Fundamentals of Web Development

Third Edition by Randy Connolly and Ricardo Hoar

Chapter 17

DevOps and Hosting

Pearson

In this chapter you will learn . . .

- About DevOps and how to apply those techniques.
- About different web server hosting and cloud hosting options
- About domain and name server configuration
- About monitoring and tuning tools to improve website performance
- About containers, and server and cloud virtualization

DevOps: Development and Operations

Historically, the operation of a server would be done by a team of system administrators completely separate from the developers.

With web development, it first became apparent that isolating hosting from development was not productive, and that knowledge about the operation of the server is essential for developers, whether working alone or in large teams.

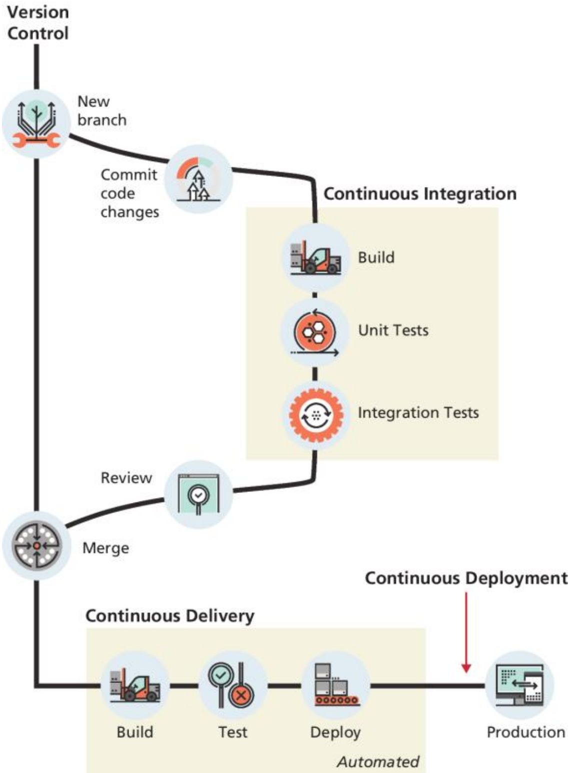
So commonplace is combining the Development and Operations roles and responsibilities, that it has a common name: **DevOps**. DevOps is a philosophy that has inspired many new ideas and strategies, all drawing on the benefit of having blurred lines between development and production.

Continuous Integration

Continuous Integration (CI) aspires to shorten development cycles by making each developer integrate their changes against a central code repository as often as possible.

The term **Continuous Delivery (CD)** refers to this practice of automating the release process.

The logical continuation of Continuous Delivery is **Continuous Deployment**, in which changes in the source code that passes the tests within a continuous integration cycle is automatically deployed to production without the intervention or approval of the developer.



Testing

Testing is especially difficult for web developers because of the complexities involved in the client server model.

There are, generally speaking, two types of testing in regards to web applications:

- **Functional testing** is testing the system's functional requirements and is familiar to programmers because we have to test if our applications work as expected, even if only in an ad-hoc way.
- **Non-functional testing** refers to a broad category of tests that do not cover the functionality of the application, but instead evaluate quality characteristics such as usability, security, and performance.

Functional Testing

A **unit test** is a small program written to test one feature (unit) of the application. Unit tests typically access low level functions/variable directly, so you can test numeric functionality without having to worry about javascript, cookies, and browser rendering.

Integration tests tests whether the smaller units tested via unit testing work together as expected.

If you want to test how your application interfaces with a browser, you will need to explore test automation tools such as Telerik TestStudio, HP Unified Functional testing, TestComplete, or the opensource *Selenium*.

Non-functional testing

Security threats are much more acute with web applications and typically require a completely different testing approach known as **penetration testing**.

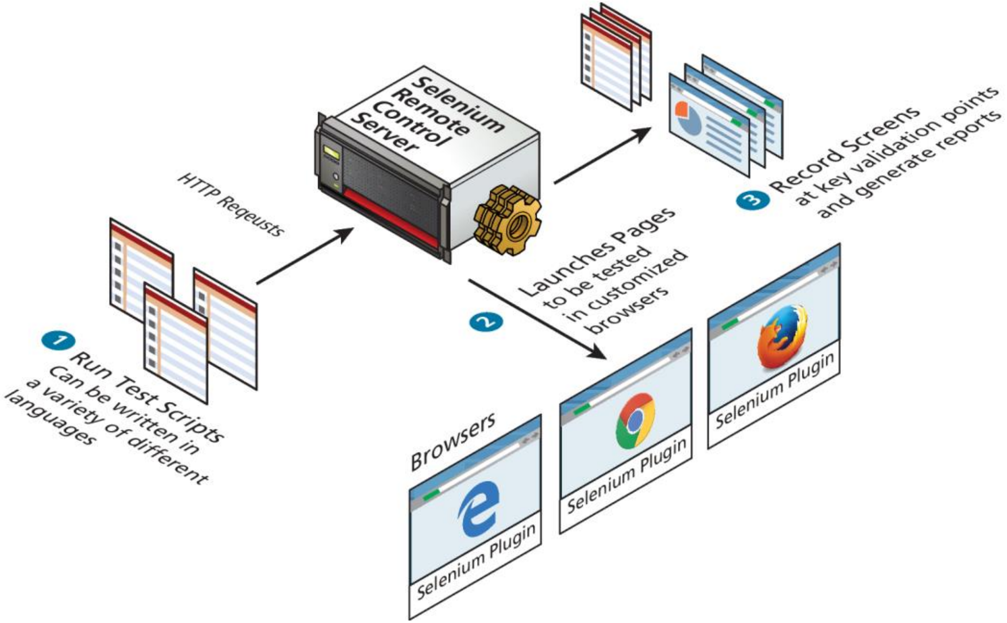
Performance and load testing are non-functional tests in which a web application is given different demand (request) levels to evaluate a system’s performance under normal and peak.

These tests normally make use of testing frameworks like Selenium, so that the web server latency, browser rendering, and timing issues can all be tested across a wide range of platforms.

Selenium testing system

Tools like Selenium allow you to program a browser to mimic clicks and scrolls as if it were being driven by a user.

You can check, for example, if an HTML page contains certain text after a sequence of clicks, or if a layout element appears within a defined area or in a certain colour.


Selenium Remote Control Server
HTTP Requests
1 Run Test Scripts
Can be written in a variety of different languages
2 Launches Pages to be tested in customized browsers
3 Record Screens at key validation points and generate reports
Browsers
Selenium Plugin
Selenium Plugin
Selenium Plugin

Infrastructure as Code

When one considers the database servers, mailhosts, and all the other systems that may be required in a production environment, the challenge of having every developer on identical systems becomes clear.

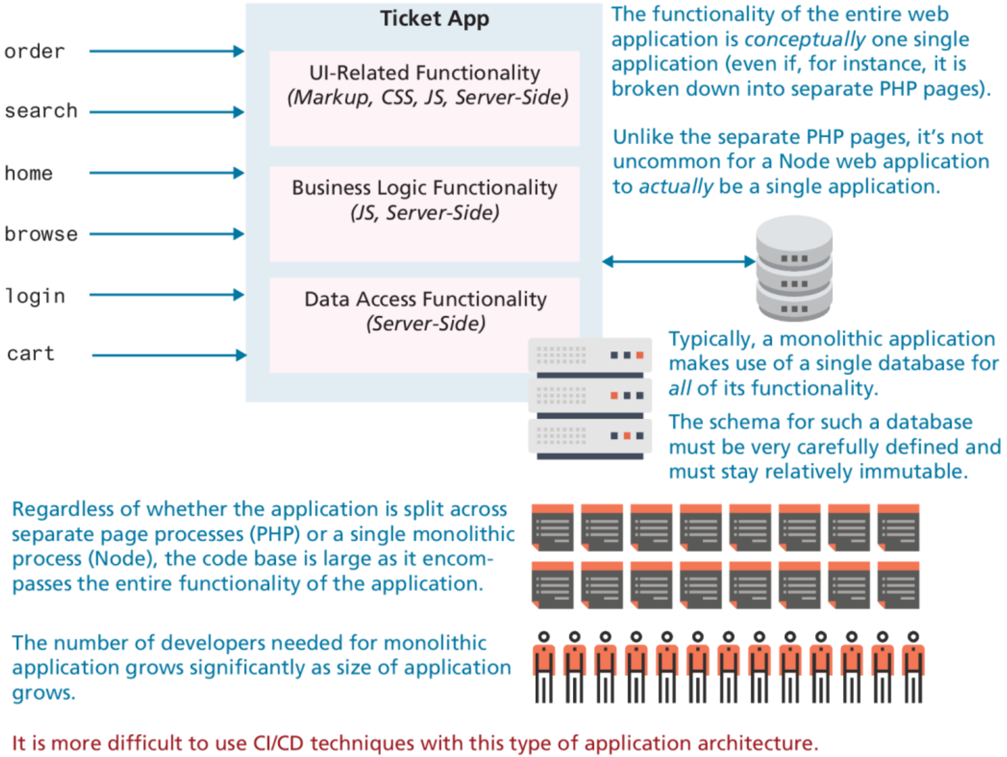
Thankfully, from the operations side of DevOps comes the answer. **Infrastructure as Code (IoC)** describe powerful systems (such as Ansible and Vagrant) that abstract system requirements into text files which can then be checked in and out of versioning systems just like code.

