://example.com  Ex: curl -d "key1=value1&key2=value2" https://example.com   -b: Used to set cookies in the request header.   -c: Used to save cookies to a file after receiving them in an HTTP response.   * -L: This flag stands for "follow location," which tells curl to follow redirects. If the server responds with a redirect (HTTP 3xx status code), curl will automatically make additional requests to the redirected location.   commonly used curl flags you mentioned:   1. **-s (or --silent):** This flag stands for "silent," and it suppresses progress meter and error messages. It makes curl operate in a quiet mode. 2. **-I (or --head):** This flag stands for "head," and it sends an HTTP HEAD request to retrieve only the headers of the response, without the actual content. 3. **-X (or --request):** This flag stands for "request," and it is used to specify the HTTP request method to be used, such as GET, POST, PUT, DELETE, etc. 4. **-o (or --output):** This flag stands for "output," and it is used to specify the file where the downloaded content should be saved. 5. **-H (or --header):** This flag stands for "header," and it is used to add custom HTTP headers to the request. 6. **-d (or --data):** This flag stands for "data," and it is used to send data as the request body of an HTTP POST request. 7. **-c (or --cookie):** This flag stands for "cookie," and it is used to specify cookies to be sent with the request. 8. **-b (or --cookie-jar):** This flag stands for "cookie-jar," and it is used to save cookies to a file after receiving them in a response. 9. **-A (or --user-agent):** This flag stands for "user-agent," and it is used to set the User-Agent header, indicating the client making the request. 10. **-L (or --location):** This flag stands for "location," and it tells curl to follow redirects. 11. **-w (or --write-out):** This flag stands for "write-out," and it is used to specify a custom output format for the data. 12. **-T (or --upload-file):** This flag stands for "upload-file," and it is used to upload a local file as part of an HTTP request. 13. **-u (or --user):** This flag stands for "user," and it is used to provide authentication credentials in the form of username:password. 14. **-k (or --insecure):** This flag stands for "insecure," and it allows curl to perform requests even if SSL/TLS certificate verification fails.   difference between –I and –H ( -I retrives only header, -H used to set Header   * **Origin:** The Origin header is used in the context of Cross-Origin Resource Sharing (CORS) to indicate the origin (scheme, domain, and port) of the requesting client. It's sent by the browser to the server to let the server know where the request is coming from. It helps servers decide whether to allow or deny the request based on security policies. * **Browser-Name:** "Browser-Name" is not a standard HTTP request header. The correct header to indicate the browser is the User-Agent header. * **Referer (or Referrer):** The Referer header indicates the URL of the referring page that led the client to make the current request. It's useful for tracking where the request originated from. * **Host:** The Host header specifies the host/domain of the server that the client wants to communicate with. It's essential for virtual hosting, where a single server hosts multiple domains. * **Accept:** The Accept header specifies the media types that the client can understand. It helps the server determine the appropriate format for the response. For example, Accept: application/json indicates the client can handle JSON data. * **Authorization:** The Authorization header is used to include credentials or tokens for authentication purposes. It's commonly used in various authentication mechanisms like Basic Auth, Bearer Token, and more. * **Location** When an HTTP response indicates a redirection, the Location header is used to define the URL the client should be redirected to. This header provides the new URL where the client should reissue the request to follow the redirection. It helps the client's browser or application know where to navigate next. * **Content-Type:** The Content-Type header specifies the media type of the data being sent in the request body. It tells the server how to interpret the data. For example, Content-Type: application/json indicates JSON data in the request body. * **Content-Length:** The Content-Length header indicates the size of the data in the request body, in bytes. It helps the server know when it has received the complete request data. * **Cache-Control:** The Cache-Control header provides directives to control caching behavior, specifying how responses should be cached by the client and intermediate caches. * **Accept-Language:** The Accept-Language header indicates the preferred language(s) of the client. It helps the server determine the appropriate language for the response content.   - Content-Type HTTP response header is used to define the formatting of the body of the response. This header provides details to the client about how to interpret the data received in the response. It specifies the media type or MIME type of the data being sent, which helps the client understand how to process the content. For example, a common Content-Type value for JSON data is application/json, indicating that the response body contains JSON-formatted data.  **- MIME** (Multipurpose Internet Mail Extensions) type is a standardized way to indicate the nature and format of a file or data being transferred over the internet.  A MIME type consists of two parts:   * **Media Type:** This part indicates the general category of the data. Examples include text, image, audio, video, application, and more. * **Subtype:** The subtype further refines the type into a more specific format or purpose. For example, the subtype plain in the text media type indicates plain text, while html indicates HTML content. * text/plain indicates plain text. * image/jpeg indicates JPEG image data. * application/json indicates JSON-formatted data. * audio/mp3 indicates MP3 audio data. * "verb" refers to the HTTP verb or HTTP method that is used when making an HTTP request to a web server. | HTTP status codes are three-digit numbers that indicate the outcome of an HTTP request made by a client to a web server. Here's a list of some common HTTP status codes and their general meanings:   1. **1xx Informational:**    * 100 Continue    * 101 Switching Protocols 2. **2xx Success:**    * 200 OK    * 201 Created    * 202 Accepted    * 204 No Content 3. **3xx Redirection:**    * 300 Multiple Choices    * 301 Moved Permanently    * 302 Found (also used for temporary redirects)    * 304 Not Modified 4. **4xx Client Errors:**    * 400 Bad Request malformed syntax    * 401 Unauthorized    * 403 Forbidden    * 404 Not Found    * 405 Method Not Allowed    * 409 Conflict    * 429 Too Many Requests 5. **5xx Server Errors:**    * 500 Internal Server Error    * 501 Not Implemented    * 502 Bad Gateway    * 503 Service Unavailable    * 504 Gateway Timeout |

