SRE Field Guide

If you'd like to contribute, please feel free to add to an existing section, or create your own. Just please don't delete other people's contributions (though if you disagree, feel free to write that/your alternate suggestion/etc!) It helps if you sign your name so people can ask your for more :)

Make sure this document stays in the Holberton Community, please!!

Networking, etc:

My best suggestion is to start with learning Google in a Browser front to back! I've noticed a switch to 'what happens when you type curl blah.com' for SRE stuff, so definitely know how to answer it both ways. Assume the site will be using HTTPS so being able to briefly touch things like public key crypto, HSTS is important!

curl --verbose --trace-time --trace-ascii tracedump.txt --include --output response.txt [https://SOMESITE]

The way I tackled this is by writing down all the steps, and then using TCPDump and Wireshark (wireshark.org) to analyze the traffic and strace to watch the processes themselves. Then I wrote down the major points, the major system calls, the files opened, etc, and tried to consolidate it down to important and interesting ones. You can learn a LOT this way and having a couple things that most interviewers don't know will really set you apart. See: /etc/nsswitch.conf -tb

TCP/IP:

TCP SSL Handshake:

- Hello Request
- Client Hello
- Server Hello
- Certificate
- Server Key Exchange
- Certificate Request
- Server Hello Done
- Certificate Verify
- Client Key Exchange

- Finished

* Reference Cisco

Major points of TCP/IP Header
Basic understanding of DHCP helps

UDP vs TCP vs ICMP

How Traceroute works, what protocol it uses, etc. Just need like, a one sentence response but this is a pretty common question. -tb

Diffie Hellman:

You don't need to know the math but it's very nice to be able to talk about it in a way that shows you understand the concept.

Sample question: At what point are the keys symmetric and at which point is it asymmetric? -tb

https://www.youtube.com/watch?v=YEBfamv-_do

https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

SSH:

Some companies use 'how does SSH work' as an interview question. You should know how keys are made, how they're used (Did you realize your keys are used for authentication, and not encryption? How? What is used for encryption?) and what configuration files are used, etc. If you can discuss HTTPS, SSH shouldn't be that hard! -tb

SSH RFC: https://www.ietf.org/rfc/rfc4251.txt

https://www.digitalocean.com/community/tutorials/understanding-the-ssh-encryption-and-connection-process

From Tim:

What happens when you type "ssh root@my_server_name.com" in the terminal?

Bash:
Checks for aliases
Checks for builtin commands

Checks your path for a command called SSH
Finds /usr/bin/ssh

Forks:
- Parent
  - waits
- Child process
  - execve /usr/bin/ssh
  - Checks /etc/nsswitch.conf
    - This tells it where to find the passwd file (on file or over NIS normally), what order to check for hosts in (file, then mdns, then dns by default)
- SSH opens /etc/passwd to find the current user's home
- Then checks config files for any settings related to connecting:
  - /usr/lib/openssl.conf
  - ~/.ssh/config
  - /etc/ssh/ssh_config
- SSH then checks /etc/services to figure out what port to use for SSH by default, since we didn't specify
- With default nsswitch:
- Checks /etc/hosts to see if we have hard-coded this host
- Then go to /etc/resolv.conf to find a nameserver to resolve it
- Find route for DNS server:
  - First look at ip route table. If not specified in ip route, use default route
  - Default route should point to router, 192.168.1.1
  - If 192.168.1.1 not in ARP table:
    - ARP request for 192.168.1.1
      - Router responds with MAC address
  - Now we can address packet to the router:
  - Frame:
    - dest router MAC, src local NIC MAC, type, checksum
  - IP:
    - total length, dest (nameserver), src (private IP), ttl, protocol, header checksum, options, padding
  - UDP:
    - src port (random), dst port (53), length, checksum
  - Data:
    - standard query for address
- Now that we have the IP address, TCP connection 3 way handshake:
  - tcp header:
  - src port, dst port, seq, ack, header length, window size, checksum, urg, options, data

- SYN (client -> server)

SYN, ACK (server <- client)
ACK (client -> server)

On Server:

Server SSHD process is waiting with select, so it accepts connection, forks a new worker process.

