CS188
Scalable Internet Services

John Rothfels, 11/19/20

Recall

Where we left off last week...

(code on the jnr/perf branch of the course website)

- Use k6 load testing script
- Look at data in Honeycomb
- Identify performance problems
- Fix
- Re-measure

Investigation 🕵️

⁉️ **Hypothesis? What's going on to create the latency?**

- Too many computations? Not enough RAM?

Usually this is easy to see, but harder to identify the root cause of.

- CPU: look for trace spans that are long, events with high `duration_ms` or `SUM(duration_ms)`
- RAM: look for long lists, too much data being read out of your database

Investigation 🕵️

⁉️ **Hypothesis? What's going on to create the latency?**

- Too many computations? Not enough RAM?

One way to potentially reduce load on our server is to add **pagination** to our API. Instead of allowing the client to request *all* data in a single request, the client can instead request a smaller *page* of data per request. If the client needs more data, they can make another request for another page.

Pagination

schema.graphql

```
type Query {
 self: User

 surveys: [Survey!]!
 survey (surveyId: Int!): Survey

 candies: [UserCandy!]!
}
```

Pagination

schema.graphql

```
type Query {
 self: User

 surveys (limit: Int!): [Survey!]!
 survey (surveyId: Int!): Survey

 candies: [UserCandy!]!
}
```

Pagination

schema.graphql

```
type Query {
 self: User

 surveys (createdSince: String!): [Survey!]!
 survey (surveyId: Int!): Survey

 candies: [UserCandy!]!
}
```

Pagination

schema.graphql

```
type Query {
 surveys (cursor: Int): PaginatedSurveys
}


type PaginatedSurveys {
 surveys: [Survey!]!
 cursor: Int!
 has_next: Boolean!
}
```

Pagination

There are various ways you can add pagination to your API. Consider adding pagination to your API for your final paper!

Investigation 🕵️

⁉️ **Hypothesis? What's going on to create the latency?**

- Too many computations? Not enough RAM?

Either of these situations can potentially be mitigated by deploying our application onto different hardware. Increasing the size of the computer(s) running our app is **vertical scaling**. Increasing the number of computers running our app is **horizontal scaling**.

Aside: local load testing

When running a local dev server, some things (like vertical and horizontal scaling) are not possible or easy to test.

More generally, your application will behave differently in a deployed environment.

⁉️ Why?

Aside: local load testing

When running a local dev server, some things (like vertical and horizontal scaling) are not possible or easy to test.

More generally, your application will behave differently in a deployed environment.

⁉️ Why?

- Locally running `ts-node-dev` vs. node in production
- Dev builds vs. production builds
- Other applications run locally
- Distributed production environment
- Other code executing in production

Aside: local load testing

Most of the scaling strategies discussed in this class can be tested and measured against a local dev server. This is easier and recommended for your final projects.

Once you've exhausted everything you can do testing locally, you can try deploying your application to test its performance.

Running Servers

⁉️ So far, we've only focused on writing and running code on our own computer. What if we want to connect to real users? What kinds of problems do we need to think about?

Running Servers

⁉️ So far, we've only focused on writing and running code on our own computer. What if we want to connect to real users? What kinds of problems do we need to think about?

- Knowing ahead of time if my server is close to maxing out CPU / RAM
- Keeping my server running at all times
- Connecting my server to the network
- Providing dependencies to my server (runtime for executing code, databases, caches, etc)

Running Servers

⁉️ So far, we've only focused on writing and running code on our own computer. What if we want to connect to real users? What kinds of problems do we need to think about?

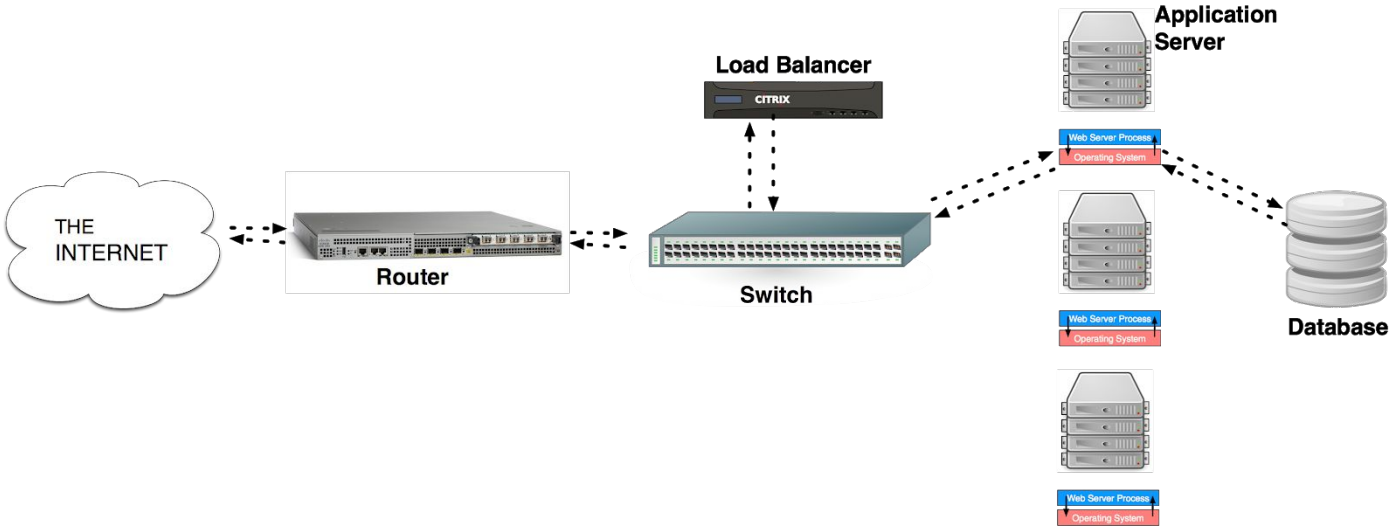
- **Keeping my server running at all times**

We call a server that is (almost) always available to handle requests **highly available (HA)**. HA is increasingly important. A common phrase targeted by businesses is "X nines"

- Three nines = 99.9% uptime = ~45 minutes down / month
- Four nines = 99.99% uptime = ~5 minutes down / month (business apps)
- Five nines = 99.999% uptime = ~5 minutes down / year (comms companies)