|  |  |
| --- | --- |
| **0x11 Python Classes** | |
| The urllib.request.urlopen(request) function is used to open a URL (Uniform Resource Locator) specified by the request object and returns a response object that allows you to interact with the content of that URL. Here's what this function does:   1. **Opens the URL:** It sends an HTTP GET request to the URL specified in the request object. The URL can point to a web page, an API endpoint, or any other web resource. 2. **Returns a Response Object:** The function returns a response object that contains information about the HTTP response received from the server, including the response headers, status code, and the response's content. 3. **Connection Management:** urllib.request.urlopen() handles the underlying network connection to the server, including tasks such as establishing the connection, sending the request, and receiving the response. 4. **Content Retrieval:** You can use methods of the response object to retrieve and work with the content of the response. For example, you can use response.read() to read the content as bytes or response.read().decode('utf-8') to decode the content as a string assuming UTF-8 encoding.  * response.read() to read the content as bytes * response.read().decode('utf-8') to decode the content as a string assuming UTF-8 encoding. | NB: response contains 3 things   1. Header 2. status\_code 3. body   **import requests:**  response.json() to attempt parsing the response as JSON, which is a safer and more reliable way to handle JSON data.  response.text to get the body of the response  response.headers to get the header of the response  response.status\_code: to get the status\_code of the response  **import urllib**  response.body to get the body of the response  response.headers to get the header of the response  response.status\_code: to get the status\_code of the response  **Practical example:**  https://jsonplaceholder.typicode.com/todos  **(** this is a mock data for experiment )  The data is in this format. [ “id”: {---}, “1”: {--}, “2”: {--}]   1. To get data with id 1   Response = request.get(url, param={“id”: 1}).json() |

|  |  |
| --- | --- |
| 0x0F. Load balancer | |
| In an HAProxy configuration, there are various settings and options you can configure to tailor the load balancer's behavior to your specific needs. Here are some common configuration elements and settings:   1. **Frontend Configuration:**    * bind: Defines the IP address and port to which HAProxy listens for incoming requests.    * mode: Specifies the protocol mode (e.g., http, tcp, ssl) for the frontend.    * default\_backend: Specifies the default backend to which requests are directed if they don't match any specific rules. 2. **Backend Configuration:**    * balance: Defines the load-balancing algorithm (e.g., roundrobin, leastconn, source) for distributing requests among backend servers.    * server: Specifies the backend servers and their configurations, including IP address, port, and health checks. 3. **ACLs (Access Control Lists):**    * ACLs are used to define conditional rules that determine which backend server to route requests to based on criteria such as URL paths, request headers, or source IP addresses. 4. **Timeouts:**    * timeout client: Specifies the maximum time a client can be idle.    * timeout server: Sets the maximum time for a server to respond to a request.    * timeout connect: Defines the maximum time to establish a connection to a backend server. 5. **Logging:**    * Configurations to control logging, including where logs are stored and their format. 6. **Global Configuration:**    * Settings that apply globally to HAProxy, such as the maximum number of connections or the maximum allowed session rate. 7. **Error Handling:**    * Custom error pages and handling for HTTP error responses. 8. **Security:**    * SSL/TLS termination settings if you're handling encrypted traffic.    * Configuration to protect against common security threats, like DDoS attacks. 9. **Monitoring and Metrics:**    * Options for gathering performance data and metrics for monitoring HAProxy. 10. **Session Persistence:**     * Configuration to maintain session persistence, useful for stateful applications. 11. **Advanced Features:**     * Features like content switching, request rewriting, and more, depending on your use case.   Each HAProxy configuration will vary based on your specific requirements. It's important to thoroughly understand your application's needs and the capabilities of HAProxy to create an effective and efficient configuration. Additionally, regularly monitoring and tuning your HAProxy configuration is essential to ensure optimal performance and reliability. | <<EOF: This is known as the "here-document marker" or "here-doc delimiter." It specifies the start of the block of text you want to include. EOF is a commonly used marker, but you can use any word as long as it's consistent (i.e., you use the same word at the beginning and end of the block). For example, you can use <<CONFIG instead of <<EOF  In Fabric, you can use the following functions to interact with remote servers and execute commands:   1. put: Upload files or directories from the local machine to a remote server.   python   from fabric import put  put(local\_path, remote\_path)   * local\_path: The path to the local file or directory you want to upload. * remote\_path: The path on the remote server where you want to place the file or directory.    get: Download files or directories from a remote server to the local machine.  python   from fabric import get  get(remote\_path, local\_path)   * remote\_path: The path on the remote server to the file or directory you want to download. * local\_path: The path on the local machine where you want to save the downloaded file or directory.    local: Run a shell command on the local machine.  python   from fabric import local  local(command)   * command: The shell command you want to run on the local machine.    run: Run a shell command on a remote server.  python   1. from fabric import run 2. run(command)    * command: The shell command you want to run on the remote server.   These Fabric functions allow you to automate tasks on remote servers and interact with the local machine as well. |