Worker:

Opens the host key in /etc/ssh/, either RSA/DSA/OpenSSL

Checks /etc/hosts.allow and /etc/hosts.deny for entries related to this host

Client:

SSH checks /etc/passwd again for the home

Tries ~/.ssh/ keys:

If keys were specified in the ssh config, it first tries those keys.

Each side sends the protocol version:
client -> server
server <- client

Each side sends encryption and compression options:
client -> server
server <- client

Server also sends its public host key that will be used for the transaction.
Client checks ~/.ssh/known_hosts for fingerprint. If it's been to this host before, it will compare to make sure the key hasn't changed. If it has, it will warn you.

Client requests the start of a key exchange:
client -> server
server <- client

Diffie-Hellman key exchange:
Each side agrees upon a large prime for a seed. SEED SHOULD BE RANDOM. Failing to use a random seed is how NSA has broken traffic in the past
Each side adds a random prime to create a private key.
Each side exchanges public keys based on that private key.
Each side combines the public key with their private key to create a new private key that should be the same between the two.
New private key is the symmetric key that both sides will use to encrypt/decrypt.

TLS encrypted traffic begins.
SSH packet: length (only piece not encrypted), padding, type, data, crc

Now that the session is encrypted, authentication. (asymmetric)
If password:
Client sends password, server checks it against the user's password.
If key:
Client sends fingerprint for key/pair it'd like to use.
Server checks authorized_keys, and if matching pair, it generates a random message and encrypts it with said public key.
Client decrypts the message with the private key.
Client combines the decrypted number with the session encryption key and calculates MD5, responds with MD5.
Server calculates MD5 on its own, compares. If match, authenticated.

The server now displays the motd, forks to execve /bin/sh (or whatever the default shell is for the user), and starts routing the input/output through the open socket.

things to note:
pub/priv keys never provide encryption, only authentication. shared key is used for encryption
issues:
random padding (no padding or padding with the same values can result in being able to decrypt by encrypting the same text over and over w/ the pubkey until a match is found)
failing to use a random secret can result in breaks; NSA suggested having cracked a large percentage of VPNs
-tb

OSI Model:

I never actually got a top down quiz like, ‘WHAT IS EACH LAYER OF THE OSI MODEL’ but be prepared to answer it. I did get questions like ‘so what layer of the OSI model would that be’. It’s super important to know the network stack, and where packets go when they arrive at a destination. For example, most software firewalls operate at from layer 4 (the transport layer) up. That means that physical, data link and network layers are unaffected by software firewalls. There are three layers that are still not protected from network threats! -tb

This is not a ‘real’ model. It is a high level abstraction that packet structures generally trend towards. (It’s a guideline, not a rule) The lines are a little bit fuzzy sometimes

#	Name	Description	Protocols	Mnemonic

1	**Physical**	Physical medium packets are transferred through	Switches and wires	Please
2	**Data Link**	How packets get from one host to another	ARP, MAC Addresses	Do
3	**Network**	Where packets go	IP / ICMP	Not
4	**Transport**	Flow control, Connection or speed?	TCP/UDP	Throw
5	**Session**	Authentication for layer 7 data	See application	Stuffed
6	**Presentation**	Syntax layer for layer 7 data Also handles encryption and compression	See application	Peppers
7	**Application**	End User Layer	HTTP / S, DHCP, SSH	Away

NOTE: Often times, the last three layers: Session, Presentation, and Application (layers 5, 6, and 7) are combined into one layer.

When a packet is transferred between computers (ex: your computer -> your router), it gets encoded from layer 7 -> layer 1. When the packet successfully reaches the router, the router decodes it from layer 1 -> layer 3, to figure out where the packet should go next. The packet hops between servers in this manner before it reaches the destination. When the packet is received by the final application, it is decoded from layer 1 -> 7, and a response is encoded from layer 7->1.

