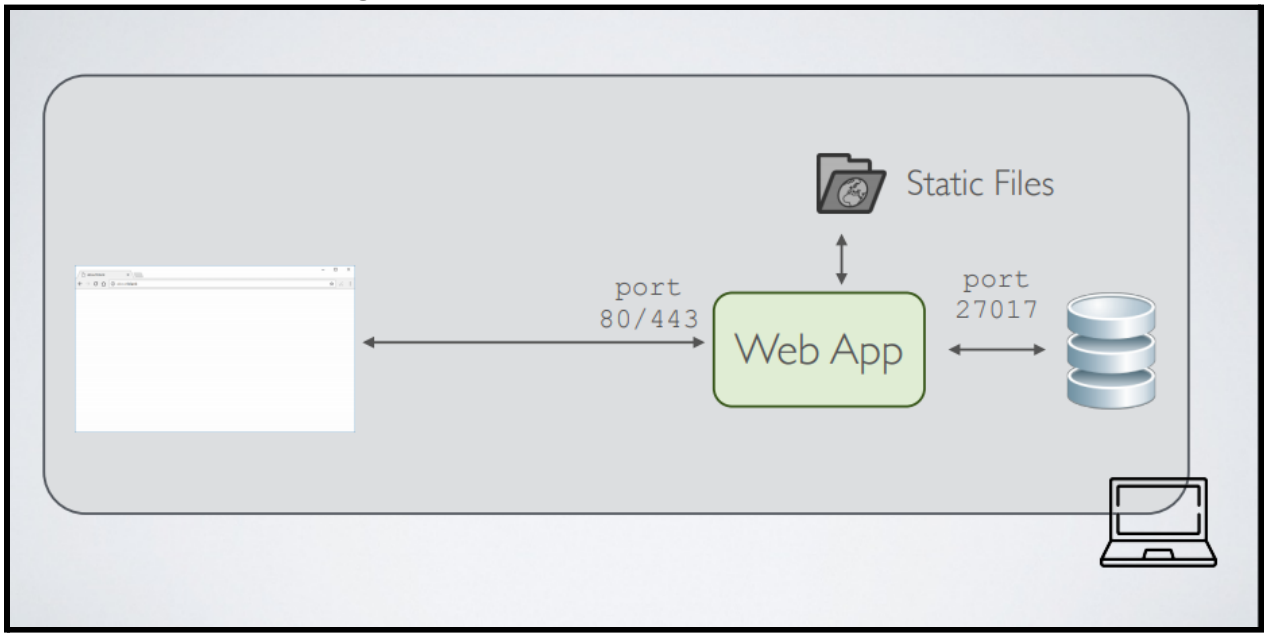
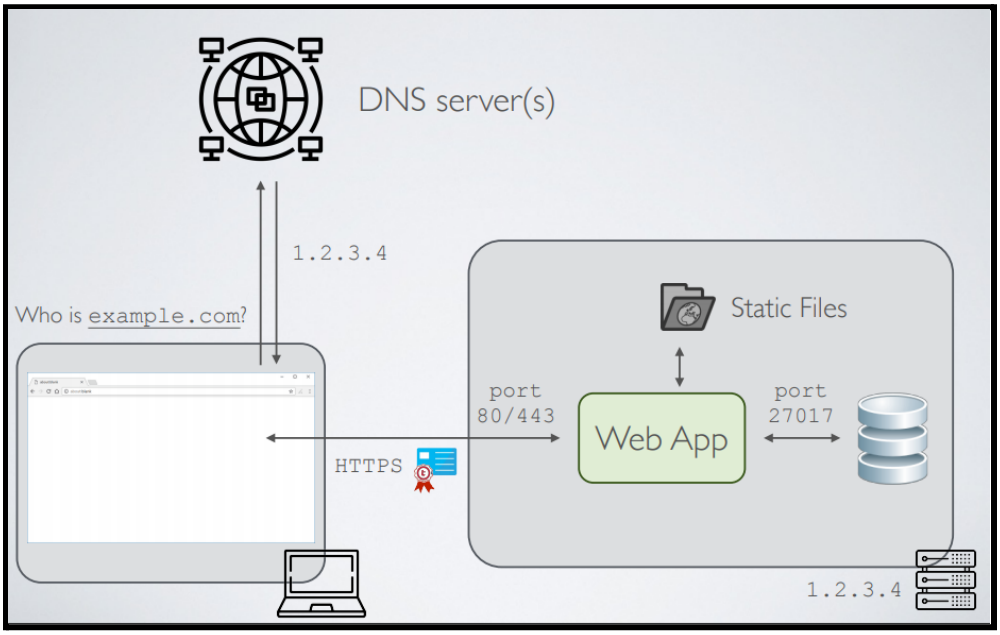
Deploying Web Applications:

