#09 Web Deployment

CLIENT/SERVER COMPUTING AND WEB TECHNOLOGIES

Web Deployment

- Production Environment
- Reverse Proxy
- Docker
- Load Test

Production Environment

- App must be build or run in production mode.
- Case Study:
  - https://expressjs.com/en/advanced/best-practice-performance.html

Things to do

Dev part

- Use gzip compression
- Don't use synchronous functions
- Do logging correctly
- Handle exceptions properly

Ops part

- Set NODE_ENV to "production"
- Ensure your app automatically restarts
- Run your app in a cluster
- Cache request results
- Use a load balancer
- Use a reverse proxy

PM2

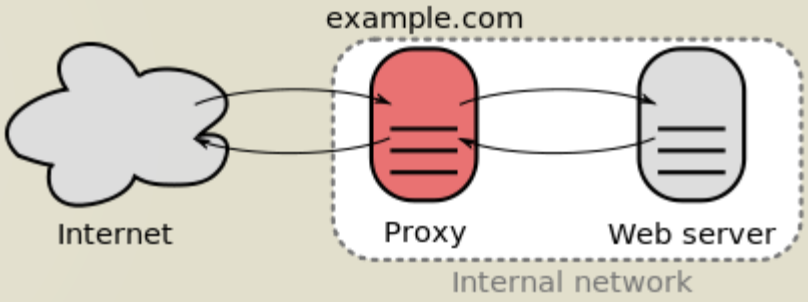
- Advanced, production process manager for Node.js
- Offering Cluster Mode
- Simple Application Declaration
  - Example: process.yml

```
apps:
  - script: src/main.js
    name: mms
    env:
      NODE_ENV: production
```

```
>> npm install pm2@latest -g
>> pm2 start process.yml
>> pm2 startup
```

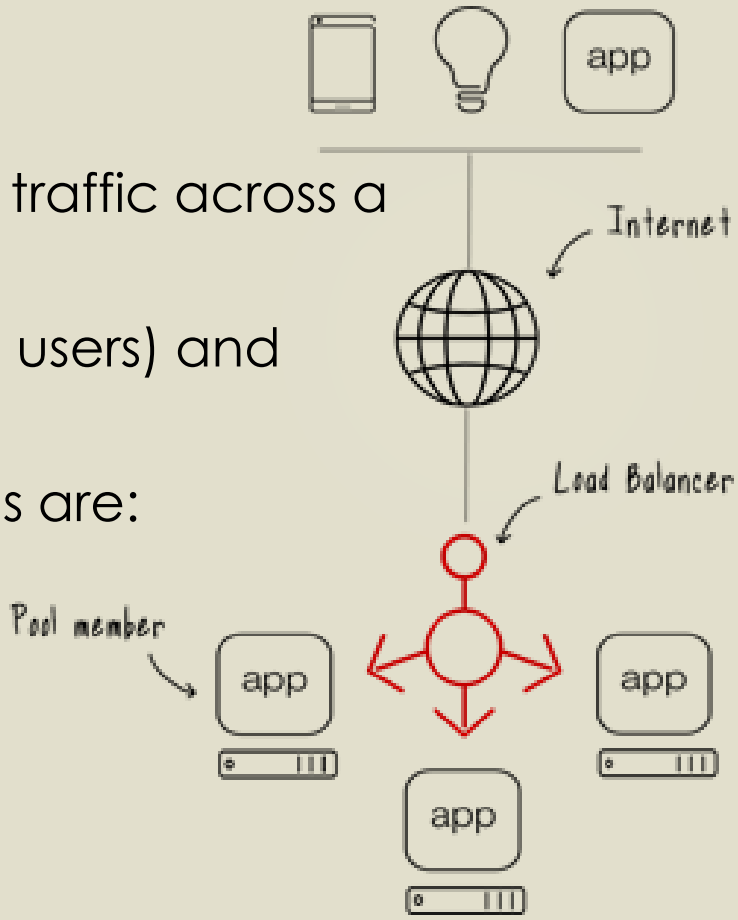
Reverse Proxy

- a type of proxy server that retrieves resources on behalf of a client from one or more servers.
- Unlike a forward proxy, which is an intermediary for its associated clients to contact any server, a reverse proxy is an intermediary for its associated servers to be contacted by any client.