-ixlj

Coding stuff!

I personally found the coding interview stuff for SRE interviews to be fairly easy. The hardest part (and where I screwed up one interview) was not being comfortable with linked lists in Python, lol. Most of the interview questions will be string related: if you're gonna study algo, string stuff like anagrams, palindromes, reversing a string, etc, are super important.

Other things I practiced included API calls, using os.walk, log parsing (in Python and BASH) and basic regex. -tb

Bash:

We should probably talk a lot more about Bash here. In practice, writing a Bash (or other shell variant) script occurs more frequently than writing a python/ruby script. While whiteboarding python is easier on the eyes, writing it in Bash might leave more of an impression. (NOTE: knowing how to code is more important than what you code it in. So if you are struggling with coding, focus on what you are comfortable with.) - RH

http://www.tldp.org/LDP/abs/html/ - This page has been useful to me in particular for String Manipulation, but I would guess that there is much more than just that to be gleaned from here.

Python3:

os.walk:

used for directory traversal. Makes a lot of really hard whiteboard problems really, really short and easy. -tb

defaultdict:

Super nice to know, really helps for a lot of parsing stuff. Once again, this really helps for whiteboarding and looks like you know what you're doing. Half the whiteboard interviews I used it in, I got "Oh man, that's one of my favorite containers in Python". -tb

log parsing:

Find sample Apache logs or syslogs, parse into requests per minute. -tb

Basic regex:

RegexOne is my favorite Regex resource for the basics. If you use it during an interview, make sure you tell them if you're not 100% confident (which you shouldn't be) and it seems to help to tell them that you normally test the hell out of your regex before using it. If you don't understand why regex can be a hell of a double edged blade, read more into it. -tb

To check if your python regex expressions are valid, use this service: http://pythex.org/ The python module for regex is "import re"

A great way to practice basic regex is to start using sed, awk, or perl commands, and/or to start scripting with them in bash. Grep is a great way to start, and you can specify that you're going to use a regular expression with the -E flag. Example:`sudo grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" /var/log/auth.log`(Note, you can use egrep for this instead of grep -oE) This example searches through auth.log for any ip addresses and prints them out. The -o flag specifies that you want "**o**nly matching patterns", and the -E flag specifies that you are using an "**E**xtended regular

expression". Grep's extended regular expressions are a bit different from Sed's, which are also different from awk's, bash's or pythons, but when you get the basics down, it's going to make more sense. Dissect the regular expression, and scour the man page for more clarity on how it works.
Sed: `sed 's/^lala$/I am a banana/g' file.txt` Replaces all instances of the line that contains only the letters 'lala' with "I am a banana"

Use cases:

One thing you need to think about is "why am I doing this in Python instead of BASH". I asked this for a couple whiteboard questions where it would likely have been faster using something like grep. Interviewers like to hear this, but be prepared to also give them a BASH answer. -tb

Python Virtual Environments:

Python is awesome. There's a tool you can install with `pip3 install virtualenv` that lets you create isolated Python environments. Basically it creates a folder that contains all the dependencies you need for a python program, without installing it in the global scope.

1. `Pip3 install virtualenv`
2. `Cd repo_name`
3. `Virtualenv new_env`
4. `Source new_env/bin/activate`

From there, you can pip install whatever, and it will stay in the new_env/bin/ directory.
To get out of the virtual environment, type: deactivate. You might want to record the dependencies for later. To do that, `pip3 freeze > requirements.txtt` You can later reuse this list of requirements with `pip3 install -r requirements.txt`
Source: http://python-guide-pt-br.readthedocs.io/en/latest/dev/virtualenvs/

APIs (Application Programming Interface):

While you normally see API's in the context of a remote server that you request a payload from, APIs are more general than that. Basically, it is a set of protocols or rules for how to communicate with a piece of software.

There are a few different types of API standards, but the most common are SOAP and REST. At this point though, RESTful API's are winning out over the other standards.