- **DNS Server:**
- The **Domain Name System (DNS)** is the phonebook of the Internet. When users type domain names such as ‘google.com’ or ‘nytimes.com’ into web browsers, the DNS is responsible for finding the correct IP address for those sites. Browsers then use those addresses to communicate with origin servers or CDN edge servers to access website information. This all happens thanks to DNS servers: machines dedicated to answering DNS queries.
- **What We Have Been Doing:**



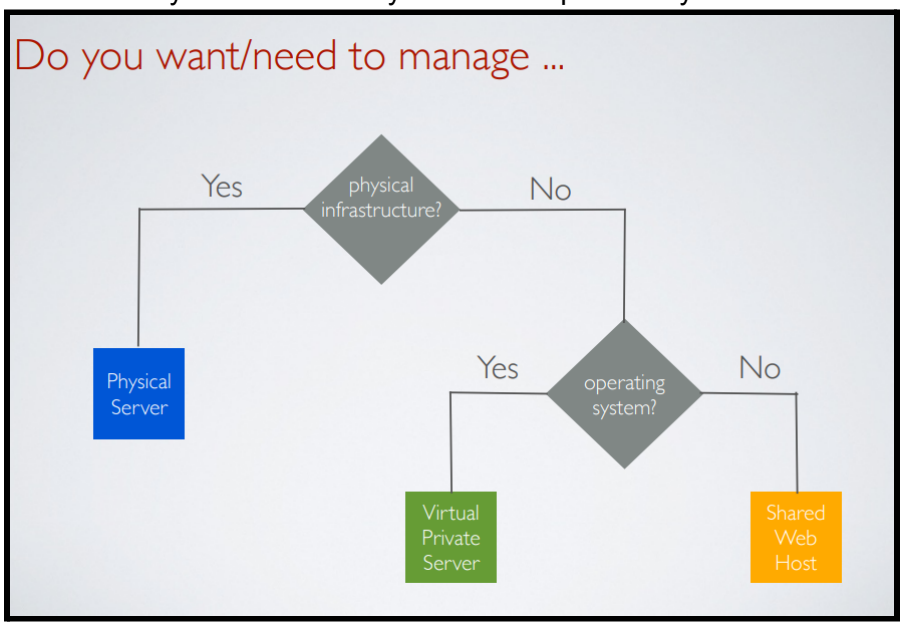
So far, we’ve been running both the frontend and backend on our laptops/PC.