Load Balancer

- Acts as a reverse proxy
- Distributes network or application traffic across a number of servers.
- To increase capacity (concurrent users) and reliability of applications.
- Some industry standard algorithms are:
  - Round robin
  - Weighted round robin
  - Least connections
  - Least response time



Case Study: Nginx as a reverse proxy

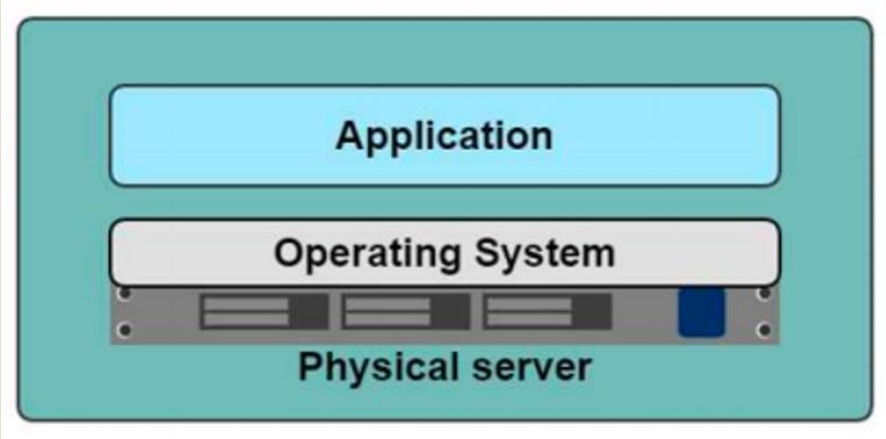
How To Set Up a Node.js Application for Production on Ubuntu 18.04

https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-18-04

Depolyment in Dark Ages

One Application on one physical server

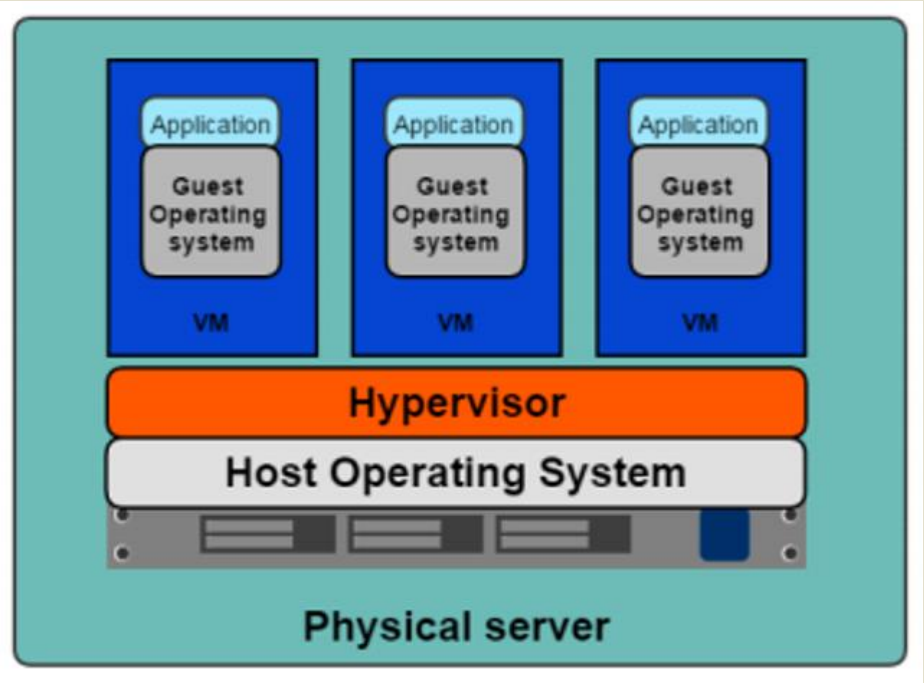
- Slow Deployment times
- Huge costs
- Wasted resources
- Difficult to scale
- Difficult to migrate
- Vendor lock in



Hypervisor-based Virtualization

One Physical server can contain multiple applications. Each application runs in a virtual machine (VM).