SOAP API: Simple Object Access Protocol -- XML

REST API: REpresentational State Transfer -- JSON

The difference between SOAP and REST API's:
https://www.soapui.org/testing-dojo/world-of-api-testing/soap-vs--rest-challenges.html

For Practice:

Write a recursive function that makes API calls. I used the Reddit API for practice, but there's a lot of cool free APIs. DigitalOcean has a cool API, and being able to use it in a project is nifty when you can say "Oh yeah this script launches a new droplet for me with one click" -tb

List of free RESTFUL API's you can play with You don't need to authenticate
Starwars API
Github API
Reddit API
JSONPlaceHolder

List of SOAP API's to play with
soap-client

JQ '.':

JQ is awesome. It's a very helpful command line tool that lets you prettify and filter JSON payloads. It's useful for stripping out irrelevant information from an API call.

The simplest way to use jq is like this:

Curl -s https://swapi.co/api/people/1/ | jq '.'

The above is the starwars api. First we will query it to get a json payload, then we pipe it through to jq, which prettifies and colorizes the output. Just a period is the simplest filter you can use.

Let's try another:

curl -s https://swapi.co/api/people/1/ | jq '.homeworld'

As you may notice, there is the name of a key after the dot this time. This creates a filter for all keys that start with the keyword 'homeworld' and prints the values to the screen.

Let's try something with a more complicated json payload:

curl https://api.github.com/users/stedolan/repos | jq '.[]|.git_url'

This filter uses two things we haven't seen before. The '[]' square brackets and a pipe '|'. The square brackets are for iterating through lists. The pipes work the same way that bash pipes do. The above command says "for each item in the json payload, look for the key 'git_url' and return them to me."

This is a very brief intro to JQ. Try it out it's super helpful, and you can do a lot with it. JQ, like awk, can be it's own language. Visit this link for a brainfuck interpreter in JQ

To install the easy way: visit this page to download the binary and set it in a directory that's in your path.
(For example, mine is in ~/bin, and I put this line in my .bashrc: export PATH=$PATH+":/home/$(whoami)/bin")

AWK

Holberton does not encourage using awk,because you can become too reliant on it. You might come to only know how to do awk stuff, when there may be a way to do something with simpler tools, or just sit down and write some python.

The simplest use cases for awk are printing fields. By default, the fields AWK delimits with are tabs and spaces.

Let's use /var/log/auth.log (a file on all linux machines.)

One line looks like this without parsing:

Aug 13 06:44:06 vagrant-ubuntu-trusty-64 CRON[26287]: pam_unix(cron:session): session closed for user root

Here is the line parsed with sudo Cat /var/log/auth.log | awk '{print $5}'
CRON[26287]:
The command is only printing the fifth field and nothing else.

Another part of the AWK syntax is searching for text then printing out select fields.

Say I have 3 lines this time:

Aug 13 09:17:01 vagrant-ubuntu-trusty-64 CRON[27780]: pam_unix(cron:session): session opened for user root by (uid=0)
Aug 13 09:17:01 vagrant-ubuntu-trusty-64 CRON[27780]: pam_unix(cron:session): session closed for user root
Sep 23 07:36:42 vagrant-ubuntu-trusty-64 systemd-logind[956]: Watching system buttons on /dev/input/event0 (Power Button)

And I run this: sudo cat auth.log | awk '/CRON/ {print $3}'
09:17:01
09:17:01

I searched for all of the lines with CRON in them and printed out the third field (time). The word "/CRON/" in between forward slashes can be any regex or word. Awk '/^[0-9].*#/ {print $2}' will print out all lines with a number followed by any number of any characters until a hashtag and print out the second field of that line.