Managing infrastructure as textbased descriptions is a powerful concept that also helps system administrators in setting up redundant distributed systems with advanced deployment features, such as load balancing and DDOS handling

Monolithic architecture

In a **monolithic architecture** the code base—encompassing, HTML, CSS, JavaScript, and whatever server-side language files are being used.

Such an architecture, with dozens if not hundreds of dependencies, is difficult to understand, test, maintain, and expand.



The functionality of the entire web application is *conceptually* one single application (even if, for instance, it is broken down into separate PHP pages).

Unlike the separate PHP pages, it's not uncommon for a Node web application to *actually* be a single application.

Typically, a monolithic application makes use of a single database for *all* of its functionality.

The schema for such a database must be very carefully defined and must stay relatively immutable.

Regardless of whether the application is split across separate page processes (PHP) or a single monolithic process (Node), the code base is large as it encompasses the entire functionality of the application.

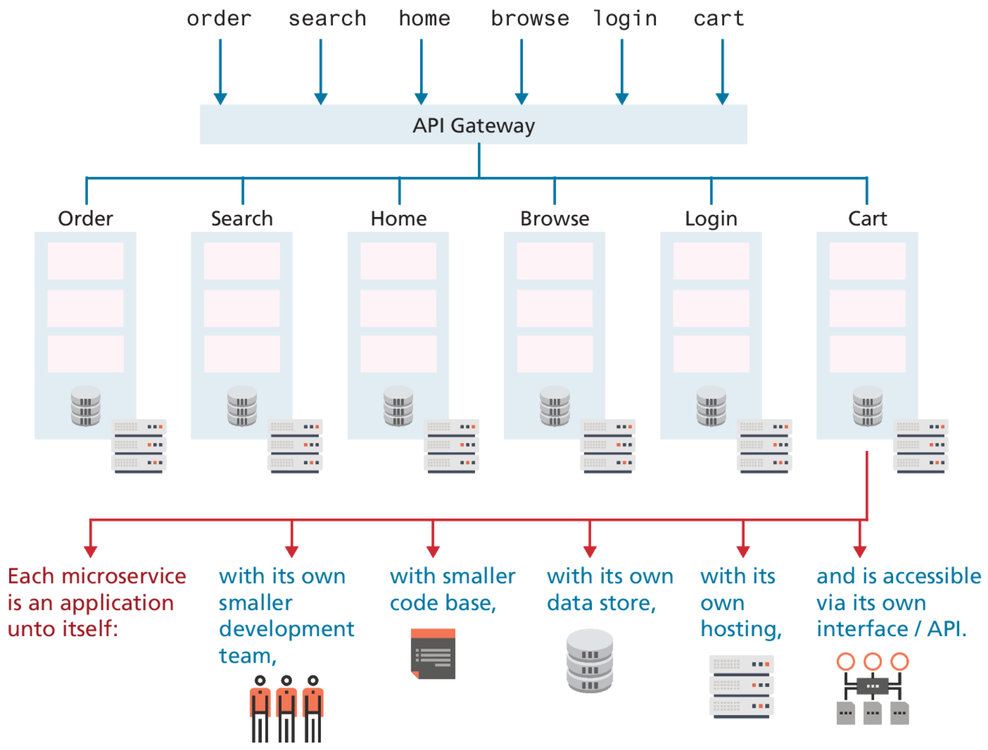
The number of developers needed for monolithic application grows significantly as size of application grows.

It is more difficult to use CI/CD techniques with this type of application architecture.

Microservice architecture

Microservice architecture—disaggregates the single monolith into a system comprising many small, well-defined modules, scripts or programs.

The advantages of a microservice architecture is that it encourages distributed, non-centralized code bases and teams.

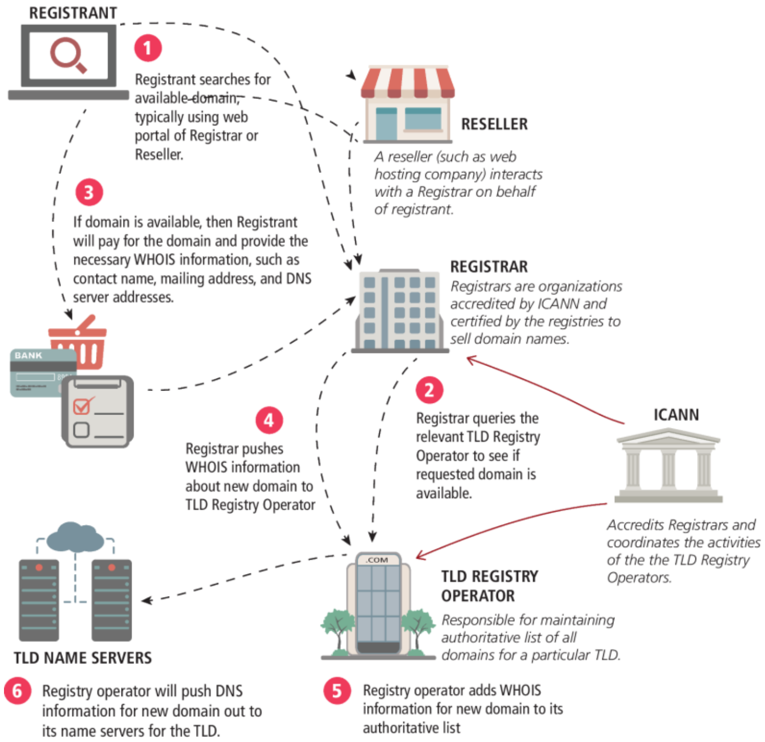


It is easier to use CI/CD techniques with this type of application architecture.

Domain Name Administration

How to take ownership of a domain and then associate it with your preferred method of hosting is all facilitated through the domain name system (DNS) first covered back in Chapter 2.

The details about managing a domain name for your site require that you understand the parties involved in a DNS resolution request as shown in **Figure 2.8**.



1 Registrant searches for available domain, typically using web portal of Registrar or Reseller.

A reseller (such as web hosting company) interacts with a Registrar on behalf of registrant.

3 If domain is available, then Registrant will pay for the domain and provide the necessary WHOIS information, such as contact name, mailing address, and DNS server addresses.

Registrars are organizations accredited by ICANN and certified by the registries to sell domain names.

2 Registrar queries the relevant TLD Registry Operator to see if requested domain is available.

4 Registrar pushes WHOIS information about new domain to TLD Registry Operator

Accredits Registrars and coordinates the activities of the the TLD Registry Operators.

Responsible for maintaining authoritative list of all domains for a particular TLD.

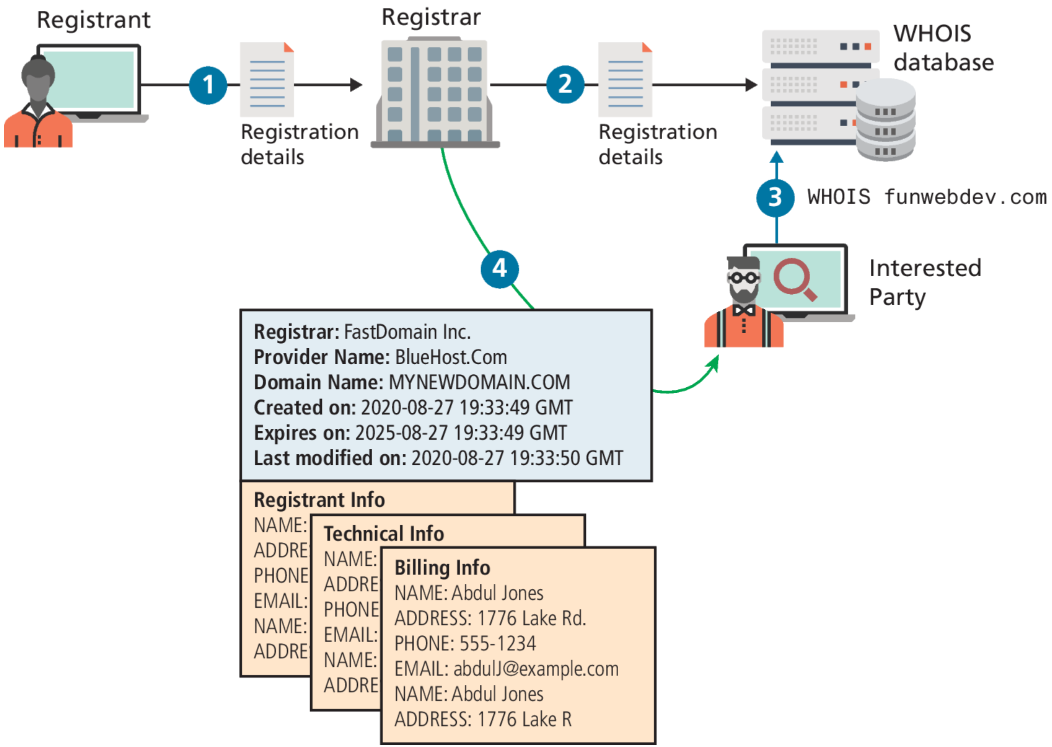
6 Registry operator will push DNS information for new domain out to its name servers for the TLD.