- **What We Want To Do:**



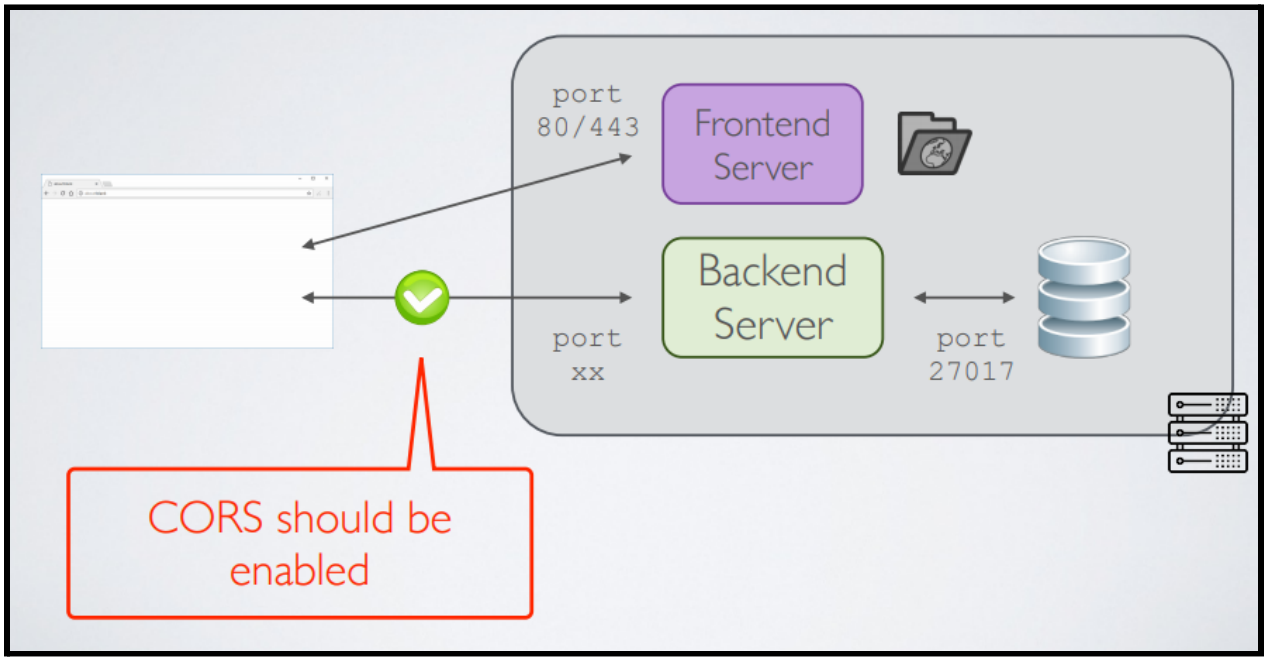
We want to deploy our web application (backend) on a web server that’s remote and has a fixed IP address. We also need a domain name and set up HTTPS on the server.

- **What We Need:**
    1. **Web Host:** A server to host your website.
    2. **Domain Name:** A url for your website.
    3. **Valid Certificate:** A signed certificate for HTTPS.
- **Web Hosting:**
- Most web frameworks provide a development server. These development servers may not be production ready and might not scale with multiple requests (multi-threading).
- **Note:** Node.js is production ready.
- Choosing a web host will depend on many things:
    1. Processing Power: How much CPU and RAM do you need?
    2. Storage: How much space do you need?
    3. Bandwidth: How much traffic do you expect?
    4. Money: How much do you want to spend daily



- Choosing a hosting solution depends on:
    1. Specific needs: Specific applications that your web applications use.
    2. Security: What you are comfortable to administer.
- Dedicated Physical Server:
    - You have total control.
    - However, you have to worry about the maintenance of the physical infrastructure, administration of the operating system.
    - It is not flexible. If you need more power, you need to buy more RAM, CPU, computers, etc.
- Virtual Private Server (VPS):
    - You have to worry about the administration of the operating system.
    - No maintenance of the physical infrastructure. Someone else will take care of this.
    - Flexibility (pay for what you need).
    - E.g. AWS
- Shared Web Host:
    - No administration of the operating system.
    - Very expensive.
    - Not adequate for specific needs.