|  |  |
| --- | --- |
|  | |
| To give ownership of the /data/ folder and everything inside it to the ubuntu user and group (assuming the ubuntu user and group exist), you can use the chown command with the -R option to make the ownership change recursive. Here's the command:  sudo chown -R ubuntu:ubuntu /data/  ln -sf /data/web\_static/releases/test/ /data/web\_static/current  This command uses the following options:   * -s: Creates a symbolic link. * -f: Removes the existing symbolic link (if it exists) and replaces it with the new one.   To create a .tgz (tarball) archive from a folder or files, you can use the tar command in Unix-like operating systems. Here's the basic syntax to create a .tgz archive:  bash  tar -czvf archive\_name.tgz source\_directory\_or\_files   * -c: Create a new archive. * -z: Compress the archive using gzip. * -v: Verbose mode (optional, for displaying the progress). * -f: Specify the archive file name.   To uncompress a .tgz (tarball) archive, you can use the tar command in Unix-like operating systems. Here's the basic syntax to uncompress a .tgz archive:  tar -xzvf archive\_name.tgz   * -x: Extract files from an archive. * -z: Decompress the archive using gzip. * -v: Verbose mode (optional, for displaying the progress). * -f: Specify the archive file name. | **ENABLE default site:**  We can enable the default configuration in two ways   1. **creating a symbolic link ( which is the standard)**   **sudo ln -s /etc/nginx/sites-available/default /etc/nginx/sites-enabled/**  2: cat /etc/nginx/sites-available/default" > /etc/nginx/sites-enabled/default"  The code is essentially copying the content of the Nginx server configuration file from the sites-available directory to the sites-enabled directory. This is a common way to enable or activate a specific server configuration in Nginx. Once the configuration is in the sites-enabled directory, it will be picked up by Nginx when it starts or reloads its configuration.  **To work with multiple configurations we can use this code to enable sites according to our choice**  **sudo ln -s /etc/nginx/sites-available/server\_01.conf /etc/nginx/sites-enabled/**  **sudo ln -s /etc/nginx/sites-available/server\_02.conf /etc/nginx/sites-enabled/**  **note:** In many cases, you might have one or more symbolic links in the sites-enabled directory, each representing a different site or virtual host. This allows you to enable and disable individual sites as needed.   * default symbolic link in sites-enabled pointing to default in sites-available, which acts as the default or catch-all configuration for requests that don't match any specific site. |