High Availability

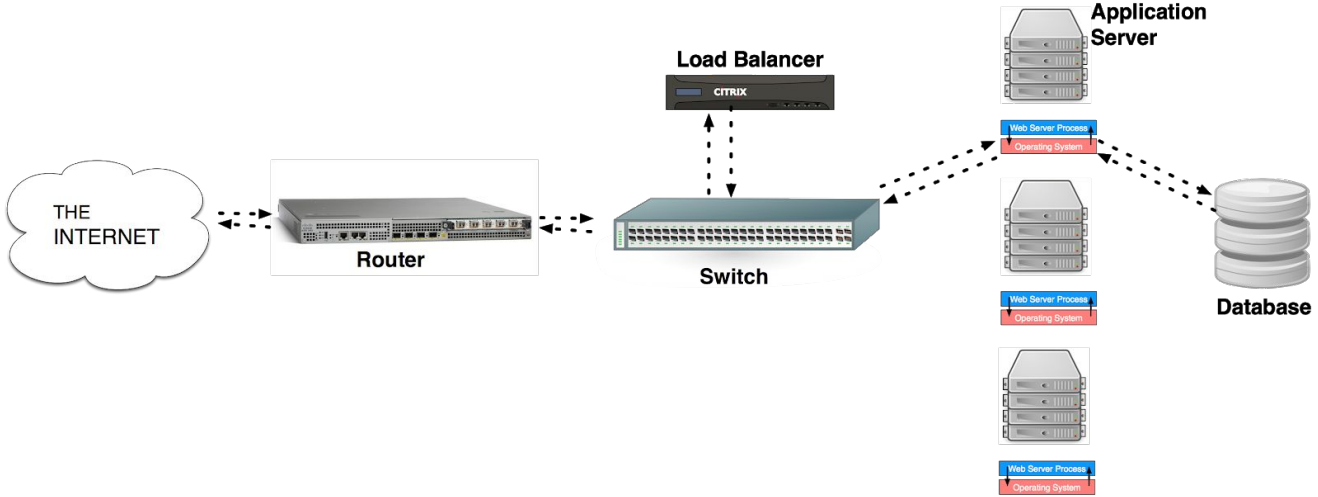
What are the possible causes of failure?



High Availability

What are the possible causes of failure?

- Server process dies?
- Application server fails?
- Load balancer fails?
- Switch fails?
- Internet fails?
- Database fails?
- Entire datacenter fails?



High Availability

Application server fails

High Availability

Application server fails

- We've already seen a lot here!
- Having process-level isolation reduces disruptions to a single process failure
- Running processes on separate hardware can increase isolation even further
- A load-balanced configuration means any single app server can go down and we can direct load elsewhere

High Availability

Database server fails

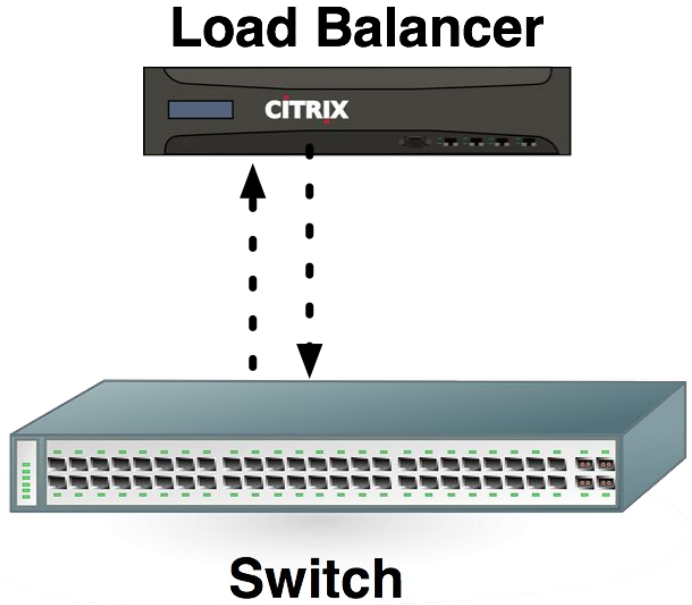
High Availability

Database server fails

- Most production SQL servers have **replication** enabled, which means you have one or more *replicas* of your master database available at all times.
- If master fails, promote a replica to master.
- Database *snapshots* are point-in-time copies of your data. If your database is completely wiped out, you can restore from the most recent snapshot.
    - Take regular snapshots! 😅

High Availability

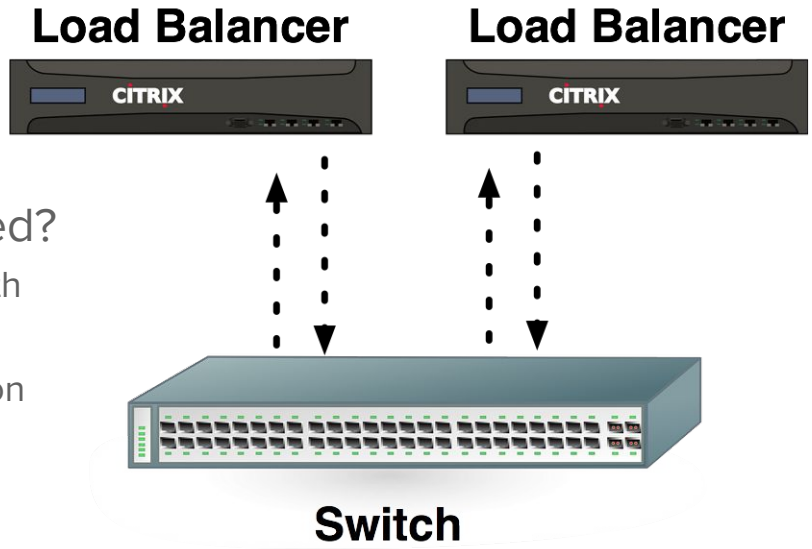
Load balancer fails



High Availability

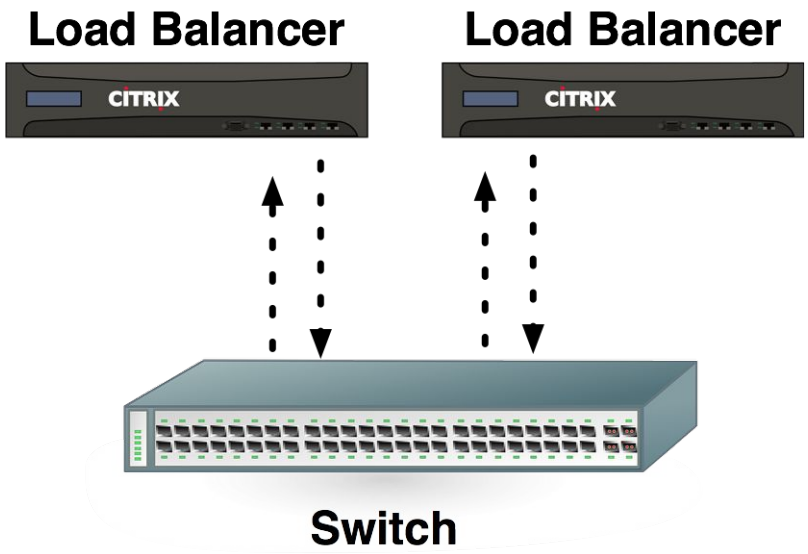
Load balancer fails

- Let’s buy two: primary and failover
- How do we detect when failure has occurred?
  - Load balancers use heartbeats to determine health
  - During failover, the secondary issues a network command (gratuitous ARP) so that other devices on the network that were communicating with the primary cleanly switch over to the secondary