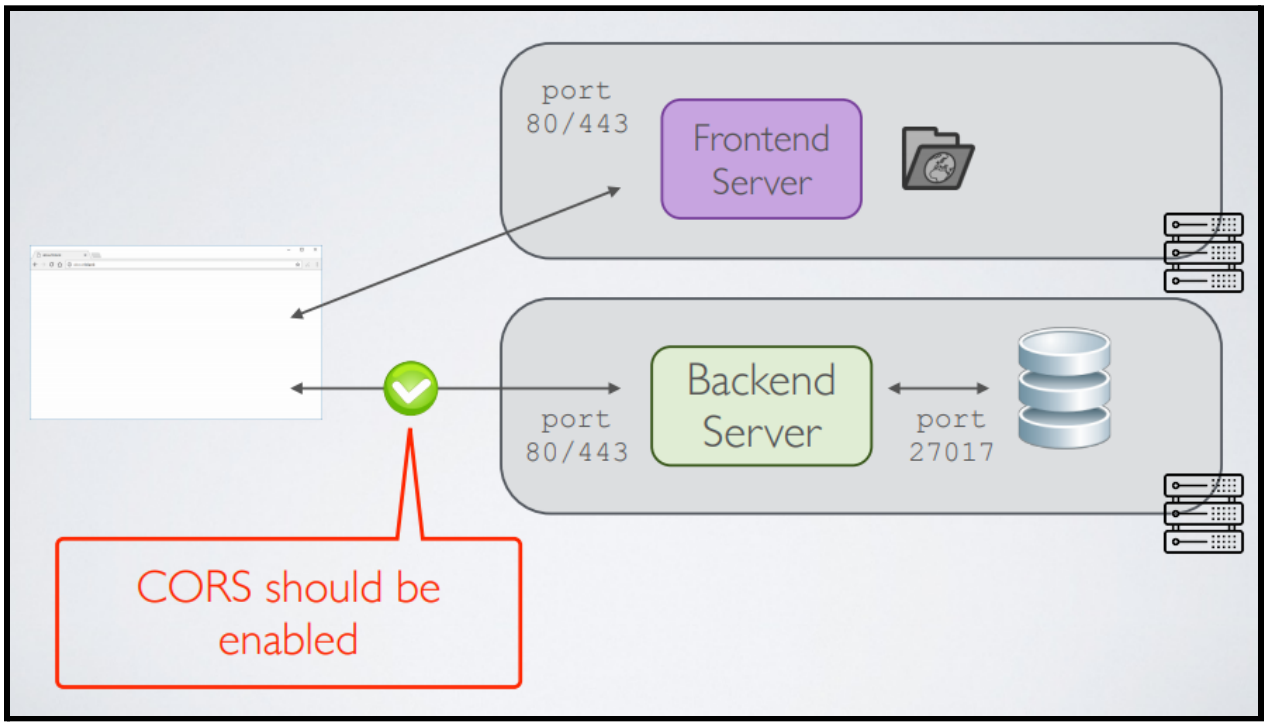
- **Deploying on physical or virtual server:**
- Two servers on the same host



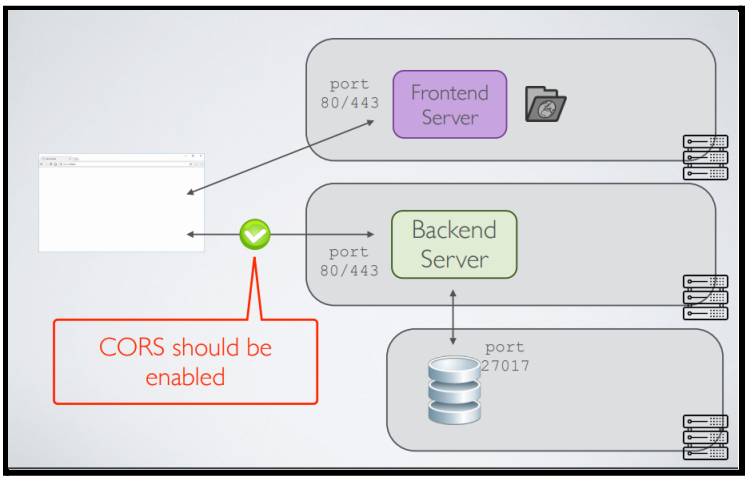
Note: 2 drawbacks of this method are:

1. The database port may be exposed.
2. The packages you installed for the frontend and backend may conflict and cause weird things to happen.