Another cool thing you can do with awk is use printf for more fine grained control of how your line is printed.

```
 sudo cat auth.log | awk '/CRON/ {printf "%s,%s\n", $3,$8}'
09:17:01,opened
09:17:01,closed
```

Or print out the last field

```
 sudo cat auth.log | awk '/CRON/ {print $NF}'
(uid=0)
root
```

Or the third to last word by using the NF variable and arithmetic:

```
 sudo cat auth.log | awk '/CRON/ {print $(NF-2)}'
root
for
```

Great page for awk one liners: http://www.pement.org/awk/awk1line.txt

Monitoring:

What are key aspects of your system you should be monitoring?
Also remember that you can monitor too much. Monitoring software is still software that takes up computer resources, and above all, man power. If someone is getting 50 monitoring alerts a day, that engineer has to stop what they're doing and resolve the issue, open the ticket, or escalate it to the appropriate department. That takes up energy and time, and can lead to frustration if it is something they have no ability to resolve, or if it is a trivial issue.

1. Hardware:
    a. CPU
    b. Memory Usage
    c. Disk I/O
    d. Bandwidth
    e. Number of network connections
2. Service Measurement:
    a. Process uptime
    b. Per-process memory usage

    c. Number of threads per process
    d. QPS - Queries per second
    e. Database transactions per second
    f. 5xx internal service errors
    g. 4xx Not found/authorized
    h. 3xx Redirection
3. External Monitoring solutions (ie. Sumologic, Datadog, Fireye): Have a third party keep an eye on how your service looks to an end user. ALso called “Black-Box Monitoring”
    a. Page load times
    b. How the service is in different geographic locations
4. Development / Administration:
    a. How are admins / devs using the platform?
    b. Who is authenticating where?
    c. What are they doing when they are there?
    d. When is code pushed?
    e. When does deployed code cause an outage?

Monitoring By Priority

A question that might be asked is, “You are hired to setup monitoring on a web application that has no monitoring currently in place? What would be your week 1 goal? First year goal? Etc.” Here is a list in order of priority. NOTE: I’m writing out my opinion on this mostly because I haven’t found good (read here as ‘non-marketing’) articles on monitoring strategy. - RH

- Customer Endpoint Monitoring - Find all of the customer endpoints of a service. This typically would require an external service to be most effective. This is most critical and should be done week 1 because Customers = $$$. If customers are impacted, your sleep should be impacted.
- Service Monitoring - Once you have established that you will be alerted if customers are getting impacted, then you move on to monitoring individual components of the overall service (e.g. web server, database, app server, etc.). Ideally each of these components are load balanced in some way. So priority would go to monitoring the overall load balanced service, and then to the individual nodes contributing to that service.
- System Performance Monitoring - Now that the overall services are covered, it’s time to start monitoring the hardware it is running on. System performance can sometimes be indicators of service issues. It can also help identify points where the infrastructure or codebase needs improvement.
- Release Monitoring - Code releases can introduce new bugs into the system. Devs might say, “Code releases introduce new ways the infrastructure has to be improved.” Regardless, monitoring the release process (assuming there is one) will help to identify problems before the vast majority of customers are impacted.

Monitoring vs Alerting

I want to make a clear delineation between these two things because they are often thrown together, but they are two very different albeit related things.

- Monitoring
    - Definition: Collection of data into a structured, useful format
    - Data can be overwhelming. There are full-time jobs that focus on the collection and organization of data. This is not your job as an SRE. Beyond this there is a cost factor whether managing that data within your own infrastructure or using a third-party. So while in theory collecting 'all of the data' seems valuable, in practice this is simply not true. If you disagree (and there are reasons to do so), your management will often make clear that collecting all of the data is simply not feasible particularly once they start getting the bill. So your job as a SRE is to understand the services, and identify data points that are valuable for those services. Sometimes you don't know what is useful. So you might collect more than needed and then pare away the unnecessary.
- Alerting
    - Definition: Notifications based upon defined triggers from the collected monitoring data.
    - The tendency when setting up alerts might be to create them for any abnormality in the collected data. If data can be overwhelming, alerting can be annoying. Once people become annoyed with alerting, it is a short hop to ignoring alerting. If an actual problem happens and is ignored, customers can be impacted and the company loses money. Your goal here is to create actionable alerting. If you can't do anything about an alert, then why should you be alerted? The theory of actionable alerting is simple, but the implementation of this is hard (read here as 'why you have a job').