- Better resource pooling
  - One physical machine divided into multiple virtual machines
- Easier to scale
- VMs in the cloud
  - Rapid elasticity
  - Pay as you go model



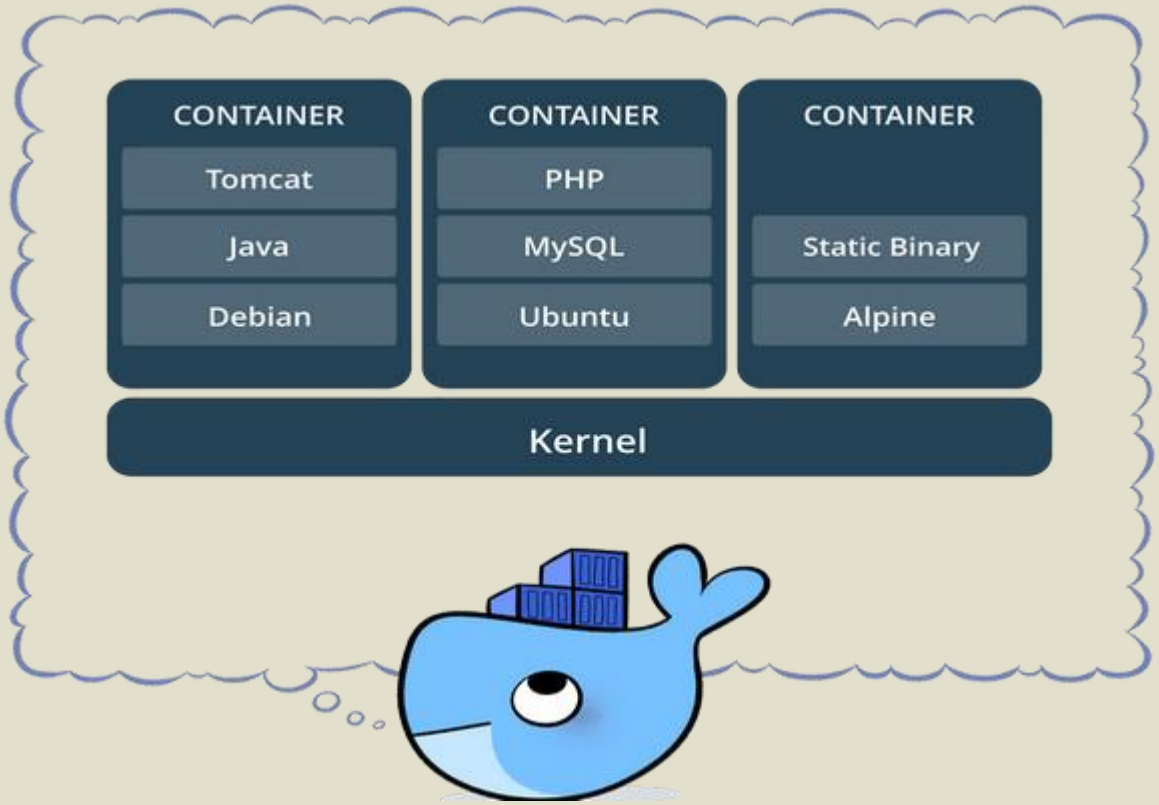
Microsoft Azure amazon web services vmware

Limitations of VMs

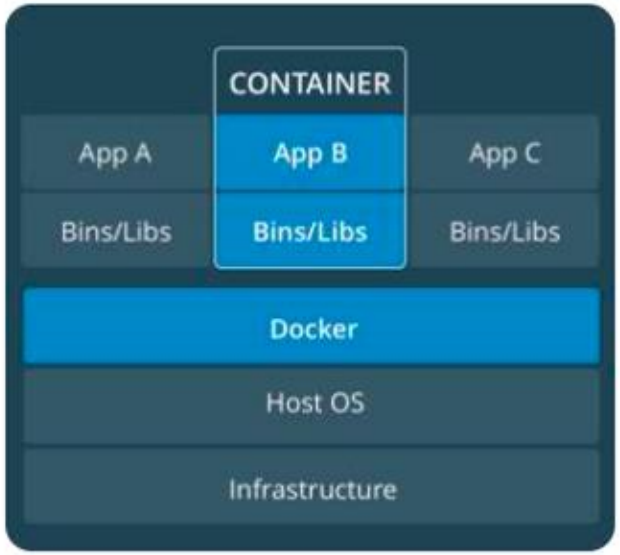
- Each VM stills requires
  - CPU allocation
  - Storage
  - RAM
  - An entire guest operating system
- The more VMs you run, the more resources you need
- Guest OS means wasted resources
- Application portability not guaranteed

What is container?