- Two servers on different hosts

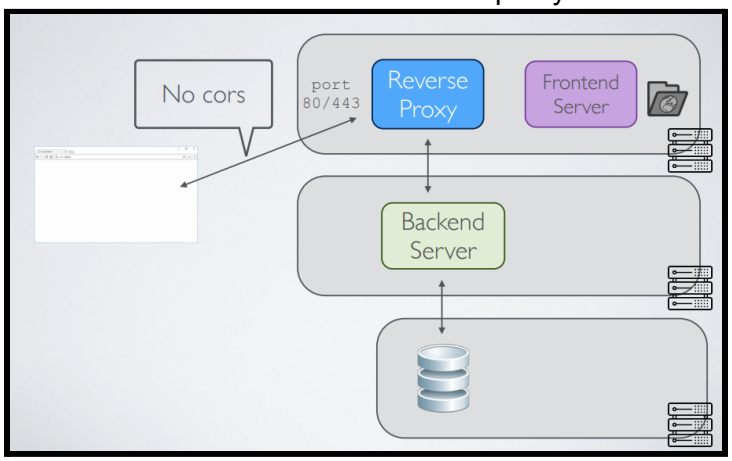


- Three-tiered architecture on different hosts

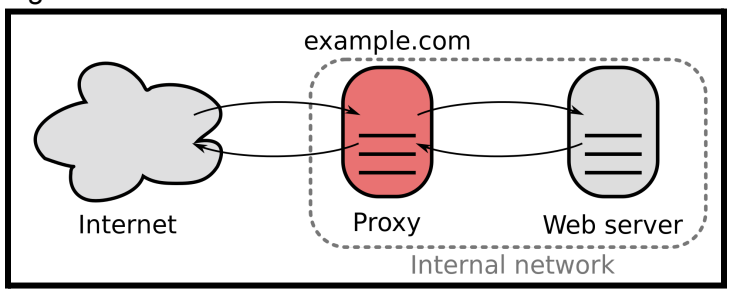


If your frontend crashes, your backend is still running and vice versa.
A drawback is now you need 3 servers.
Another drawback is now enabling CORS can be annoying.