|  |  |
| --- | --- |
| **Load balancer slenderer**  Name(247464-lb-01) username (Ubuntu) ip (54.173.67.36)  NB: if only https requests are allowed add “redirect scheme https if !{ ssl\_fc }” to **backend www-http** block  **Command to open server: ssh –i ~/.ssh/school** [**ubuntu@34.229.69.114**](mailto:ubuntu@34.229.69.114)  **( the private key school must be copied to ~/.ssh) passphrase: betty** | |
| ubuntu@247464-lb-01:~$ cat /etc/nginx/sites-available/default  upstream backend {  server 34.229.69.114:80;  server 100.26.122.201:80;  }  server {  listen 80 default\_server;  listen [::]:80 default\_server;  server\_name lb-01.tomalx.tech;  index index.html index.htm;  error\_page 404 /404.html;  add\_header X-Served-By $hostname;  location / {  proxy\_pass http://backend;  }  } | ubuntu@247464-lb-01:~$ cat /etc/haproxy/haproxy.cfg  global  log /dev/log local0  log /dev/log local1 notice  chroot /var/lib/haproxy  stats socket /run/haproxy/admin.sock mode 660 level admin  stats timeout 30s  user haproxy  group haproxy  maxconn 2048  tune.ssl.default-dh-param 2048  defaults  log global  mode http  option dontlognull  retries 3  option redispatch  timeout connect 5000  timeout client 50000  timeout server 50000  option http-server-close  errorfile 400 /etc/haproxy/errors/400.http  errorfile 403 /etc/haproxy/errors/403.http  errorfile 408 /etc/haproxy/errors/408.http  errorfile 500 /etc/haproxy/errors/500.http  errorfile 502 /etc/haproxy/errors/502.http  errorfile 503 /etc/haproxy/errors/503.http  errorfile 504 /etc/haproxy/errors/504.http  frontend www-http  bind 0.0.0.0:80  http-request add-header X-Forwarded-Proto http  mode http  default\_backend www-backend  frontend www-https  bind \*:443 ssl crt /etc/letsencrypt/archive/www.tomalx.tech/www.tomalx.tech.pem  mode http  option forwardfor  option http-server-close  acl letsencrypt-acl path\_beg /.well-known/acme-challenge/  use\_backend letsencrypt-backend if letsencrypt-acl  default\_backend www-backend  backend www-backend  balance roundrobin  server 247464-web-01 34.229.69.114:80 check  server 247464-web-02 100.26.122.201:80 check  backend letsencrypt-backend  server letsencrypt 127.0.0.1:54321 |
| SERVER 1  Name(247464-web-01) username (Ubuntu) ip(34.229.69.114) | SERVER 2  Name(247464-web-02) username(Ubuntu) ip (100.26.122.201) |
| Both servers must be the same | |
| server {  listen 80 default\_server;  listen [::]:80 default\_server;  server\_name \_;  index index.html index.htm;  error\_page 404 /404.html;  add\_header X-Served-By $hostname;  location / {  root /var/www/html/;  try\_files $uri $uri/ =404;  }  location /hbnb\_static/ {  alias /data/web\_static/current/;  try\_files $uri $uri/ =404;  }  if ($request\_filename ~ redirect\_me) {  rewrite ^ https://www.youtube.com/watch?v=QH2-TGUlwu4? permanent;  }  location = /404.html {  root /var/www/error/;  internal;  }  } | server {  listen 80 default\_server;  listen [::]:80 default\_server;  server\_name \_;  index index.html index.htm;  error\_page 404 /404.html;  add\_header X-Served-By $hostname;  location / {  root /var/www/html/;  try\_files $uri $uri/ =404;  }  location /hbnb\_static/ {  alias /data/web\_static/current/;  try\_files $uri $uri/ =404;  }  if ($request\_filename ~ redirect\_me) {  rewrite ^ https://www.youtube.com/watch?v=QH2-TGUlwu4? permanent;  }  location = /404.html {  root /var/www/error/;  internal;  }  } |
| **Fire wall**  ufw( uncomplicated fire wall): | |
|  |  |
| Mysql | |
| Primary mysql database:  Configuration file: /etc/mysql/mysql.conf.d/mysqld.cfg  Check configuration file for syntax: sudo mysql –-check-config  Bloaks added to the default config file:   * 1. Server-id = 1 | Step 1: Uninstall MySQL  Open a terminal window.  Depending on your Linux distribution, use one of the following commands to uninstall MySQL:  For Ubuntu and Debian-based systems:  arduino  sudo apt-get remove --purge mysql-server mysql-client mysql-common  For CentOS and Red Hat-based systems:  Step 2: Remove Configuration Files  After uninstalling MySQL, there may still be configuration files left behind. You can remove them manually:  For Ubuntu and Debian-based systems:  bash  sudo rm -rf /etc/mysql /var/lib/mysql  For CentOS and Red Hat-based systems:  bash  sudo rm -rf /etc/my.cnf /var/lib/mysql  For Fedora:  bash  sudo rm -rf /etc/my.cnf /var/lib/mysql  Step 3: Remove MySQL Data Directory  To completely remove the MySQL data directory, use the following command:  bash  sudo rm -rf /var/lib/mysql  Step 4: Remove MySQL User and Group  MySQL may create a system user and group during installation. You can remove them as follows:  sudo userdel mysql  sudo groupdel mysql  Step 5: Remove MySQL Software Repositories (if applicable)  If you installed MySQL from a third-party repository, you should remove the repository to prevent any future installations. The commands to remove the repository will depend on how it was added. For example, if you used the MySQL APT repository on Ubuntu, you can remove it like this:  bash  sudo rm /etc/apt/sources.list.d/mysql.list  Step 6: Update Package Cache (if needed)  Finally, after making these changes, it's a good practice to update the package cache:  For Debian/Ubuntu:  sql  sudo apt-get update |
| Datadog installation on web server 1  DD\_API\_KEY=3687191ce8c363cc3fa8c4832414043d DD\_SITE="datadoghq.com" bash -c "$(curl -L https://s3.amazonaws.com/dd-agent/scripts/install\_script\_agent7.sh)" |  |