5 Registry operator adds WHOIS information for new domain to its authoritative list

WHOIS

Registrars must collect and maintain your information in a database. Anyone can try and find out who owns a domain by running the WHOIS command and reading the output.

Your registration agreement requires you to provide accurate information to WHOIS information.



Registrar: FastDomain Inc.
Provider Name: BlueHost.Com
Domain Name: MYNEWDOMAIN.COM
Created on: 2020-08-27 19:33:49 GMT
Expires on: 2025-08-27 19:33:49 GMT
Last modified on: 2020-08-27 19:33:50 GMT

Registrant Info
NAME:
ADDRE
PHONE
EMAIL:
NAME:
ADDRE

Technical Info
NAME:
ADDRE
PHONE
EMAIL:
NAME:
ADDRE

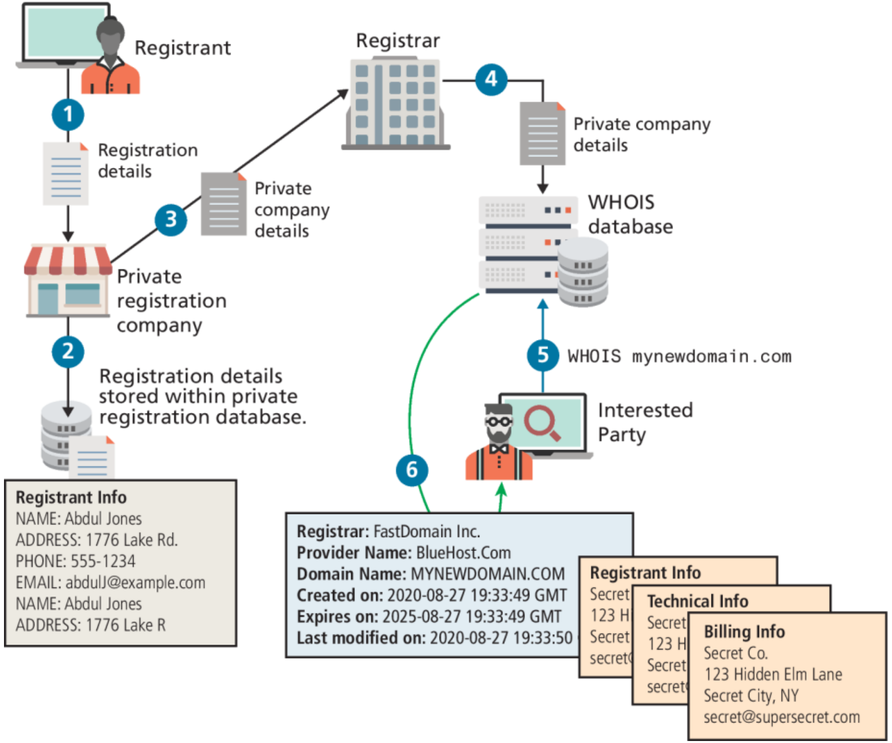
Billing Info
NAME: Abdul Jones
ADDRESS: 1776 Lake Rd.
PHONE: 555-1234
EMAIL: abdulJ@example.com
NAME: Abdul Jones
ADDRESS: 1776 Lake R

Private Registration

Many registrars provide private registration services, which broker a deal with a private company as an intermediary

These third-party companies use their own contact information in the WHOIS system, keeping your contact information hidden from stalkers, spammers, and other threats.

Private registrants will turn your information over to authorities upon request



Registrant Info
NAME: Abdul Jones
ADDRESS: 1776 Lake Rd.
PHONE: 555-1234
EMAIL: abdulJ@example.com
NAME: Abdul Jones
ADDRESS: 1776 Lake R

Registrar: FastDomain Inc.
Provider Name: BlueHost.Com
Domain Name: MYNEWDOMAIN.COM
Created on: 2020-08-27 19:33:49 GMT
Expires on: 2025-08-27 19:33:49 GMT
Last modified on: 2020-08-27 19:33:50

Registrant Info
Secret
123 H
Secret
secret

Technical Info
Secret
123 H
Secret
secret

Billing Info
Secret Co.
123 Hidden Elm Lane
Secret City, NY
secret@supersecret.com

Updating the Name Servers

Registrars will typically point your domain at their own temporary landing pages by default until you are ready.

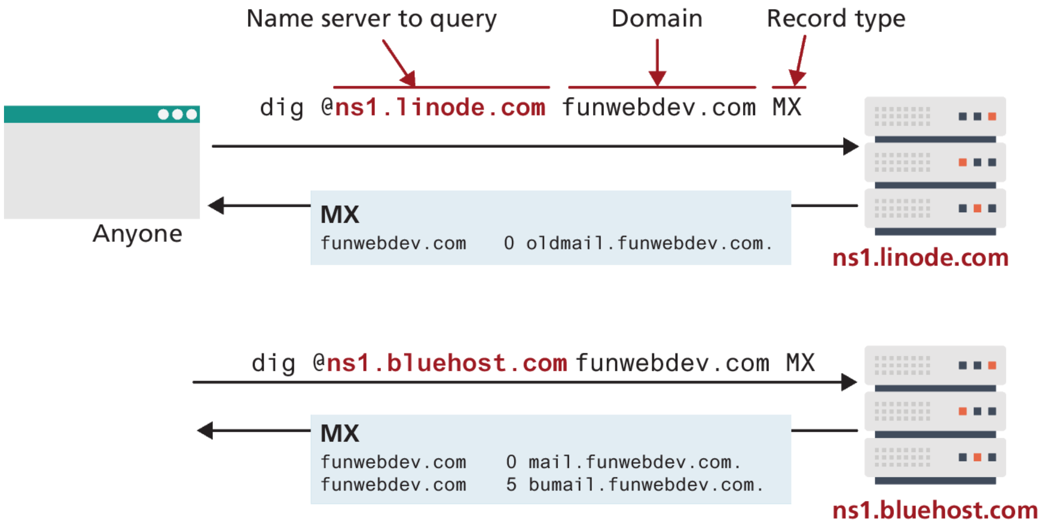
When you finally do purchase hosting (described in the next section), you will simply associate your new host's name servers with your domain on the registrar’s name servers.

When you update your name server, the registrar, on your behalf, updates your name server records on the top-level domain (TLD) name servers, thereby starting the process of updating your domain name for anyone who types it.

Checking Name Servers

Updating records in DNS may require at least 48 hours to ensure that the changes have propagated throughout the system. With so long to wait, you must be able to confirm that the changes are correct **before** that 48-hour window.

Linux has some helpful command-line tools to facilitate name server queries such as **nslookup** and **dig**.



```
dig @ns1.linode.com funwebdev.com MX
```

```
MX
funwebdev.com     0 oldmail.funwebdev.com.
```

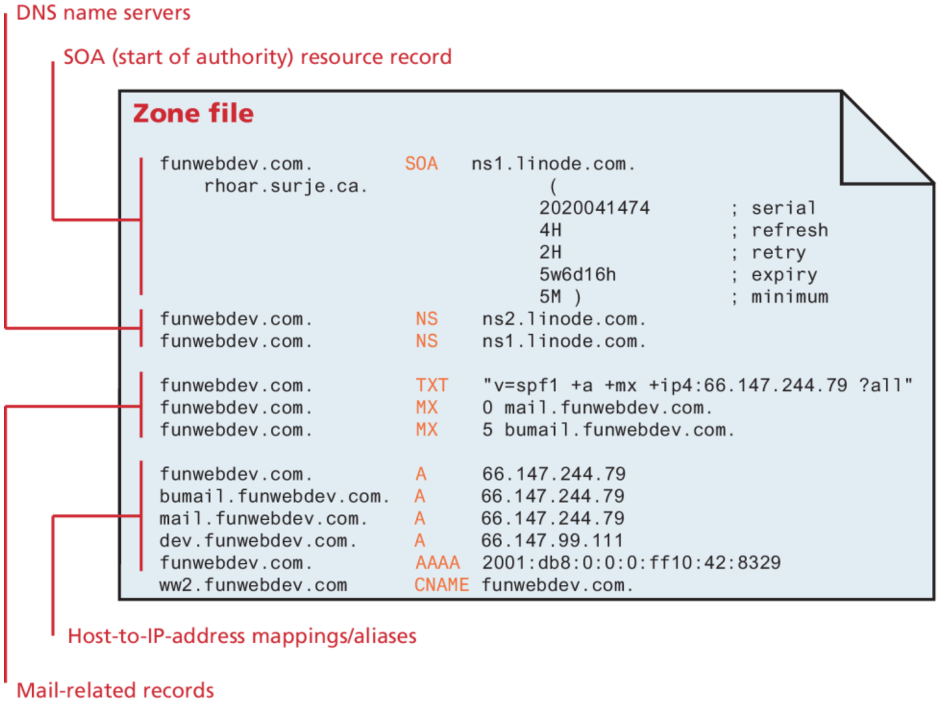
```
dig @ns1.bluehost.com funwebdev.com MX
```

```
MX
funwebdev.com     0 mail.funwebdev.com.
funwebdev.com     5 bumail.funwebdev.com.
```

DNS Record Types

In practice, all of a domain's records are stored in a single file called the DNS **zone file**.

Typically the DNS zone file is administered through a web interface on your host that lets you set one record at a time.