- Three-tiered architecture with reverse proxy

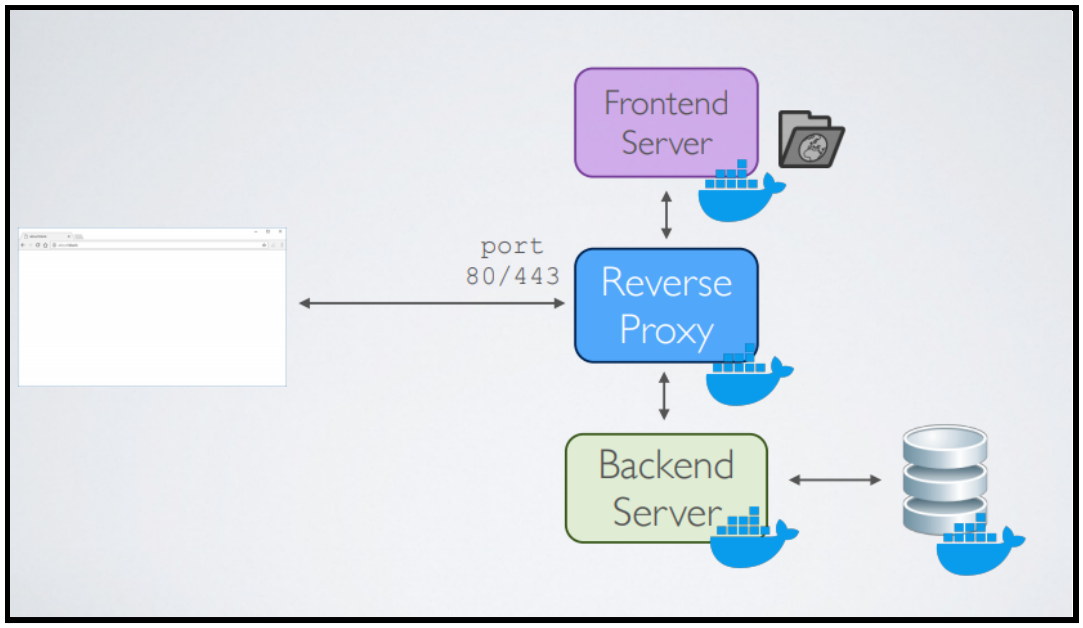


The reverse proxy acts as a gateway/router.
E.g.

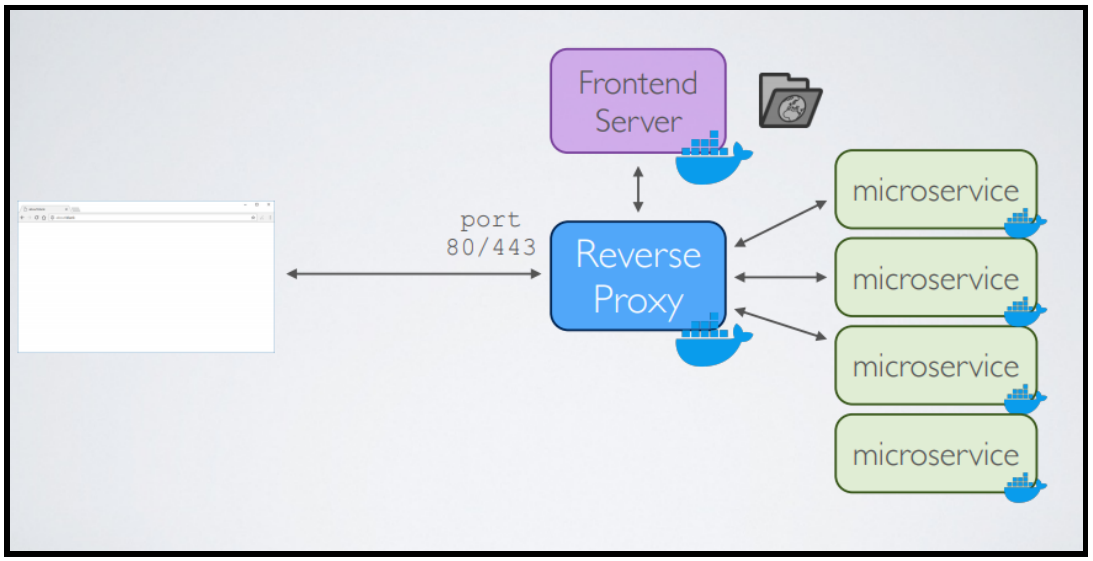


A client makes a request to a reverse proxy server. The proxy inspects the request, determines that it is valid and that it does not have the requested resource in its own cache. It then forwards the request to some internal web server. The internal server delivers the requested resource back to the proxy, which in turn delivers it to the client. The client is unaware of the internal network, and cannot tell whether it is communicating with a proxy or directly with a web server.