DEBUGGING:

Strace, ltrace, and ptrace:

Strace -f apache2 -- follow a process with strace. Use when the process is waiting for a response.
Strace -p <pid> -- attach to a running process
Playing with ptrace: http://www.linuxjournal.com/article/6100
Another cool flag: -e will let you specify a specific call. If you're looking for a certain file that is being opened, you can use -e open to show only open calls.

GDB:

GDB can be very confusing, but did you know it has a built in text based user interface? Run it like this: `gdb -tui <executable name>` Change layouts like this: `layout split`, `layout reg`, etc

Fundamentals: disass main, set a breakpoint (b main), run, next instruction (ni or n), step to the next breakpoint (s), quit (q)
Resources:
https://beej.us/guide/bggdb/
https://ftp.gnu.org/old-gnu/Manuals/gdb/html_chapter/gdb_19.html

Wireshark and tcpdump (packet analysis)

https://www.wireshark.org/

Browser developer tools

One of the most useful items for debugging webapps in your toolkit. All modern browsers come with a set of these, but my personal opinion is Chrome. (I can drag tabs into a different order, but firefox it's nicer to view JSON)
Mac shortcut to open up the inspector in firefox and chrome: CMD+SHIFT+C
Windows and Linux shortcut to open the debugging tools: CTL+SHIFT+C

There are a few parts to the developer's console. The most useful parts are the the network, console, and elements tabs. The network tab shows you each one of the pieces of the webpage. Images, javascript, css, etc. For each item, you can view the headers...

Useful commands:

Find:

`find . -type f -name "*.o" -delete:` This will find all .o files and delete them.

Grep:

`grep -Rn "lala" *.c` : will search all c files for the pattern "lala" in all subdirectories and display the line number and filename next to each result)

`python -m SimpleHTTPServer <port>` : Spin up a simple HTTP server in your local directory. (only works with python2)

`python3 -m http.server <PORT>` -- Spin up a simple HTTP server in your local directory. (works with python3) courtesy of John Coleman

ps aux | grep <program_name>

Pgrep, pkill, file, which, type, top, w, service, netstat -tnlp, users, ps -aux, cat, sed, awk, lsb_release, printf, iostat, ifconfig, iwconfig, stat, lshw, lsof, locate (not available everywhere),

Hexdump -C / od -tx1 -- Display the raw bytes in hex of the file. Hexdump displays them in their true order, od displays them in the order we think they will be in. Don't worry about the difference immediately. The difference might not be so apparent with simple text files, but will be with big/little endian binary files.

pushd / popd (start a directory stack with pushd, and pop into the latest directory on the stack with popd)

Shortcuts in Shells (most unix flavors) in order of the most useful:
CTL+L -- Clear the screen
CTL+C -- kill the current process (though you probably knew that already)
CTL+A -- jump to the beginning of the line
CTL+E -- jump to the end of the line
CTL+K -- delete the rest of the line starting from your cursor
CTL+U -- delete the beginning of the line to your cursor.
CTL+W -- delete the word before your cursor
CTL+Z -- suspend the job and put it into the background. (the command `fg` will resume the last suspended process. `jobs` will list all of your suspended processes, and `fg <number>` will resume that numbered process from `jobs`)
CTL+R -- perform a reverse-I-search for a previously entered command. Just start typing the command, and the latest matching command will auto-complete
CTL+P -- go to the previous command in bash history
CTL+D -- Send the EOF character to the currently running terminal (terminates the bash session)
ALT + F -- Jump the cursor forward a word
ALT + B -- Jump the cursor backwards a word
CTL+F -- move the cursor forward one character
CTL+B -- move the cursor back one character
CTL+T -- switch the character the cursor is on with the character before it
CTL+H -- Delete the character immediately under the cursor

Databases:

High Level Concepts:

ACID (*Atomicity, Consistency, Isolation, Durability*) - Database transactions,https://en.wikipedia.org/wiki/ACID