High Availability

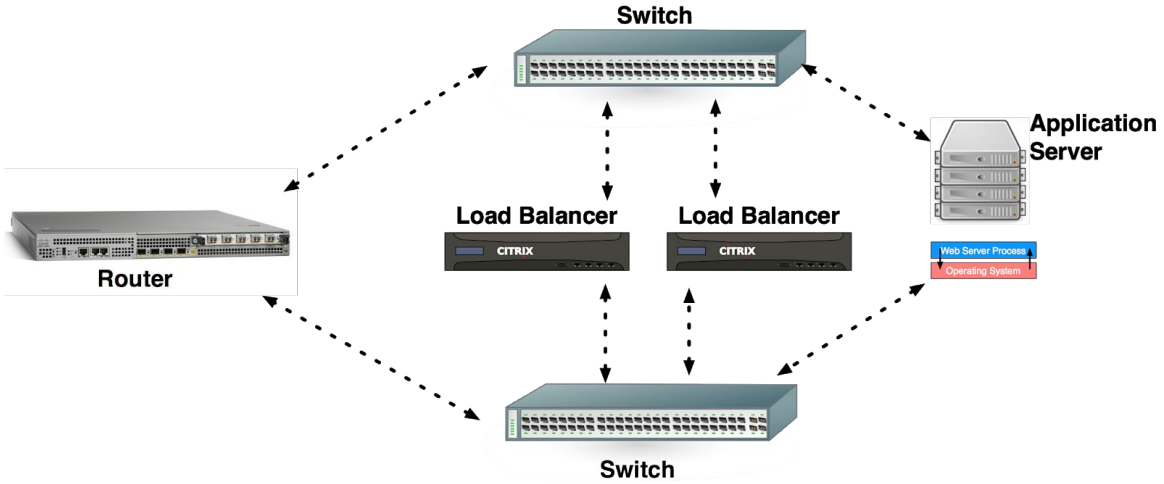
Switch fails



High Availability

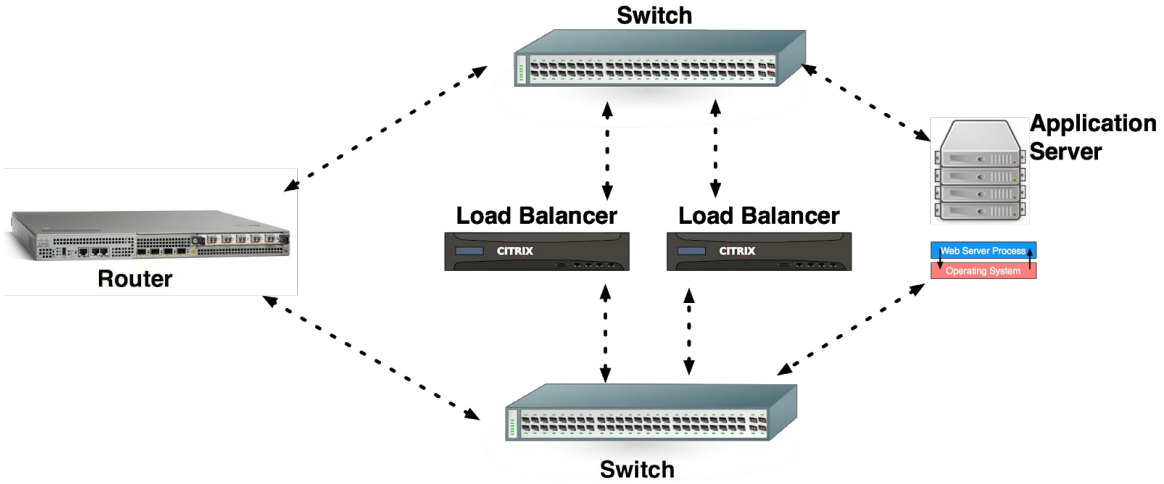
Switch fails

- Let’s buy two!