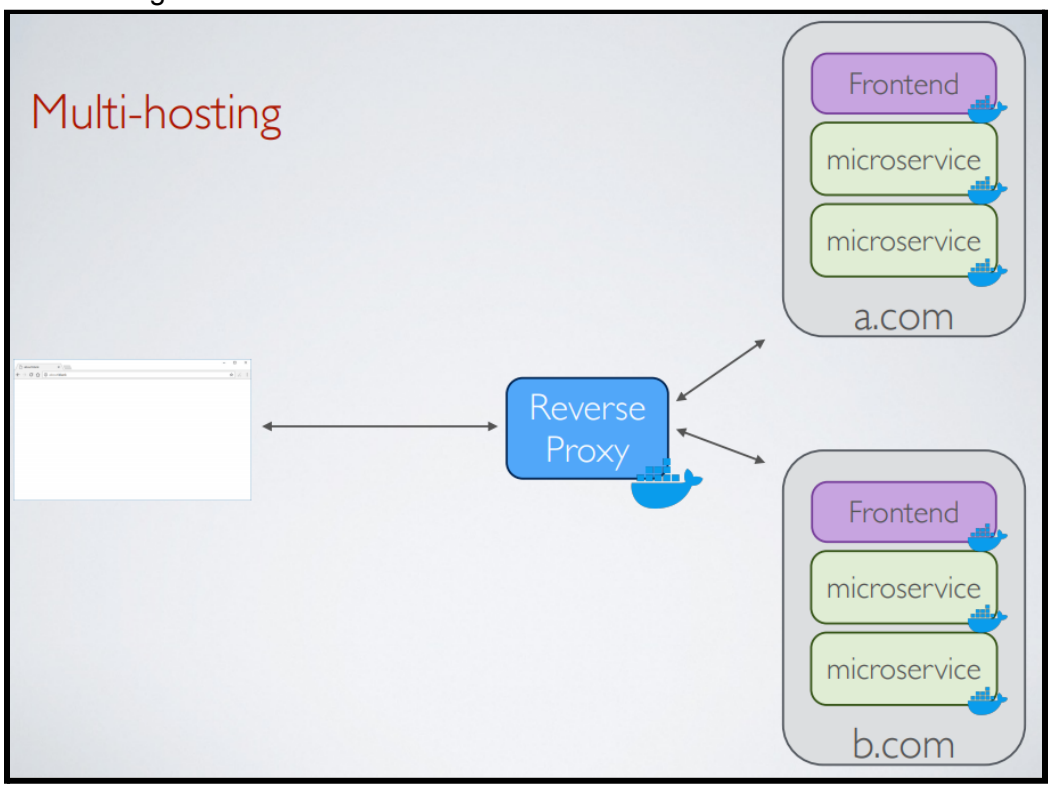
- Why have separated servers:
  - Each piece of our three-tiered architecture relies on specific OS configurations, libraries and runtime environment. These environments might conflict with each other. Having several servers isolates them, making sure they won't conflict with each other. It is easier to maintain and more reliable. Furthermore, if one server (frontend/backend/database) crashes, it won't affect the other parts.
  - However having several servers has a cost. A solution to this is to use virtual servers or containerized servers. It is cost effective and even simpler to maintain and to scale. You can have virtual machines (VMs) inside other VMs.
    One popular containerized server is Docker.
- Dockerized three-tiered architecture



- Dockerized micro-service architecture



- Multi-hosting:



- **Domain Name:**
- Top Level Name:
    - A **top level name** is the last part of the URL. It is the .com or .ca or .edu, etc. Here is a list of top level names: https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains. Note that not all top level names are available for purchase. .edu is reserved for educational facilities like a university and .gov is reserved for governments.
    - Each country also has a top level name. For Canada, it is .ca.
- To get a domain name, you need to buy one from a **domain name registrar**. Domain name registrars are companies that sell domain names.
  Examples of domain name registrar are GoDaddy and Namecheap.
  Note: You have to renew the domain name. Otherwise, other people can buy it.
- **WHOIS** is a query and response protocol that is widely used for querying databases that store the registered users or assignees of an Internet resource, such as a domain name. I.e. WHOIS gets information about a website.
- **A Valid Certificate:**
- The domain name registrar can provide a valid certificate.
- There are also specific companies that provide certificates. You need to be able to prove to them that you own the website.