High level difference between MongoDB/NoSQL and SQL setups https://code.facebook.com/posts/148779861995359/windex-automation-for-database-provisioning/

https://code.facebook.com/posts/180455938822278/under-the-hood-mysql-pool-scanner-mps-/

https://www.youtube.com/watch?v=bxhYNfFeVF4

https://mirror.as35701.net/video.fosdem.org/2016/ua2220/managing-a-complex-dns-environment.webm

Different Types of Databases and what they're used for:

Check out the wikipedia page first: https://en.wikipedia.org/wiki/Database

SQLite

Postgres

Hadoop: http://hadoop.apache.org/

Relational DB:

MySQL

Distributed DB:

HDFS: Hadoop File System

NoSQL:

Redis, CouchBase, MongoDB, Cassandra,

In-Memory DB: (From Wikipedia:) An in-memory database is a database that primarily resides in main memory, but is typically backed-up by non-volatile computer data storage. Main memory databases are faster than disk databases, and so are often used where response time is critical, such as in telecommunications network equipment.

Caches: Cache all the things

(it's super useful)

You can (and probably should) have caching at each layer of the stack, from the end user, all the way to the databases in your data centers.

Edge caching and CDNs (Content Delivery Network)

Keep all of your common requests in a place that is closer to your end user. For example, If your datacenter lives in San Francisco, but you have a user in Thailand, the distance the packet has to hop can take a lot longer than it takes to get to the user in San Francisco. Then multiply that distance by 3 because of TCP handshaking, then multiply that by 8 or more because of SSL certificate exchange. It would be much easier to deliver content to the end user if we had a some servers in Thailand that could serve Thai traffic. This is where edge caching gets it's name: It is a cache that lives outside of your main data center, and handles distant traffic. It also takes some load off of your data centers. They don't have to process as many similar requests.

In memory caches (Redis, Couchbase, Memcache)

In browser caches. You have a cache living inside your browser.

How Does Bittorrent propagate a file throughout a network?

https://en.wikipedia.org/wiki/BitTorrent

Distributed systems:

AAA (*Authentication, Authorization, Accounting*) -
Security Diameter Protocol: https://en.wikipedia.org/wiki/Diameter_protocol

AAAS (*Authentication, Authorization, Accounting, Secure Transport*)
Security Diameter Protocol: See above link.

CAP Theorem

Transactions

The 12 Factor App:

The twelve-factor app is a methodology for building software-as-a-service apps
https://12factor.net/

Codebase, Dependencies, Config, Backing services, {Build, release, run}, Processes, Port binding, Concurrency, Disposability, Dev/prod parity, Logs, Admin processes

File distribution
(How do you distribute a 1 terabyte file to 10,000 servers?)

System Design:
Say you have a service, what needs to go into that service? Reference the HBnB clone project.

LOAD BALANCING:

HAproxy

Proxies:

Transparent, Caching, Static

Additional Resources:

Books:

Google SRE book: https://landing.google.com/sre/book/index.html

Podcasts:

https://thepracticalsysadmin.com/podcasts-for-system-administrators/
https://sysadmincasts.com/episodes/51-mechanics-of-building-a-carpooling-service-introduction
https://thepracticalsysadmin.com/podcasts-for-system-administrators/

MISC:

Creating a filesystem in RAM (a ramdisk):
https://www.cyberciti.biz/faq/howto-create-linux-ram-disk-filesystem/
Python Virtual environments (virtualenv)
http://python-guide-pt-br.readthedocs.io/en/latest/dev/virtualenvs/

From Lisa Leung: TCP Cheatsheet https://www.sans.org/security-resources/tcpip.pdf

Consistent Hashing, CAP Theorem, Load Balancing,
Caching, Data Partitioning, Indexes, Proxies, Queues, Replication, and choosing between SQL vs. NoSQL

NFS: A file system that is distributed across a network. It can act like a file system on your computer, but is not as fast because of network latency.