High Availability

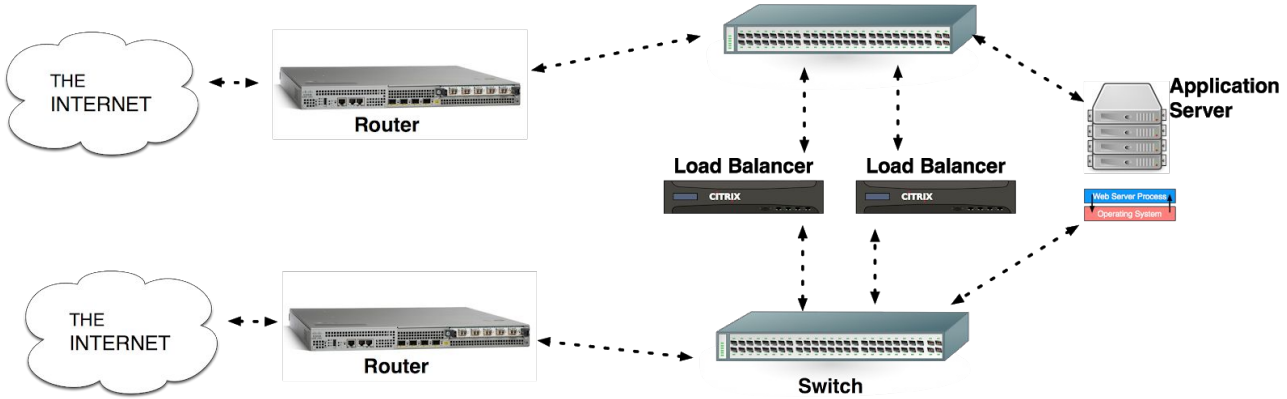
Router fails



High Availability

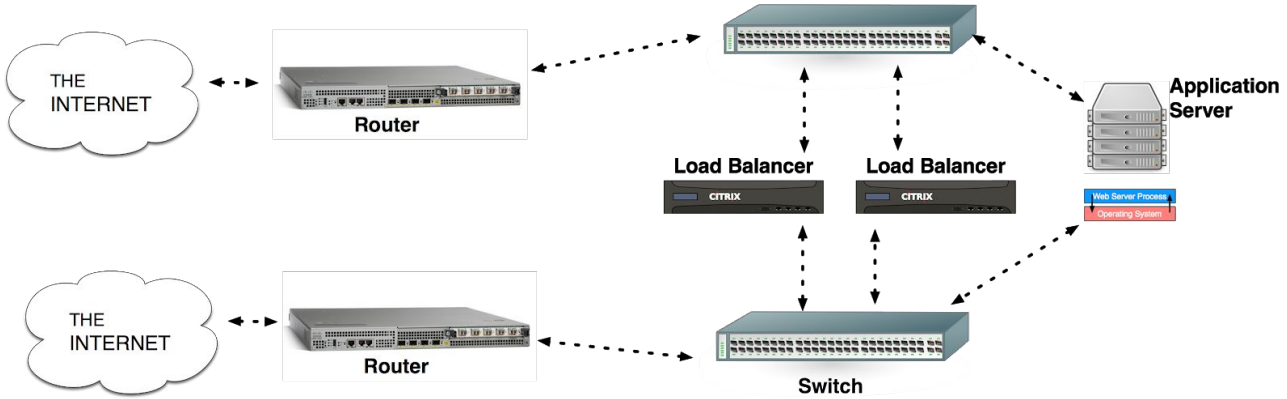
Router fails

- Let’s buy two!



High Availability

Internet fails



High Availability

Internet fails

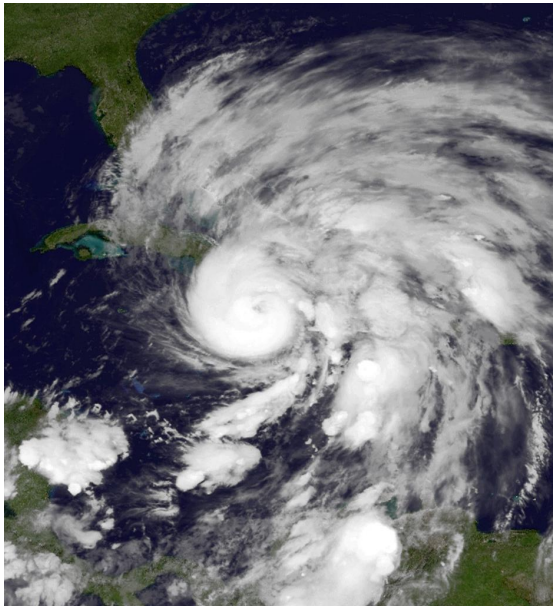
- Lets have two ISPs!
- How do we handle routing with multiple ISPs?
  - Outgoing traffic is easy since we control these decisions from our datacenter
    - Pick the cheapest or most reliable link
    - Pick the “closer” link
  - Incoming traffic is hard
    - We can’t directly tell clients how to reach our web app

High Availability

What else could possibly go wrong?

High Availability

What else could possibly go wrong?



High Availability

Hurricane Sandy takes data centers offline with flooding, power outages

Hosting customers stranded as generators in NY data centers run out of fuel.

by **Jon Brodkin** - Oct 30 2012, 9:25am PDT

Share Tweet 100

High Availability

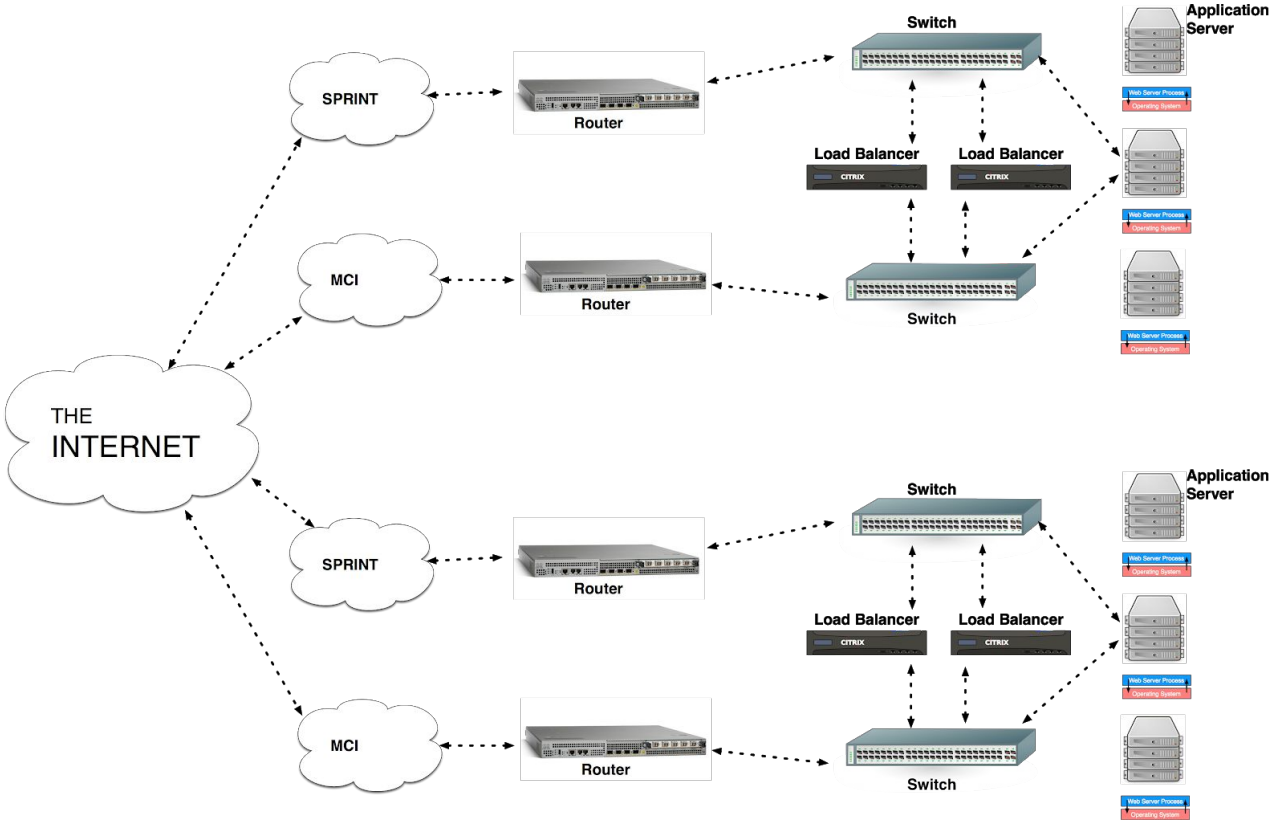
Availability Axiom (Pete Tenereillo):

The only way to achieve high-availability for browser based clients is the include the use of multiple A-records (DNS).