|  |  |
| --- | --- |
| **Firewall** | |
| **ufc**  **UFW stands for Uncomplicated Firewall.**  It is a command-line tool that is used to configure the Netfilter firewall on Ubuntu systems. UFW provides a simple and easy-to-use interface for managing firewall rules.  install ufw: apt-get ufw  ***enable ufw: ufw enable***  ***disable ufw: ufw disable***  ***reload ufw: ufw reload***  ***allow all out going: ufw default allow outgoing***  ***disallow all incoming: ufw default deny incoming***  ***allow on port 22(ssh): ufw allow 22/tcp***  **NOTE: if we are using ssh to login to ourserver we must make sure to allow port 22, otherwise we wont be able to logback into our server allow on port 22(ssh): ufw allow 22/tcp**   * Configure web-01 so that its frewall redirects port 8080/TCP to port 80/TCP.   step 1: enable port forwarding of ipv4, inside /etc/sysctl.conf file  sudo vi /etc/sysctl.conf  uncomment/add: net.ipv4.ip\_forward=1  step 2: restart or reload ufw  sudo ufw reload  step 3: add the following firewall rule to the /etc/ufw/before.rules  \*nat  :PREROUTING ACCEPT [0:0]  -A PREROUTING -p tcp --dport 8080 -j REDIRECT --to-port 80  COMMIT  step 4: add the new configuration adjustment parameter to the sysctl.conf file to the kernel  sudo sysctl –p | Important terms:   * telnet: tool to check if sockets are open with telnet IP PORT. For example, if you want to check if port 22 is open on web-02 * kernel: kernel is the core of an operating system. It is responsible for managing the hardware resources of the computer, such as the CPU, memory, and disk storage. It also provides the basic services that other programs need to run, such as file management, networking, and process management. * **sysctl -p** if you modify the /etc/sysctl.conf file to increase the maximum number of open files, you will need to run the **sysctl -p** command to load the new value of the max\_open\_files kernel parameter. * what is the need for forwarding 8080 to 80   **There are a few reasons why you might need to forward port 8080 to port 80:**  You are using a shared hosting service that does not allow you to bind to port 80. Many shared hosting services do not allow users to bind to port 80, as this port is typically reserved for web servers. If you are using a shared hosting service and you want to host a web server, you can forward port 8080 to port 80 and make your web server accessible to the public.  You are running multiple web servers on the same machine. If you are running multiple web servers on the same machine, you can forward different ports to different web servers. For example, you could forward port 8080 to one web server and port 8081 to another web server. This would allow you to access both web servers from the public internet.  You are running a development web server on port 8080 and you want to make it accessible to other developers. If you are running a development web server on port 8080 and you want to make it accessible to other developers, you can forward port 8080 to port 80. This would allow other developers to access your development web server by visiting http://localhost in their web browser.  It is important to note that port forwarding can be a security risk. If you are not careful, you could expose your host machine to unauthorized access. To mitigate this risk, it is important to only forward ports to trusted services. You should also make sure that the services that you are forwarding ports to are properly configured and secure. |
| **0x12 webstack debuging** | | |
| Which one should you use?  In general, you should use nginx reload whenever possible. This will avoid any disruption to service to users. However, if you need to make changes to the Nginx configuration that require the service to be restarted, or if Nginx is experiencing problems, then you should use nginx restart.  Here are some examples of when you might use each command:  **nginx reload:**  Changing the listen port for a website  Adding a new website to Nginx  Changing the logging configuration for Nginx  Updating the Nginx configuration to reflect changes to the upstream servers  **nginx restart:**  Changing the user that Nginx runs as  Changing the number of worker processes  If Nginx is experiencing problems  If you need to clear the Nginx cache | sudo useradd –m nginx  -m flag in the sudo useradd command is used to create a home directory for the new user. | |

|  |  |
| --- | --- |
| **0x11 mysql** | |
| Mysql configuration file: on web server 1  /etc/mysql/mysql.conf.d/mysqld.cnf  Check mysqld.cnf syntax: sudo mysqld --check-config   * SHOW MASTER STATUS   FILE : mysql-bin.000002 | POSITION: 434 | Replica: on web server 2  Mysql> CHANGE MASTER TO  MASTER\_HOST='34.229.69.114',  MASTER\_USER='replica\_user',  MASTER\_PASSWORD='betty',  MASTER\_LOG\_FILE='mysql-bin.000002',  MASTER\_LOG\_POS=434;  THEN : START SLAVE  THEN: SHOW SLAVE STATUS\G  THEIS TWO MUST BE SHOWING YES:  Slave\_IO\_Running: Yes  Slave\_SQL\_Running: Yes  Start the replication application process |