DNS name servers

SOA (start of authority) resource record

Zone file

```
funwebdev.com.          SOA   ns1.linode.com.
    rhoar.surje.ca.                 (
                                    2020041474        ; serial
                                    4H                ; refresh
                                    2H                ; retry
                                    5w6d16h           ; expiry
                                    5M )              ; minimum
funwebdev.com.           NS    ns2.linode.com.
funwebdev.com.           NS    ns1.linode.com.

funwebdev.com.           TXT   "v=spf1 +a +mx +ip4:66.147.244.79 ?all"
funwebdev.com.           MX    0 mail.funwebdev.com.
funwebdev.com.           MX    5 bumail.funwebdev.com.

funwebdev.com.           A     66.147.244.79
bumail.funwebdev.com.    A     66.147.244.79
mail.funwebdev.com.      A     66.147.244.79
dev.funwebdev.com.       A     66.147.99.111
funwebdev.com.           AAAA  2001:db8:0:0:0:ff10:42:8329
ww2.funwebdev.com        CNAME funwebdev.com.
```

Host-to-IP-address mappings/aliases

Mail-related records

Mapping Records

A records and **AAAA records** are identical except *A* records use IPv4 addresses and *AAAA* records use IPv6. Both of them simply associate a hostname with an IP address. These are the most common entries and are used whenever a user requests a domain through a browser.

Canonical Name (CName) records allow you to point multiple subdomains to an existing *A* record. This allows you to update all your domains at once by changing the one *A* record. However, it doubles the number of queries required to get resolution for your domain, making *A* records the preferred technique.

Mail Records

Mail Exchange (MX) records provide the location of the SMTP servers to receive email for this domain.

Just like the A records, they resolve to an IP address, but unlike the HTTP protocol, SMTP allows redundant mail servers for load distribution or backup purposes.

To support multiple destinations for one domain, MX records not only require an IP address but also a ranking.

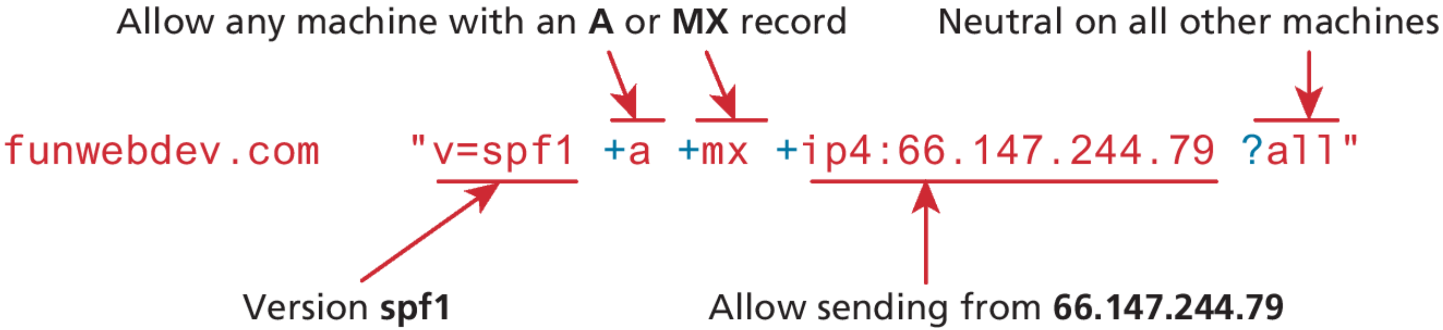
Authoritative Records

Name server (NS) records tell everyone what name servers to use for this domain. There can be (and should be) multiple name servers listed for redundancy.

Start of Authority (SOA) record contains information about how long this record is valid (called time to live [TTL]), together with a serial number that gets incremented with each update to help synchronize DNS.

Validation Records

TXT records and **Sender Policy Framework (SPF) records** are used to reduce email spam by providing another mechanism to validate your mail servers for the domain.


Allow any machine with an **A** or **MX** record
Neutral on all other machines
funwebdev.com "v=spf1 +a +mx +ip4:66.147.244.79 ?all"
Version **spf1**
Allow sending from **66.147.244.79**

Reverse DNS

You know how DNS works to resolve an IP address given a domain name. **Reverse DNS** is the reverse process, whereby you get a domain name from an IP address.

A **pointer (PTR) record** is created with a value taking the IP address prepended in *reverse order* to the domain **in-addr.arpa**

The IP address 66.147.244.79 becomes the PTR entry.

funwebdev.com PTR 79.244.147.66.in-addr.apra

Web Server Hosting Options

The deployment of your website is crucial since your users will be interacting with a server (host) first and foremost.

The three broad categories of web hosting are

- shared hosting,
- collocated hosting,
- and dedicated hosting.

Shared Hosting

Shared hosting is renting space for your site on a server that will host many sites on the same machine as illustrated in Figure 17.10.

This class of hosting is divided into two categories:

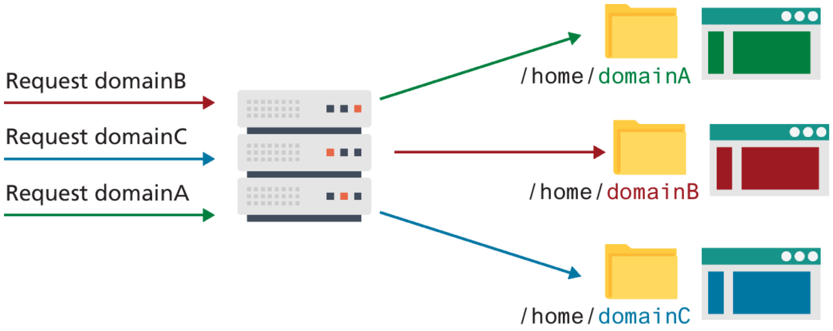
- **simple shared hosting** and
- **virtualized shared hosting**.

Simple shared hosting

Simple shared hosting is a hosting environment in which clients receive access to a folder on a web server but cannot increase their privileges to configure any part of the operating system, web server, or database.

If you want to install software on the server, most shared hosts do not permit it.

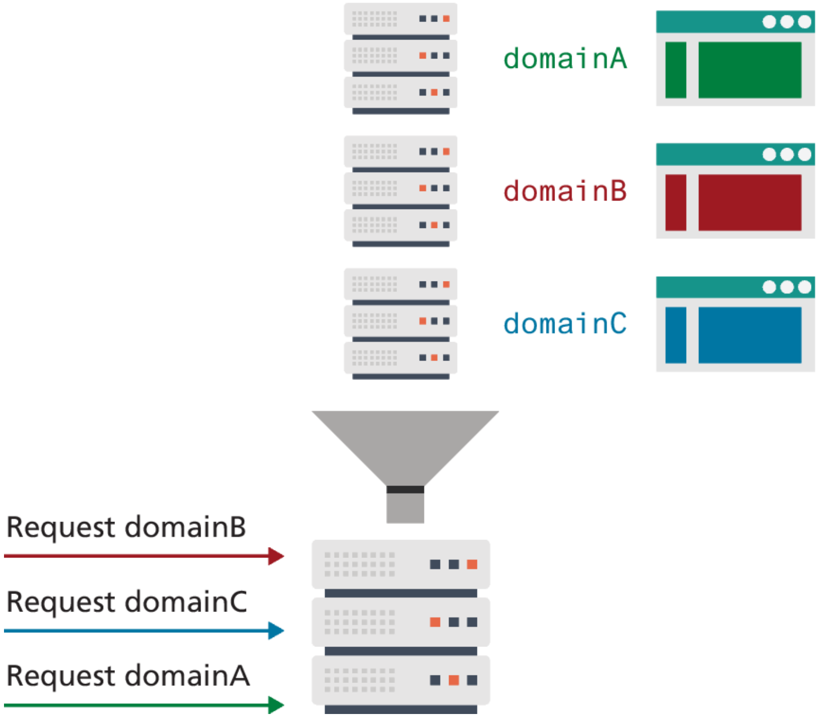
Poor performance is a more common problem with shared hosts.



Virtualized Shared Hosting

Virtualized shared hosting is a variation on the shared hosting scheme, where instead of being given a username and a home directory on a shared server, you are given a virtual server, with root access.

We call each operating system a **virtual server**



Pearson

Dedicated and Collacated Hosting

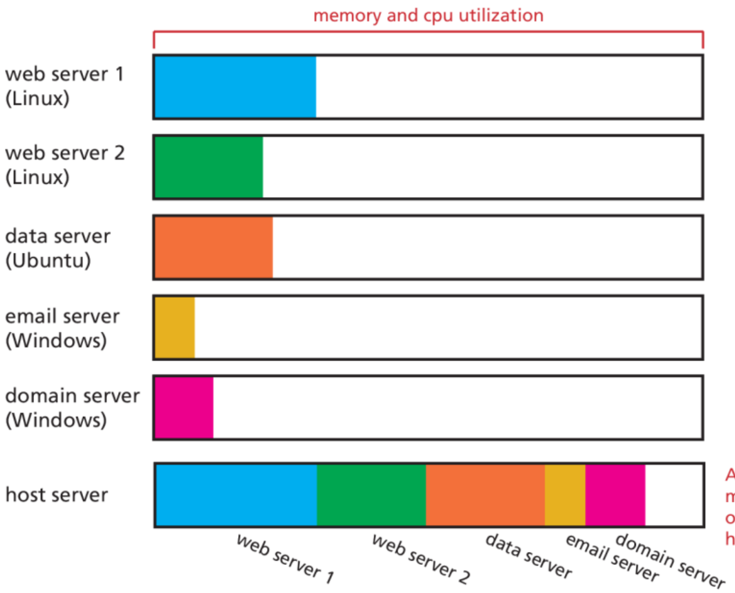
Dedicated hosting is when a physical server is rented to you in its entirety inside the data center. Hardware is normally standardized by the hosting center (with a few options to choose from), and the host takes care of any hardware issues.