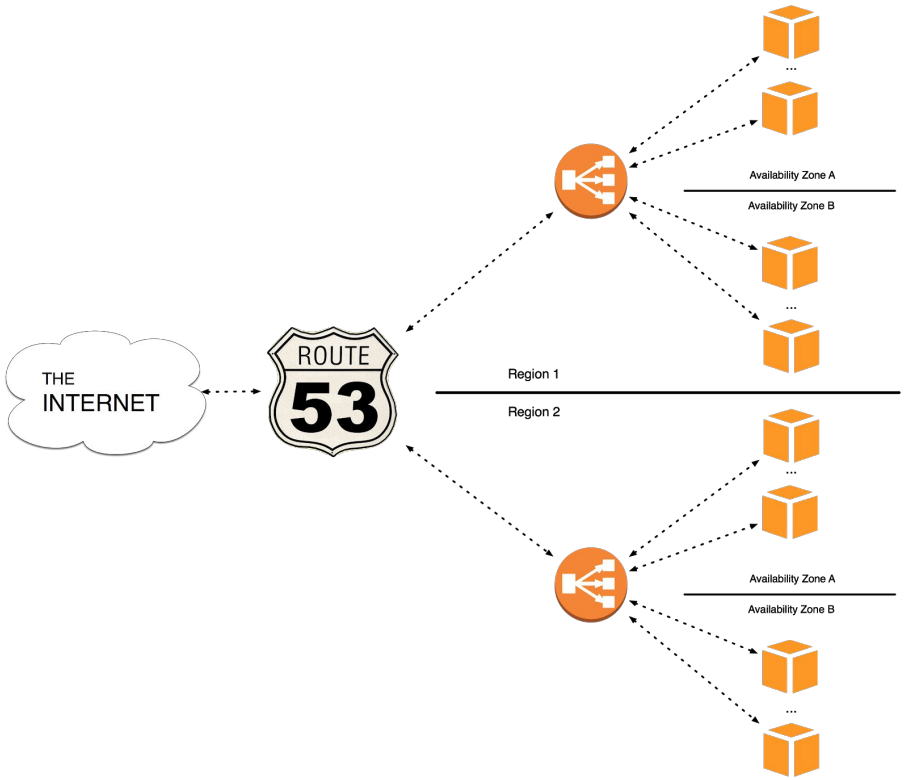
Result:

- For performance, we want to send the browser to one datacenter.
- For availability, we want to send the browser multiple A records.
- We end up having to make a choice between performance and availability!

High Availability



High Availability on AWS



AWS has regions and availability zones

- **Region**: think a city
- **Availability zone**: think a data center

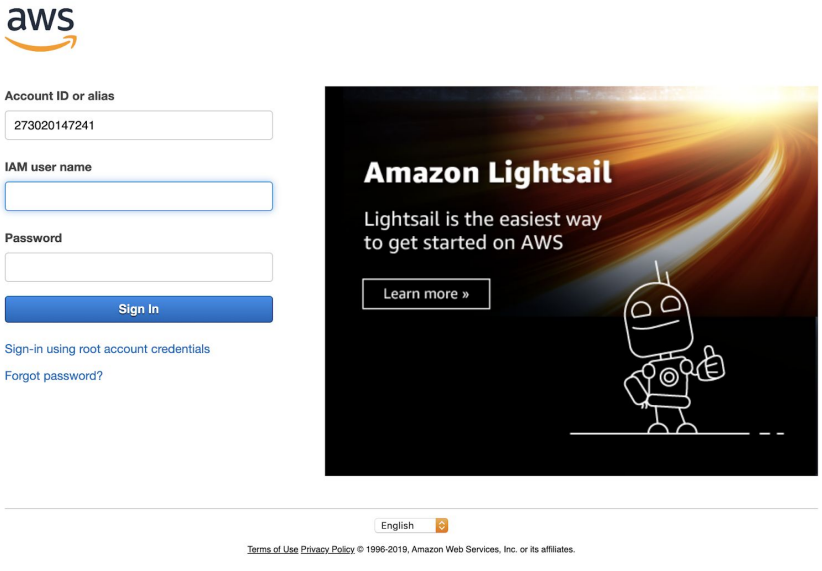
Failures between availability zones are not correlated*

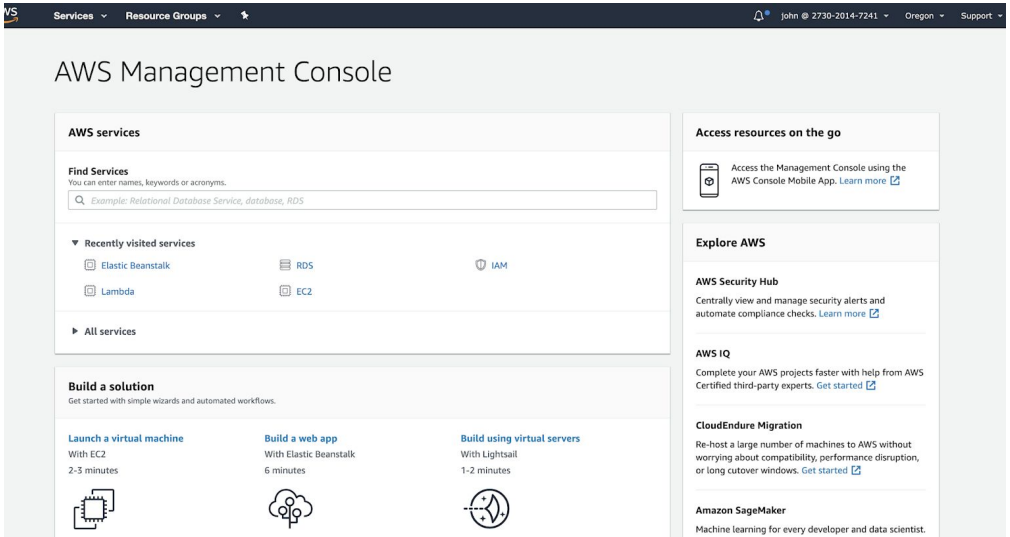
High Availability on AWS

The AWS tooling I provide you emphasizes testing scaling over High Availability.