CHANGE MASTER TO MASTER\_HOST='34.229.69.114', MASTER\_USER='replica\_user', MASTER\_PASSWORD='projectcorrection280hbtn', MASTER\_LOG\_FILE='mysql-in.000002', MASTER\_LOG\_POS=434;

|  |  |
| --- | --- |
|  | |
| * **‘web frameworks’**   as “a software framework that is designed to support the development of web applications including web services, web resources and web API  **Strict slash behavior** is a way of handling URLs that have a trailing slash (/) at the end. For example, the following two URLs would be considered different with strict slash behavior enabled:  /about  /about/  However, with strict slash behavior disabled, these two URLs would be considered the same.  Disabling strict slash behavior can be useful for a few reasons:  It can avoid redirect loops when using web servers that automatically add trailing slashes to URLs.  It can maintain consistency with other web applications that do not use strict slash behavior.  It can make URLs more readable and easier to remember.  Here is a layman's explanation of the difference between strict slash behavior enabled and disabled:  Strict slash behavior enabled:  /about and /about/ are considered different URLs.  If you request /about, Flask will redirect you to /about/.  If you request /about/, Flask will return the content for that URL.  Strict slash behavior disabled:  /about and /about/ are considered the same URL.  If you request either /about or /about/, Flask will return the content for that URL.  Whether or not to disable strict slash behavior is a matter of personal preference and the specific needs of your web application. | app.run() method in Flask is used to start the Flask development server. The host and port parameters specify the IP address and port that the server should listen on.  The host parameter defaults to localhost, which means that the server will only be accessible from the local machine. To make the server accessible to other machines on the network, you can specify the IP address of the local machine, or 0.0.0.0 to listen on all available interfaces.  The route definition **/c/<text>** in Flask is a dynamic route that matches any URL that starts with /c/ followed by any text. The text is matched using the variable <text>.  Route parameters allow you to match different URLs to the same view function. For example, the following route definition would match any URL that starts with /user/ followed by a string of 5-10 alphanumeric characters:  Python  @app.route('/user/<string:username:[a-zA-Z0-9]{5,10}>') |

|  |  |
| --- | --- |
| **AirBnB\_clone\_v3** | |
| * The code app.config['SWAGGER'] = { 'title': 'AirBnB clone Restful API', 'uiversion': 3 } is used to configure Swagger for a Node.js application. Swagger is a framework that allows you to document and generate your API documentation. |  |

|  |  |
| --- | --- |
| 0x15. JavaScript - Web jQuer | |
| Task 5: The .append() method is a jQuery method used to insert content or elements inside the selected elements (NB: insertion is made at the end of the selected element).  With .prepend(), the new <li> element will be added at the beginning of the <ul>, before any existing <li> elements.  $("DIV#add\_item").click(function () {  $("ul.my\_list").append("<li>Item</li>");  });  Taske 6: The .text() method in jQuery is used to get or set the text content of an element. It can be used to retrieve the text content of an element or to set new text content for that element. |  |

|  |  |
| --- | --- |
| **AirBnB\_clone\_v4 web dynamics** | |
| * The code app.config['SWAGGER'] = { 'title': 'AirBnB clone Restful API', 'uiversion': 3 } is used to configure Swagger for a Node.js application. Swagger is a framework that allows you to document and generate your API documentation.   Object.values(amenities): This part of the code converts the values of an amenities object into an array. For example, if your amenities object looks like this:  const amenities = {  amenity1: "Swimming Pool",  amenity2: "Gym",  amenity3: "Free Wi-Fi"  };  Then Object.values(amenities) would result in ["Swimming Pool", "Gym", "Free Wi-Fi"]  .join(', ') function is a JavaScript method that is used to join the elements of an array into a single string with a specified separator. It is not a jQuery-specific function; it's a standard JavaScript method.. |  |