Collocated hosting is almost like dedicated hosting, except rather than rent a machine, you outright purchase, build, and manage the machine yourself.

Cloud hosting leverages a distributed network of computers (cloud), which, in theory, can adapt more quickly in response to user needs than a configuration with a single physical server.

Server Virtualization

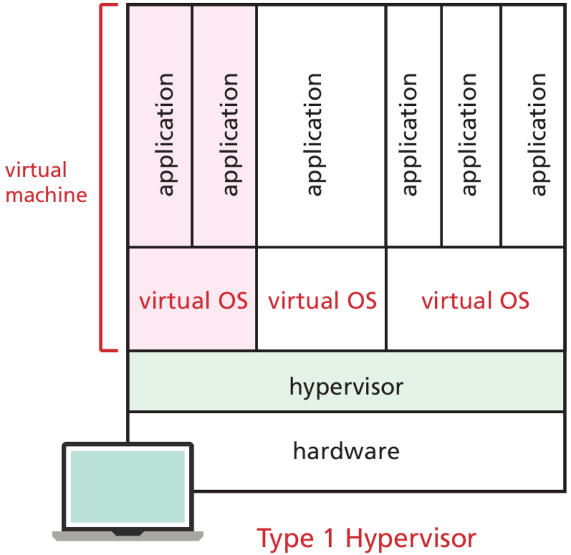
Server virtualization allows an administrator to turn a single computer into multiple computers, thereby saving on hardware and energy consumption.



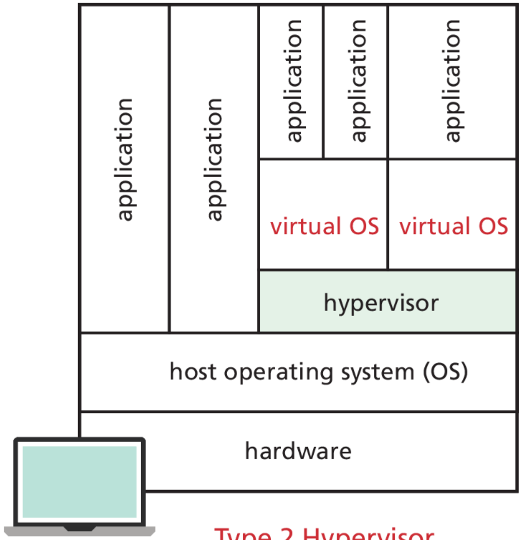
A virtualized server can be much more efficient in terms of energy consumption and hardware costs.

Hypervisors

The special software that makes virtual servers possible is generally referred to as a **hypervisor**.



Type 1 Hypervisor



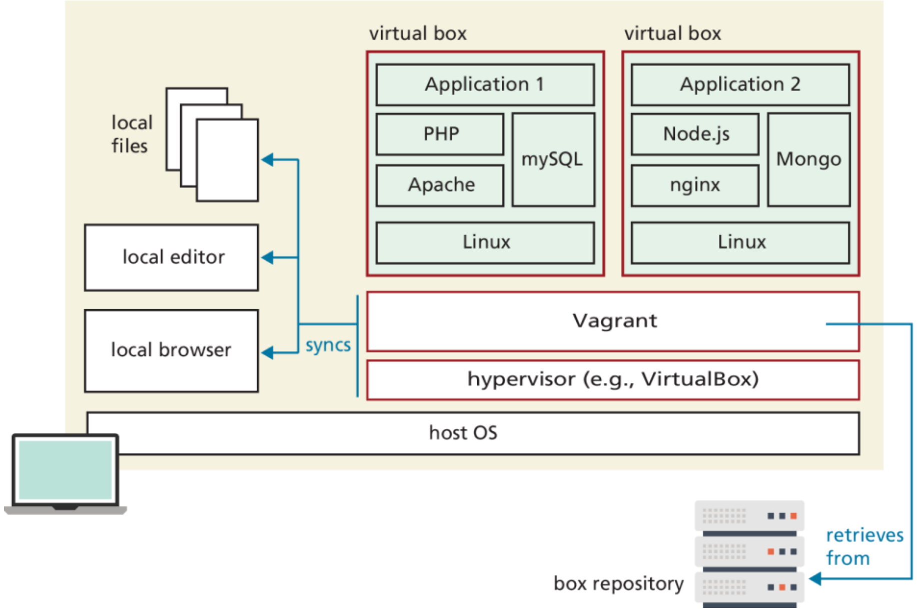
Type 2 Hypervisor

Vagrant

For instance, the popular open-source **Vagrant** tool works with a Type 2 hypervisor

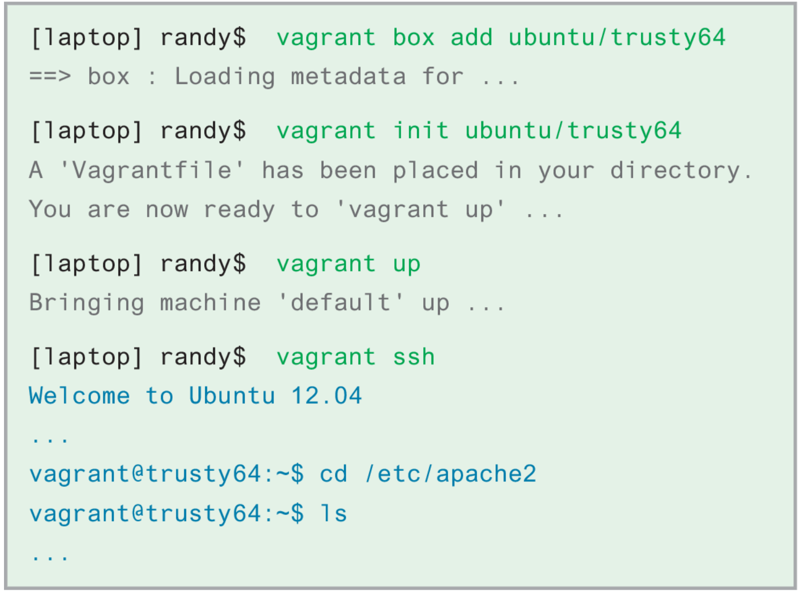
A team might create a Vagrant "box" that has the operating system, web server, database management system, programming languages, and other software installed and configured.

This box can then be shared with the rest of the team, thereby ensuring consistency.



Working with Vagrant Example

Terminal



```
[laptop] randy$  vagrant box add ubuntu/trusty64
==> box : Loading metadata for ...

[laptop] randy$  vagrant init ubuntu/trusty64
A 'Vagrantfile' has been placed in your directory.
You are now ready to 'vagrant up' ...

[laptop] randy$  vagrant up
Bringing machine 'default' up ...

[laptop] randy$  vagrant ssh
Welcome to Ubuntu 12.04
...
vagrant@trusty64:~$ cd /etc/apache2
vagrant@trusty64:~$ ls
...
```

This downloads the specified ISO box onto your local computer.

This initializes the Vagrant configuration file.

Creates a virtual machine using the current configuration.

Use the SSH command to connect to this virtual box. As fas as our local computer is concerned, we have connected to an external computer.

We can now run commands on this "external" computer system.

This particular box only contains the operating system (Ubuntu). We will have to install and configure Apache, mysql, etc.

Alternately, we could have instead downloaded a box that already has this software installed.

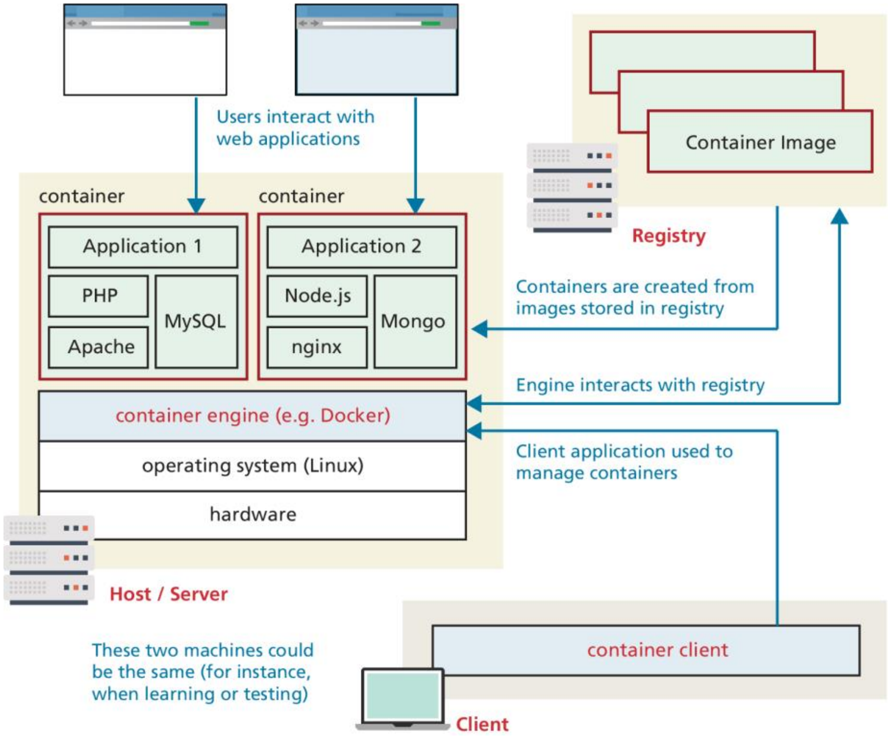
Containers

Containers allow a single machine with a single operating system to run multiple similar server instances.