We will use **terraform** to programmatically create different resources on AWS that can run your application.

AWS Resources: two approaches





I can login to AWS's UI and click around to create everything I need and wire everything together.

AWS Resources: two approaches

```
resource "aws_db_instance" "db" {
 name                   = "thecoolestdatabase"
 identifier             = "thecoolestdatabase"
 deletion_protection    = false
 skip_final_snapshot    = true
 kms_key_id             = aws_kms_key.db_key.arn
 allocated_storage      = 10
 max_allocated_storage  = 20
 storage_type           = "gp2"
 storage_encrypted      = true
 engine                 = "mysql"
 engine_version         = "8.0.11"
 instance_class         = "db.t3.micro"
 username               = "root"
 password               = "Wc9VfzWYnZJmE"
 port                   = 3306
 publicly_accessible    = true
 db_subnet_group_name   = var.subnet_group
 vpc_security_group_ids = [var.security_group]
 parameter_group_name   = "mysql-8-large-packet"
 multi_az               = false
 backup_retention_period = 7
 backup_window          = "10:30-11:30"
 maintenance_window     = "wed:09:00-wed:10:00"
}
```

Or I can define what my application needs declaratively (with text) and use a software program to do all the work.

terraform takes a collection of declared *resources* and determines the appropriate set of actions to take to provision those resources on AWS.

Terraform

Given a set of *resource* definitions, I can either:

- `terraform plan`
  - Instructs terraform to diff the current state of your provisioned resources against your resource definitions. The plan tells you the sequence of (create/update/destroy) operations terraform would take.
- `terraform apply`
  - Runs the terraform plan.

The result of `terraform apply` is a **terraform state file** (`terrraform.tfstate`) which is a JSON representation of all the resources provisioned during the last apply step.

Terraform

The result of `terraform apply` is a **terraform state file** (`terrraform.tfstate`) which is a JSON representation of all the resources provisioned during the last apply step.

On the next `terraform apply`, terraform uses the prior state to present a diff (so I know exactly what terraform plans to do to get my infrastructure up to date).

AWS Resources: two approaches

Approach #1: login to UI, to point-and-click

Approach #2: describe resources in text, use terraform

⁉️ Pros / cons?

AWS Resources: two approaches

Approach #1: login to UI, to point-and-click

- Pros:
  - I don’t have to learn software tools
- Cons:
  - AWS UI is terribly hard to use
  - How do people know when I make changes?
  - What if in a year I want to change from AWS to Azure or GCP?

AWS Resources: two approaches

Approach #2: describe resources in text, use terraform

- Pros:
  - Portable across different cloud providers
  - Changes to my infrastructure are recorded in git commits, get code review
  - I don’t have to use terrible AWS UI
  - Faster/safer (once you learn it)
- Cons:
  - I have to learn software tools

AWS Instructions

!!! Important rules and warnings !!!

AWS resources are scarce resources that **cost real $**.

Unless you have a specific reason to, *always* use micro instances.

- Example of a good reason: testing vertical scaling

AWS Instructions

!!! Important rules and warnings !!!

- Our AWS budget has a fixed limit. Don’t break the class for everyone!
- Whenever you are done with infrastructure, **REMOVE IT**!!
- Never keep important data on a server because it can go down at any time.
- I will periodically run a script that finds all instances that have been up longer than 4 hours.
    - Your instance will be terminated, and your team will be logged.
    - If this happens 3 times, I’ll revoke your team’s AWS credentials and you’ll have to complete the project doing only local testing only

AWS Instructions

!!! Important rules and warnings !!!

- Your team will receive an email with an AWS key and secret.
- **DO NOT CHECK THESE INTO ANY PUBLICLY ACCESSIBLE REPOSITORY.**
  - There are automated scripts that actively seek out AWS credentials. Why?
- Once you run terraform apply, it will generate a `terraform.tfstate` file.
  - Don't check this into your repository! Why?
  - Don't lose it either!
  - Recommendation: remove the terraform directory from your project, designate one teammate to handle infrastructure & deployment.

AWS Instructions

cd into your terraform directory. Modify `main.tf` (or any submodule) to declare the appropriate resources for your project. Then:

- `terraform init`
- `terraform apply`

It should take 5-10 minutes to provision your infrastructure.

AWS Instructions

To **REMOVE** everything you've provisioned, simply comment out all the lines in main.tf (or remove everything) and run:

- `terraform apply`

Remember, you should only provision infrastructure *that you're ready to load test against*. As soon as you're no longer testing, remove all your resources. You can re-provision them again when you're ready to test.

AWS Servers

I want:

- a web app served by Node.js
- that can be scaled horizontally & vertically
- has replicated DB storage
- load balancing
- zero-downtime deploys
- 3+ nines of uptime
- ...

AWS Servers

In this scenario, AWS provides you many building blocks but you still have to put all the pieces together.

Traditionally, you'll start by creating a VM to run your application server. AWS has a service called EC2 which lets you provision VMs. With a VM, you get SSH access.

⁉️ Why do we need to SSH?

AWS Servers

In this scenario, AWS provides you many building blocks but you still have to put all the pieces together.

Traditionally, you'll start by creating a VM to run your application server. AWS has a service called EC2 which lets you provision VMs. With a VM, you get SSH access.

⁉️ Why do we need to SSH?

- To install Node.js / other server dependencies
- To read or download logs (stored in files on the server)
- To connect to my DB (it might be protected by network security rules)

Aside: code portability

Problem:

How do I make sure my software runs correctly when it is moved from one computing environment to another?

Aside: code portability

Problem:

How do I make sure my software runs correctly when it is moved from one computing environment to another?

Answer:

- Run `provision.sh` on every server?
- Require every server can run that script?

Containers

Small digression: virtualization

- I’m AWS
- I have a server running a single customer’s app, the server is never at capacity
- How do I make more money?

Containers

Small digression: virtualization

- I’m AWS
- I have a server running a single customer’s app, the server is never at capacity
- How do I make more money?

Answer:

- Run more customer’s apps on the same server! 🎉

Containers

Small digression: virtualization

Allows multiple operating systems to run completely independently on a single machine.