|  |  |
| --- | --- |
|  |  |
| The URLs you've provided are related to Reddit and are used to access different types of content from the "programming" subreddit. The main difference between them is in the way they present the content:   1. https://www.reddit.com/r/programming/hot.json:    * This URL fetches the "hot" posts from the "programming" subreddit in JSON format.    * "Hot" posts are those that are currently popular and have a significant number of upvotes and comments. They are typically recent and trending. 2. https://www.reddit.com/r/programming/top.json:    * This URL fetches the "top" posts from the "programming" subreddit in JSON format.    * "Top" posts are those that have received the most upvotes over a longer period of time, usually within the past 24 hours, week, month, or year, depending on the selected time range. These posts are often considered the best or most interesting posts within that time frame.   So, the key distinction is that the first URL (/hot.json) provides you with currently trending and popular posts, while the second URL (/top.json) gives you the highest-rated posts within a specific time range.  **Pagination**  Pagination is a technique used in web APIs, including the Reddit API, to manage and present large sets of data in a structured and manageable way. It involves breaking up a large collection of items (such as posts, comments, or records) into smaller "pages" or "chunks" of data that can be retrieved and displayed one at a time. Pagination is essential for efficiently handling large datasets and ensuring that API responses are manageable for both the client making the request and the server providing the data.  Here's how pagination works in the context of the Reddit API:   1. **Limited Data per Page**: When you request a list of items from the Reddit API (e.g., a list of posts from a subreddit), the API does not return the entire dataset in a single response. Instead, it returns a subset of the data, typically a fixed number of items, on each page. 2. **Page Navigation**: To retrieve additional items beyond the initial page, you need to navigate to subsequent pages. This is typically achieved using one or more of the following mechanisms:    * **"After" Token**: Reddit often includes an "after" token in the API response. This token serves as a marker or reference to the next page of data. To fetch the next page, you include this "after" token in your next API request.    * **"Before" Token**: In some cases, the API may also provide a "before" token to navigate to the previous page of data.    * **Page Numbers**: Alternatively, some APIs might use page numbers or offsets to specify which page of data you want to retrieve. 3. **Requesting Pages**: To fetch additional pages of data, you make successive API requests, each time including the necessary parameters to specify the page you want. This could include the "after" token, "before" token, or page number. 4. **Aggregating Data**: As you retrieve data from multiple pages, you aggregate or concatenate the results to build a complete dataset on the client side. This allows you to work with the entire dataset even though it was returned in smaller portions. 5. **Limitations**: Most APIs, including Reddit's, impose rate limits on the number of requests you can make within a certain time frame. It's important to adhere to these rate limits to avoid being temporarily or permanently blocked from accessing the API. 6. **Efficiency**: Pagination helps improve the efficiency of API requests. Smaller responses are quicker to transfer over the internet, and they allow the server to distribute the load more evenly among all users of the API.   In summary, pagination in the Reddit API is a mechanism for breaking down large sets of data into manageable pages that can be requested and processed incrementally. This approach ensures that both the client and the API server can efficiently handle and transmit large amounts of data while providing users with a way to access all the information they need. | {  "data": {  "after": "t3\_xyz123", // "after" token for pagination  "dist": 25, // Number of items in this page  "children": [  {  "data": {  "title": "Sample Post 1"  // Other post-related data...  }  },  {  "data": {  "title": "Sample Post 2"  // Other post-related data...  }  },  // More children (posts)...  ]  }  }   * "after": This field represents the "after" token, which is used for pagination. It indicates the identifier of the last post in the current page, allowing you to fetch the next page of results. * "dist": This field represents the number of items (posts) in the current page. In this example, there are 25 items in the current page. * "children": This is an array of individual posts, and each post is represented as an object with a "data" field containing various attributes of the post. In your code, you are interested in extracting the "title" attribute from each post. |

|  |  |
| --- | --- |
|  | |
| * **Apache benchmark:** abbreviated as "ab," is a command-line tool and performance testing tool that is part of the Apache HTTP server project. ApacheBench is used to measure the performance of web servers by generating a large number of requests to a web server and measuring how quickly it responds.   **Load Testing**: ApacheBench is primarily used for load testing web servers and web applications. It allows you to simulate a large number of clients making requests to a web server simultaneously, helping you understand how well your server can handle traffic spikes and high loads.  # ab –c 100 –n 2000 localhost/  -n : number of requests  -c : concurrency level of 10 | Task 2:  /etc/security/limits.conf : a system configuration file used in Unix-like operating systems to set resource limits for users or groups of users  Cat /etc/security/limits.con  #Each line describes a limit for a user in the form:  Syntax to set resource limit  #<domain> <type> <item> <value>  can be one of the following:  # - core - limits the core file size (KB)  # - data - max data size (KB)  # - fsize - maximum filesize (KB)  # - memlock - max locked-in-memory address space (KB)  # - nofile - max number of open files  # - rss - max resident set size (KB)  # - stack - max stack size (KB)  su -l holberton instead of su - holberton. Both of these commands will start a login shell with the environment of the "holberton" user |