The open-source **Docker** project has become a very popular method for deploying applications within these containers. A Docker container is a “snapshot” of the operating system, applications, and files needed to run a web application.

Alternative container management tools like Kubernetes achieve the same outcome as Docker, while proprietary tools from cloud hosts (such as Amazon Elastic Container Service) achieve the same end with their own tools.

Container-based virtualization


Users interact with web applications
container
container
Application 1
PHP
MySQL
Apache
Application 2
Node.js
Mongo
nginx
container engine (e.g. Docker)
operating system (Linux)
hardware
Host / Server
Container Image
Registry
Containers are created from images stored in registry
Engine interacts with registry
Client application used to manage containers
container client
Client
These two machines could be the same (for instance, when learning or testing)

Cloud Virtualization

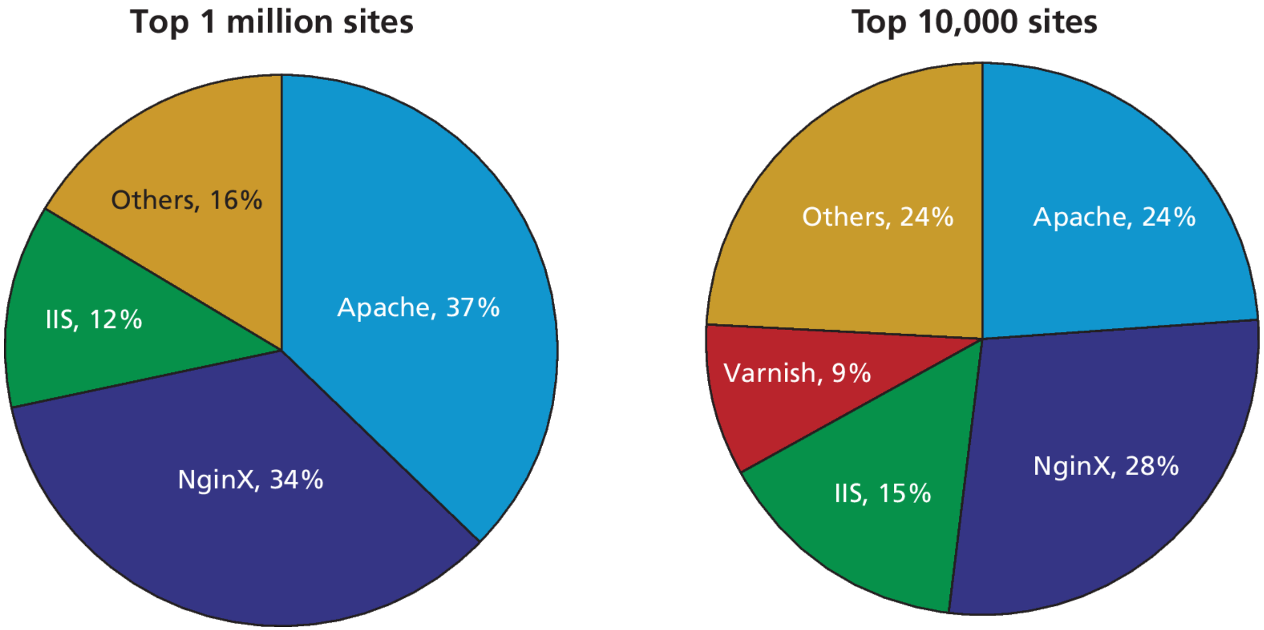
Cloud virtualization (sometimes referred to as just cloud computing) builds on virtualization technology and spreads it horizontally to multiple computers. Cloud computing promises something usually referred to as **elastic capacity/computing**, meaning that server capability can scale with demand.

There are a variety of service models available:

- **Infrastructure as a Service (IaaS)**.
- **Platform as a Service (PaaS)**.
- **Software as a Service (SaaS)**.

Linux and Web Server Configuration

Apache and NginX are the most popular web servers on the WWW



Configuration

When both Apache and NginX are started or restarted, they parse the **root configuration file**, which is normally writable by only root users.

In Apache, **directory-level configuration files** are also permitted. The files are normally named **.htaccess** (hypertext access), and they can reside inside any of the public folders served by Apache.

Inside of the configuration files, there are numerous **directives** you are allowed to make use of, each of which controls a particular aspect of the server.

Starting and Stopping the Server

Using the **systemctrl** command we can manage processes like Apache, NginX, and sql in the same way which makes things easier for us. To start Apache and NginX, we enter two commands using systemctrl:

- systemctl start httpd
- systemctl start nginx

To ensure that Apache starts when the machine boots, type the command:

- systemctl enable http

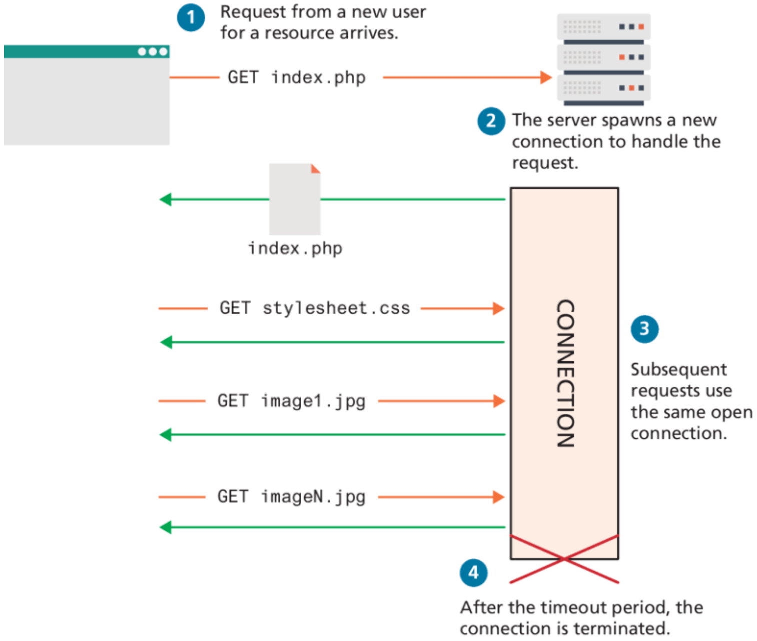
Every time you make a change to a configuration file, you must restart the service in order for the changes to take effect. This is done with

- systemctl restart httpd

Connection Management

Apache can run with multiple processes, each one with multiple threads. Key directives are:

- **Timeout** defines how long, in seconds, the server waits for receipts from the client
- **KeepAlive** is a Boolean value that tells Apache whether or not to allow more than one request per connection.
- **MaxKeepAliveRequests** sets how many requests to allow per persistent connection.
- **KeepAliveTimeout** tells the server how long to keep a connection alive between requests.



1 Request from a new user for a resource arrives.

2 The server spawns a new connection to handle the request.

3 Subsequent requests use the same open connection.

4 After the timeout period, the connection is terminated.

Data Compression

Chapter 2 showed you that the HTTP client request header **Accept-Encoding** indicates whether compression is supported by the client, and the response header **Content-Encoding** indicates whether the server is sending a compressed response.

Some files like .jpg files are already compressed, so recompressing them will use up CPU time needlessly. The directive below enables compression, but only for text, HTML, and XML files.

```
AddOutputFilterByType DEFLATE text/html text/plain text/xml
```

NginX uses a very similar way of configuring compression.

Encryption and SSL

All encrypted traffic requires the use of an X.509 public key certificate, which contains cryptographic keys as well as information about the site (identity).

While the background into certificates is described in Chapter 16, creating your own certificates is very straightforward, as illustrated by the shell script in Listing 17.1.

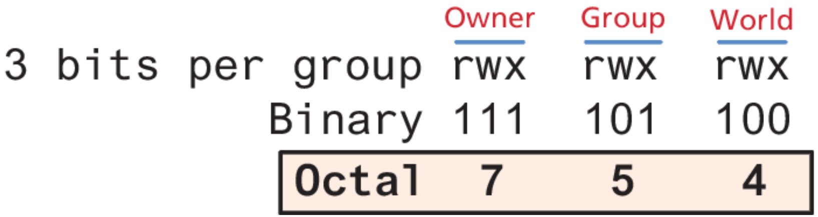
The **server.key** and the **server.crt** files are placed in a secure location (not visible to anyone except the Apache user) and referenced in Apache by adding to the root configuration file; the directives below pointing to the files.

```
SSLCertificateFile /path/to/this/server.crt
SSLCertificateKeyFile /path/to/this/server.key
```

Managing File Ownership and Permissions

Permissions are designed so that you can grant different users different abilities for particular files.