Uses specialized software called Hypervisor that encapsulates a virtual (guest) OS and emulates hardware resources like CPU, memory so they can be shared.

Containers

Containers extend the ideas of virtualization

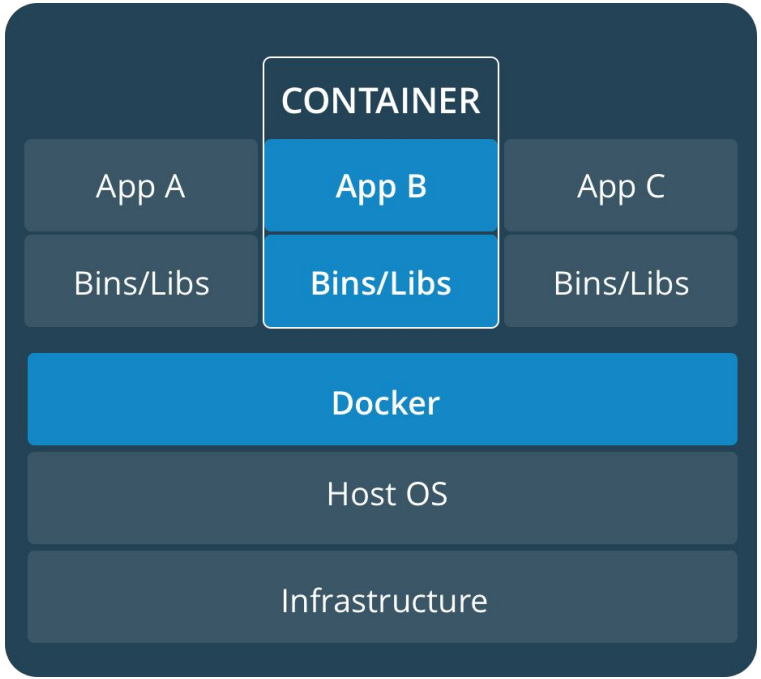
- Derived from core capability of Linux: Linux Containers (LXC)
- Emergence of Docker brought big boost in popularity
- Currently has broad industry buy-in & adoption

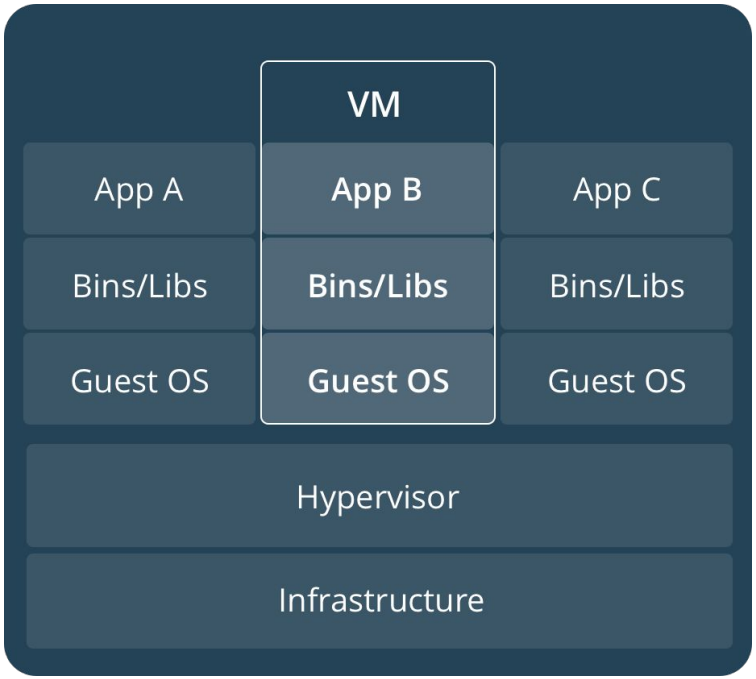
Containers

Extending virtualization

- Eliminate the Hypervisor and multiple VMs
- Containers are applications + their dependencies, packaged into virtual environments
- All running containers share a single instance of the host operating system and runs on the “bare metal” of the server (without OS virtualization)
- Much lighter-weight than traditional VMs

Containers





source

Containers

Pros:

- Lightweight: more efficient than VMs
- Portable: build locally, deploy to the cloud, run anywhere
- Loosely coupled: you can replace or upgrade one without disrupting others
- Scalable: you can increase and automatically distribute container replicas across a datacenter
- Secure: containers isolate your apps without configuration

AWS Servers

Using a newer virtualization technology called Docker, we can avoid having to provision a dedicated VM (where we have SSH access).

Instead, we create a Docker image which knows how to run our Node.js application server.

AWS has a service called ECS which lets us run Docker containers inside a “cluster” of VMs.

Dockerfile

Dockerfile specifies how to build your container/app:

- Start from some base container (e.g. “linux”)
- Run commands inside your container
  - install node, bundler
- Put files into your container (your app)
  - cp from local machine
- Tell the container how to start your app, what port to expose

Dockerfile

```
FROM ubuntu:16.04

RUN apt-get install -y nginx

RUN bundle install

COPY . /myapp

EXPOSE 3000

CMD ["rails", "server"]
```

Dockerfile

```
FROM ubuntu:16.04
RUN apt-get install -y nginx
RUN $INSTALL_APPSERVER_DEPENDENCIES
```

```
docker image save appserver:0.1
```