|  |  |
| --- | --- |
|  |  |
| Requirements to work  Install virtual environment   * Apt install python3.8-venv   Create virtual environment   * Python3 –m venv v2 # creates virtual environment named v2   Activate the virtual environment   * Source v2/bin/activate # to deactivate use the **deactivate** command   Install flask inside AirBnB\_clone\_v2()  If you are unable to curl a page: the port may be being used by another process might be using it so. Use lsof to find the process using port 5000 and kill the process   * **Sudo ps aux | grep 5000** * **Sudo lsof -i 5000** # the -i flag stands for "Internet" or "Internet address." It is used to specify that you want to list open network connections, * **Kill <pid>**   lsof: The lsof command stands for "list open files." It is a powerful and versatile utility that can be used to list information about open files, directories, and network connections on a Unix/Linux system.  Gunicorn, which stands for "Green Unicorn," is a widely used Python Web Server Gateway Interface (WSGI) HTTP server. It's designed to serve as a production-ready, high-performance application server for running Python web applications.   * **gunicorn --bind 0.0.0.0:5000 web\_flask.0-hello\_route:app**   This command is a common way to deploy a Flask application with Gunicorn, especially when you want to make the application publicly accessible over the internet.   * Pkill HUP gunicorn   Many long-running processes, such as web servers like Apache or Nginx, application servers like Gunicorn or uWSGI, and other daemons, can reload their configuration without stopping and restarting the entire process. When they receive a HUP signal, they will re-read their configuration files and apply any changes. This allows administrators to make updates to the configuration (e.g., changing server settings) without causing downtime.  In the context of application servers like Gunicorn or uWSGI, a graceful restart involves starting new worker processes with updated code or configurations and then gradually stopping the old worker processes as they finish their current requests. This allows an application to be updated without abruptly terminating connections and causing downtime.   * **init system,** short for "initialization system   It is the first process that is executed when a computer boots, and it plays a crucial role in system management.   * **Systemd** is a system and service manager for Linux operating systems. It is a replacement for the traditional System V init (SysV init) system, which was used for booting and managing services on many Unix-like systems. Systemd is now the standard init system for most modern Linux distributions. is a system and service manager for Linux operating systems. It is a replacement for the traditional System V init (SysV init) system, which was used for booting and managing services on many Unix-like systems. Systemd is now the standard init system for most modern Linux distributions. * **Daemons** are background processes or services that run on a computer or server, typically without direct user interaction. are background processes or services that run on a computer or server, typically without direct user interaction. deamons typically do not have a graphical or command-line interface that users interact with directly. They operate silently in the background. * **Systmectl:** It is used to manage system services, view their status, enable or disable them, start or stop them, and perform various service-related operations. systemctl provides a centralized interface for controlling and inspecting services in a systemd-based system. | **What is reverse proxy:**  When someone from the outside (like a website visitor) wants to see something from the people inside (the web servers), they come to you (the reverse proxy) and tell you what they need. You then go inside, talk to the right person, get what's needed, and bring it back to the visitor.  In other terms “a reverse proxy is like a helpful middle person that takes care of visitors and makes sure they get what they need from the right people inside the building (web servers). It's a way to manage and protect the traffic between the outside world and your website's servers.”  **Proxy pass directive**  proxy\_pass directive in Nginx is typically used when you want to set up Nginx as a reverse proxy server  proxy\_pass to point to your backend servers and handle the SSL termination at the Nginx level.  **Relationship between Nginx and Gunicorn**  So, when a user makes a request to your web application:   1. The request first goes to Nginx. 2. Nginx forwards the request to Gunicorn. 3. Gunicorn processes the request, executes the Python code, and generates the web page or application response. 4. Gunicorn sends the response back to Nginx. 5. Nginx then delivers the response to the user's web browser.   Gunicorn: Gunicorn is an **application server**. Specifically, it is a WSGI (Web Server Gateway Interface) HTTP server designed to serve Python web applications. An application server is responsible for running your web application code, handling  Incoming HTTP requests, and producing responses to those requests.  **TMUX**   * tmux: This is the tmux command itself, which is used to interact with the tmux terminal multiplexer. * new-session: This is a tmux subcommand that instructs tmux to create a new session. * -d: This is an option or flag used with the new-session subcommand. It stands for "detached." When you use the -d option, it tells tmux to create the new session in the background and immediately detach it from the terminal, meaning that the session will continue running, but you won't see its output in the current terminal. You can think of it as creating a session that runs in the background.   If you have multiple tmux sessions, you can specify the session by name or ID as follows:   * If there is only one detached session   **Bash $ tmux attach-session**   * If there are multiple detached sessions   **Bash $ tmux attach-session -t session\_name\_or\_id**  **to find session\_name\_or\_id**  **$ tmux list-sessions** |
| curl web-01.tomalx.tech/airbnb-dynamic/number\_odd\_or\_even/6  curl web-02.tomalx.tech/airbnb-dynamic/number\_odd\_or\_even/6  curl lb-01.tomalx.tech/airbnb-dynamic/number\_odd\_or\_even/6 |  |

|  |  |
| --- | --- |
| **0x12 webstack debuging** | |
| Which one should you use?  In general, you should use nginx reload whenever possible. This will avoid any disruption to service to users. However, if you need to make changes to the Nginx configuration that require the service to be restarted, or if Nginx is experiencing problems, then you should use nginx restart.  Here are some examples of when you might use each command:  **nginx reload:**  Changing the listen port for a website  Adding a new website to Nginx  Changing the logging configuration for Nginx  Updating the Nginx configuration to reflect changes to the upstream servers  **nginx restart:**  Changing the user that Nginx runs as  Changing the number of worker processes  If Nginx is experiencing problems  If you need to clear the Nginx cache | sudo useradd –m nginx  -m flag in the sudo useradd command is used to create a home directory for the new user. |