In Linux there are three categories of user: the owner, the group(s), and the world. Each file maintains three bits for all three categories of access (user, group, and world). The upper bit is permission to read, the next is permission to write, and the third is permission to execute.



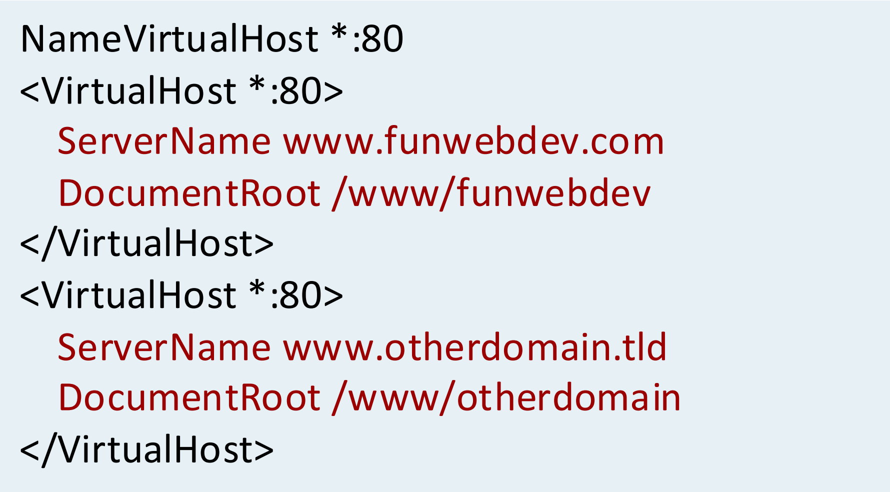
	Owner	Group	World
3 bits per group	rwx	rwx	rwx
Binary	111	101	100
Octal	7	5	4

Managing Multiple Domains on One Web Server

Every HTTP request to your web server contains the domain being requested.

A **VirtualHost** is an Apache configuration directive that associates a particular combination of server name IP and port to a folder on the server.

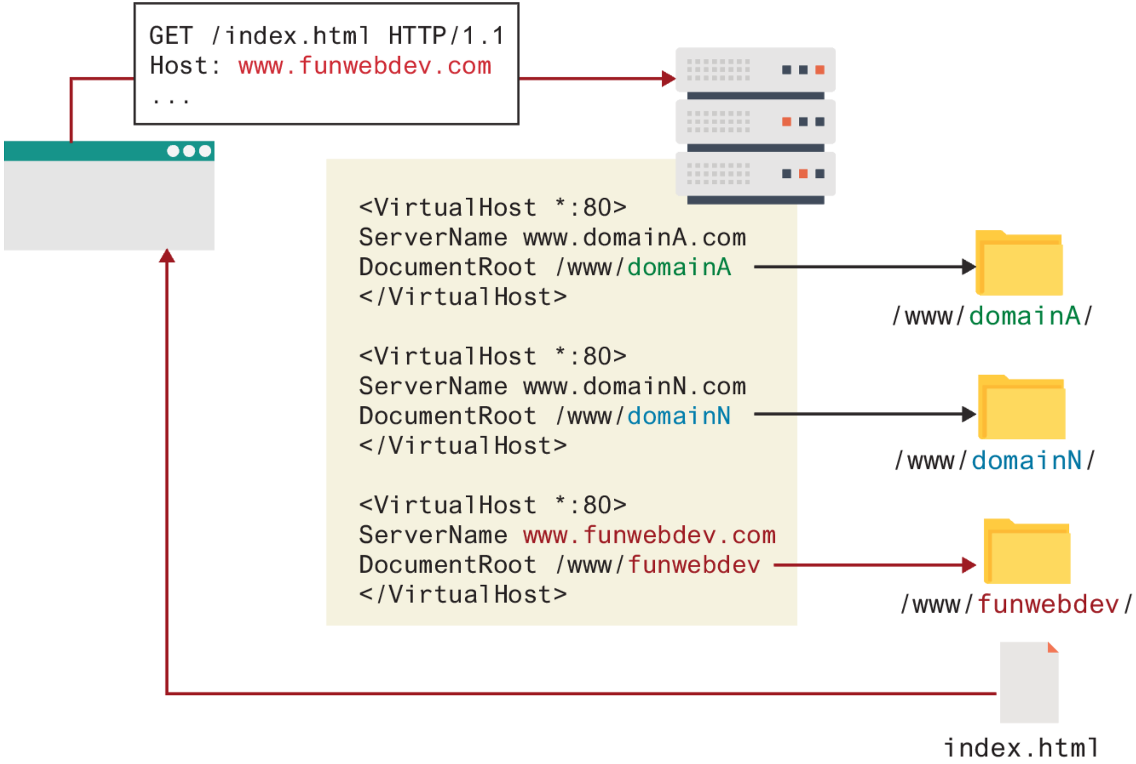
Going one step further, using **NameVirtualHost** allows you to use domain names instead of IP addresses



```
NameVirtualHost *:80
<VirtualHost *:80>
   ServerName www.funwebdev.com
   DocumentRoot /www/funwebdev
</VirtualHost>
<VirtualHost *:80>
   ServerName www.otherdomain.tld
   DocumentRoot /www/otherdomain
</VirtualHost>
```

LISTING 17.3 Apache VirtualHost directives in httpd.conf for two different domains on same IP address

Virtual host example



```
GET /index.html HTTP/1.1
Host: www.funwebdev.com
...
```

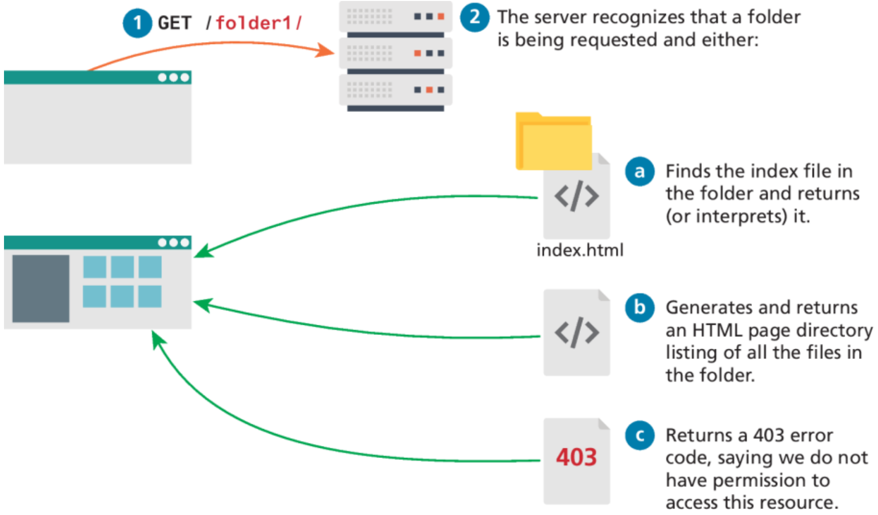
```
<VirtualHost *:80>
ServerName www.domainA.com
DocumentRoot /www/domainA
</VirtualHost>

<VirtualHost *:80>
ServerName www.domainN.com
DocumentRoot /www/domainN
</VirtualHost>

<VirtualHost *:80>
ServerName www.funwebdev.com
DocumentRoot /www/funwebdev
</VirtualHost>
```

Handling Directory Requests

When a folder is requested, the server must be able to determine what to serve in response. The server could choose a:

- file to serve
- display the directory contents, or
- return an error code

You can control this by adding **DirectoryIndex** and **Options** directives to the Apache configuration file, or adding "autoindex on" to your NginX configuration.



1 GET /folder1/
2 The server recognizes that a folder is being requested and either:
a Finds the index file in the folder and returns (or interprets) it.
index.html
b Generates and returns an HTML page directory listing of all the files in the folder.
c Returns a 403 error code, saying we do not have permission to access this resource.
403

Responding to File Requests

The most basic operation a web server performs is responding to an HTTP request for a static file.

The server sends the requested file, along with the relevant HTTP headers to signify that this request was successfully responded to.

A web server associates certain file extensions with MIME types that need to be *interpreted*.

If you wanted files with PHP as well as HTML extensions to be interpreted, you would add the directive below, which uses the PHP MIME types:

```
AddHandler application/x-httpd-php .php
AddHandler application/x-httpd-php .html
```

URL Redirection

Many times it would be nice to take the requested URL from the client and map that request to another location.

Back in Chapter 16, you learned about how nice-looking URLs are preferable to the sometimes-cryptic URLs that are useful to developers.

In Apache, there are two major classes of redirection,

- **public redirection** and
- **internal redirection** (also called **URL rewriting**).