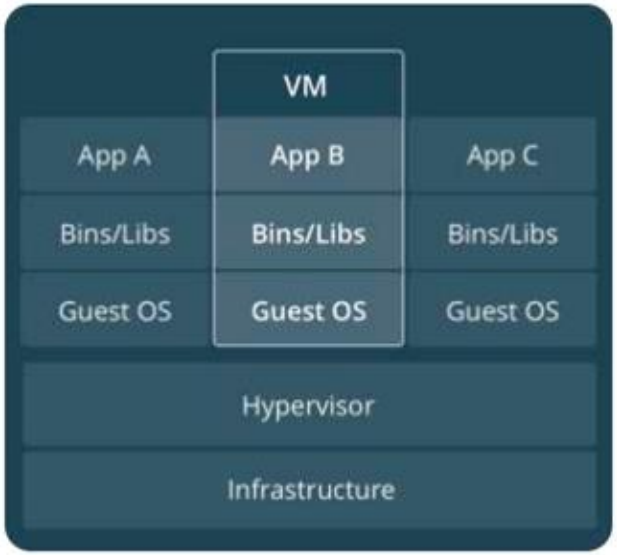
- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works with major Linux and Windows Server



Comparing Containers and VMs



Containers are an app level construct



VMs are an infrastructure level construct to turn one machine into many servers

Load Testing

- Kind of Performance Testing which determines a system's performance under real-life load conditions.
- Determine how the application behaves when multiple users access it simultaneously.
- This testing usually identifies -
  - The maximum operating capacity of an application
  - Determine whether the current infrastructure is sufficient to run the application
  - Sustainability of application with respect to peak user load
  - Number of concurrent users that an application can support, and scalability to allow more users to access it.

Goals of Load Testing

- Response time for each transaction
- Performance of System components under various loads
- Performance of Database components under different loads
- Network delay between the client and the server
- Software design issues
- Server configuration issues like a Web server, application server, database server etc.
- Hardware limitation issues like CPU maximization, memory limitations, network bottleneck, etc.

K6

k6 is a developer centric open source load testing tool for testing the performance of your backend infrastructure.

```
$ docker pull loadimpact/k6
$ cat script.js
import { check, sleep } from "k6";
import http from "k6/http";

export default function() {
   let res = http.get("https://httpbin.org/");
   check(res, {
      "is status 200": (r) => r.status === 200
   });
   sleep(3);
};

$ docker run -i loadimpact/k6 run --vus 10 --duration 30s -< script.js
```

K6 Metrics

Metric name	Type	Description
vus	Gauge	Current number of active virtual users
vus_max	Gauge	Max possible number of virtual users (VU resources are preallocated, to ensure performance will not be affected when scaling up the load level)
iterations	Counter	The aggregate number of times the VUs in the test have executed the JS script (the `default` function). Or, in case the test is not using a JS script but accessing a single URL, the number of times the VUs have requested that URL.
data_received	Counter	The amount of received data.
data_sent	Counter	The amount of data sent.
checks	Rate	Number of failed checks.

HTTP-specific built-in metrics

Metric name	Type	Description
`http_reqs`	Counter	How many HTTP requests has k6 generated, in total.
`http_req_blocked`	Trend	Time spent blocked (waiting for a free TCP connection slot) before initiating request. `float`
`http_req_waiting`	Trend	Time spent waiting for response from remote host (a.k.a. "time to first byte", or "TTFB"). `float`
`http_req_duration`	Trend	Total time for the request. It's equal to `http_req_sending + http_req_waiting + http_req_receiving` (i.e. how long did the remote server take to process the request and respond, without the initial DNS lookup/connection times). `float`

See completed list of metrics at https://docs.k6.io/docs/result-metrics

Case Study

K6 with InfluxDB and Grafana

https://docs.k6.io/docs/influxdb-grafana

Note: Docker Compose is a tool for defining and running multi-container Docker applications.