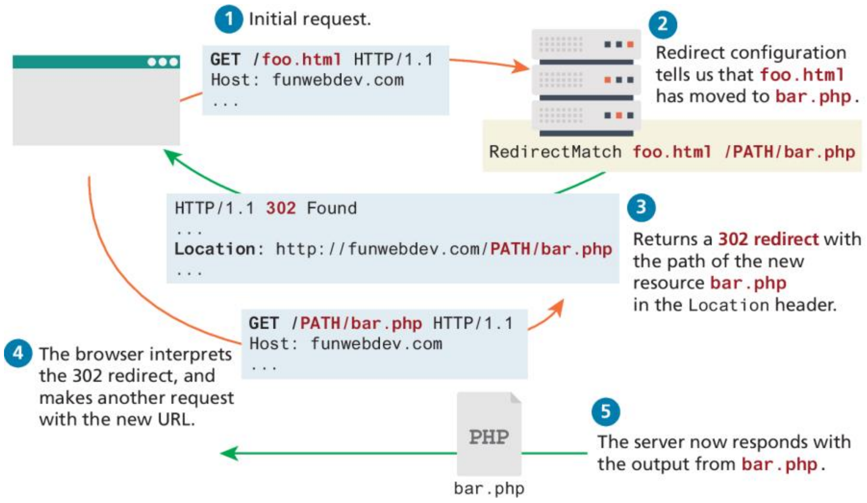
Public Redirection

In **public redirection**, you have a URL that no longer exists or has been moved.

If users have bookmarks to the old URLs, they will get 404 error codes when requesting them.

It is a better to inform users that their old pages have moved, using a HTTP 302 header.

In Apache, such URL redirection is easily achieved, using Apache directives



1 Initial request.

```
GET /foo.html HTTP/1.1
Host: funwebdev.com
...
```

2 Redirect configuration tells us that foo.html has moved to bar.php.

```
RedirectMatch foo.html /PATH/bar.php
```

```
HTTP/1.1 302 Found
...
Location: http://funwebdev.com/PATH/bar.php
...
```

3 Returns a 302 redirect with the path of the new resource bar.php in the Location header.

4 The browser interprets the 302 redirect, and makes another request with the new URL.

```
GET /PATH/bar.php HTTP/1.1
Host: funwebdev.com
...
```

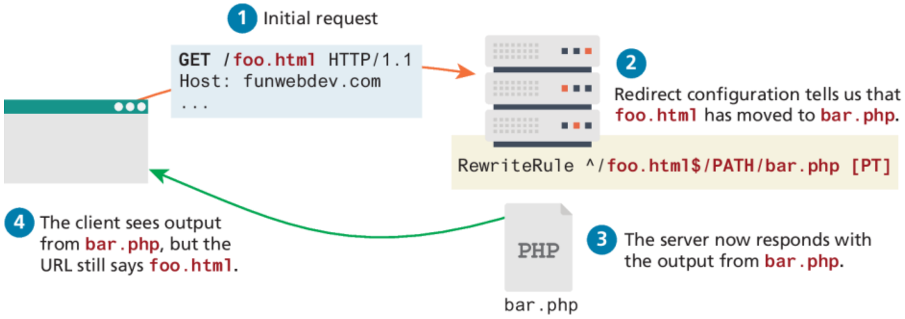
5 The server now responds with the output from bar.php.

Internal Redirection

The above redirections work well, but one drawback is that they notify the client of the moved resource.

This means that multiple requests and responses are required.

If the server had instead applied an **internal redirect** rule, the client would not know that **foo.html** had moved, and it would only require one request, rather than two.



1 Initial request

```
GET /foo.html HTTP/1.1
Host: funwebdev.com
...
```

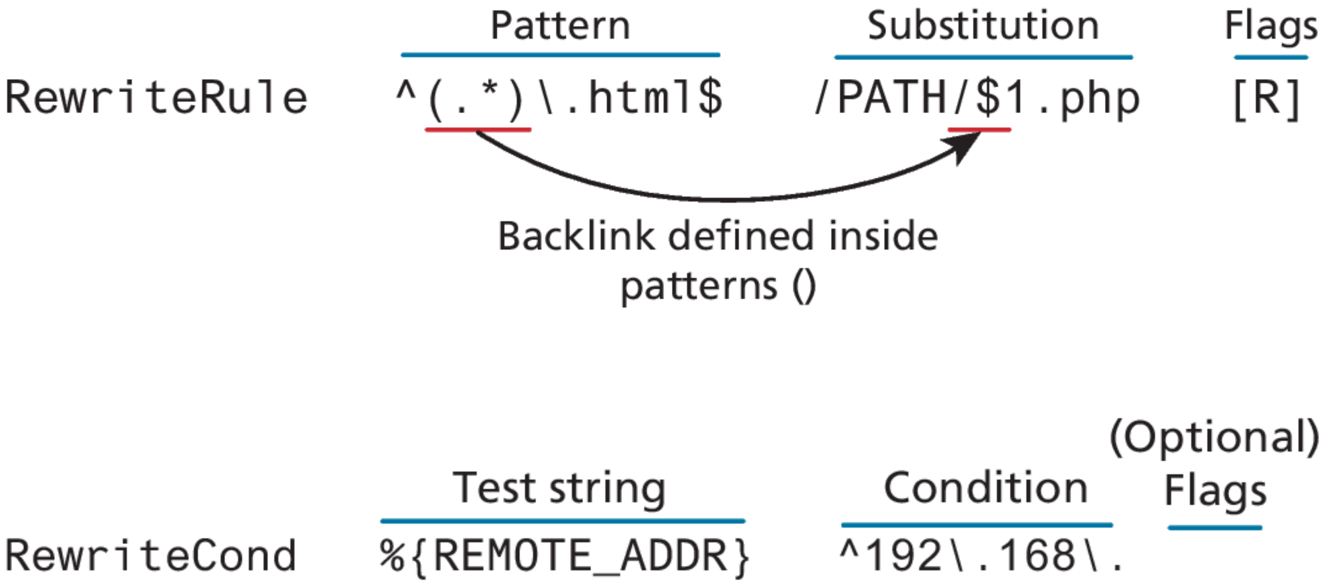
2 Redirect configuration tells us that foo.html has moved to bar.php.

```
RewriteRule ^/foo.html$/PATH/bar.php [PT]
```

3 The server now responds with the output from bar.php.

4 The client sees output from bar.php, but the URL still says foo.html.

RewriteRule Syntax



```
RewriteRule   ^(.*)\.html$   /PATH/$1.php   [R]
```

Pattern

Substitution

Flags

Backlink defined inside patterns ()

```
RewriteCond   %{REMOTE_ADDR}   ^192\.168\.
```

Test string

Condition

(Optional) Flags

Managing Access with .htaccess

Within a folder, **.htaccess** files are the directory- level configuration files used by Apache to store directives to apply to this particular folder.

Whether in Apache or NginX, you first create a password file. You then link that password file to the webserver's authentication mechanism.

```
AuthUserFile /location/of/our/passwordFile
AuthName "Enter your Password to access this secret folder"
AuthType Basic
require valid-user
```

LISTING 17.6 A sample .htaccess file to password protect a folder

Server Caching

Server caching is distinct from the caching mechanism built into the HTTP protocol (called **HTTP caching**). And the caching technique using PHP described in Chapter 13.

Server caching in Apache and NginX allows you to save copies of HTTP responses on the server so that the PHP script that created them won't have to run again.

- The first time any URL is requested, no cache exists and the page is created dynamically using the PHP script and then saved as the cached version with the key being the URL.
- Whenever subsequent requests for the same URL occur, the web server can decide to serve the cached page rather than create a fresh one based on configuration options you control.

Internal Monitoring

Internal monitoring reads the outputted logs of all the daemons to look for potential issues.

Apache and NginX provide some good default logging options, but also allow you to override what’s logged by configuring custom log types.

logrotate is the daemon running on most systems by default to handle log rotation, so that logs are periodically moved and deleted.

External Monitoring

External monitoring is installed off of the server and checks to see that connections to required services are open.

As part of a good security and administration policy, monitoring software like **Nagios** was illustrated back in Chapter 16. It can check for uptime and immediately notify the administrator if a service goes down.

Much like internal logs, external monitoring logs can be used to generate uptime reports and other visual summaries of your server. These summaries can help you determine if the host is performing adequately in the longer term.

Key Terms

- A records
- AAAA records
- canonical name (Cname) records
- cloud hosting
- cloud virtualization
- CName records
- collocated hosting
- containers
- continuous delivery (CD)
- continuous deployment
- continuous integration (CI)
- daemon
- dedicated hosting
- DevOps
- directives
- directory listings
- directory-level configuration files
- Docker
- elastic capacity/ computing
- external monitoring
- functional testing
- HTTP caching
- http heavy lifting
- hot-linking
- hypervisor
- Infrastructure as a Service (IaaS)
- infrastructure as code (IoC)
- integration tests
- internal monitoring
- internal redirection
- Linux shell script
- log rotation
- mail exchange (MX) record
- microservice architecture
- MME Types
- monolithic architecture
- name server (NS) records
- non-functional testing
- permissions
- Platform as a Service (PaaS)
- pointer record
- pointer (PTR) record
- public redirection
- regular expression syntax
- reverse DNS
- root configuration file
- Sender Policy Framework (SPF) records
- server virtualization
- shared hosting
- simple shared hosting
- Software as a Service (SaaS)
- Start of Authority (SOA) record
- systemctrl
- TXT records
- unit test
- URL rewriting
- Vagrant
- VirtualHost
- virtual server
- virtualized shared hosting
- wildcard certificate
- zone file

Copyright