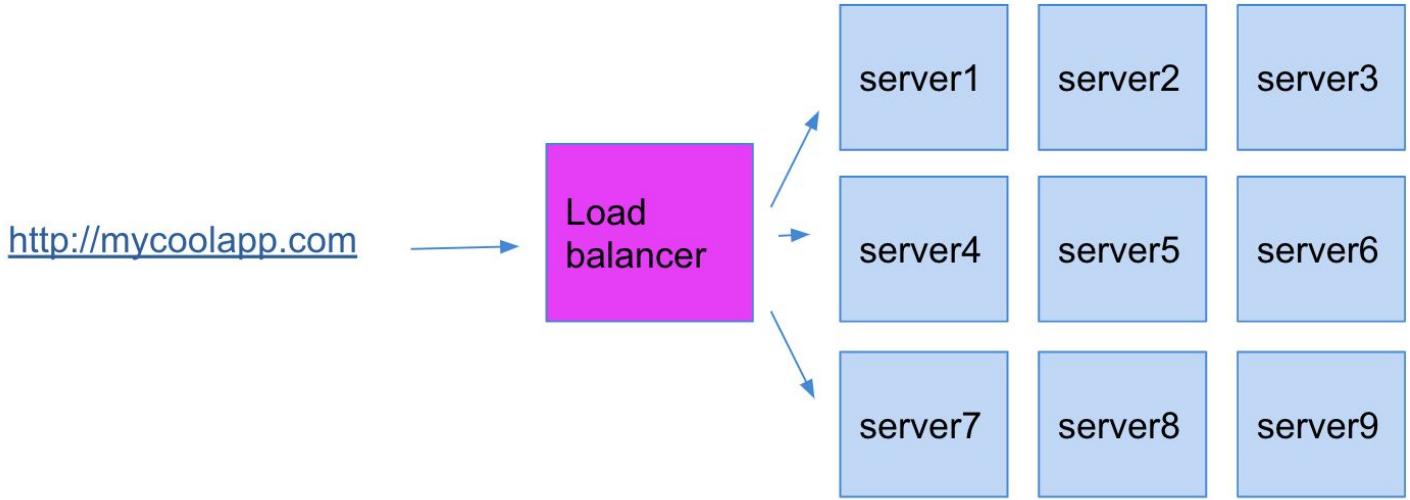
```
FROM appserver:0.1
...
```

Shipping Containers

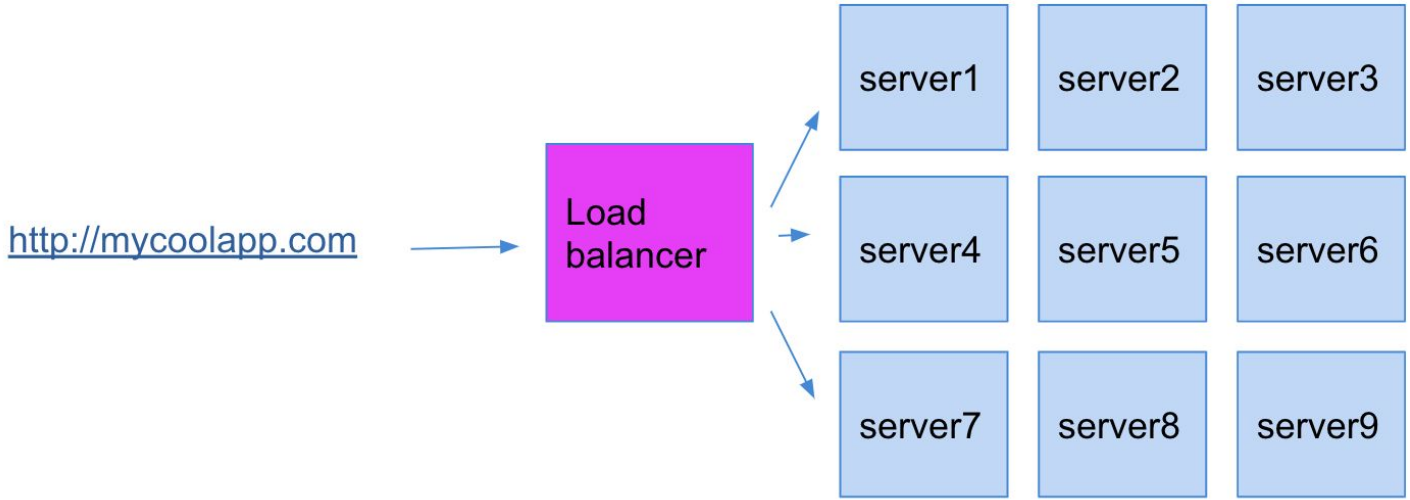
I have an app wrapped in a container...now what?

- It's a typical Node.js application
- Single process, single thread
- I want to run 9 of them serving http://mycoolapp.com

Shipping Containers

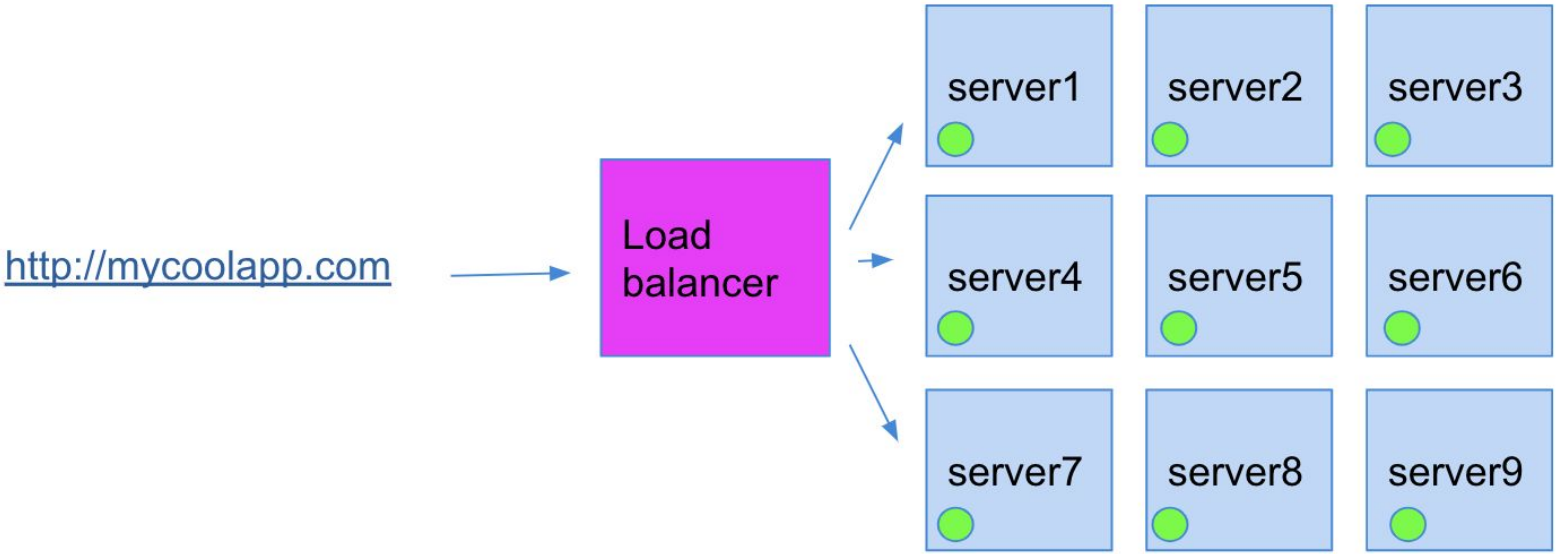


Shipping Containers



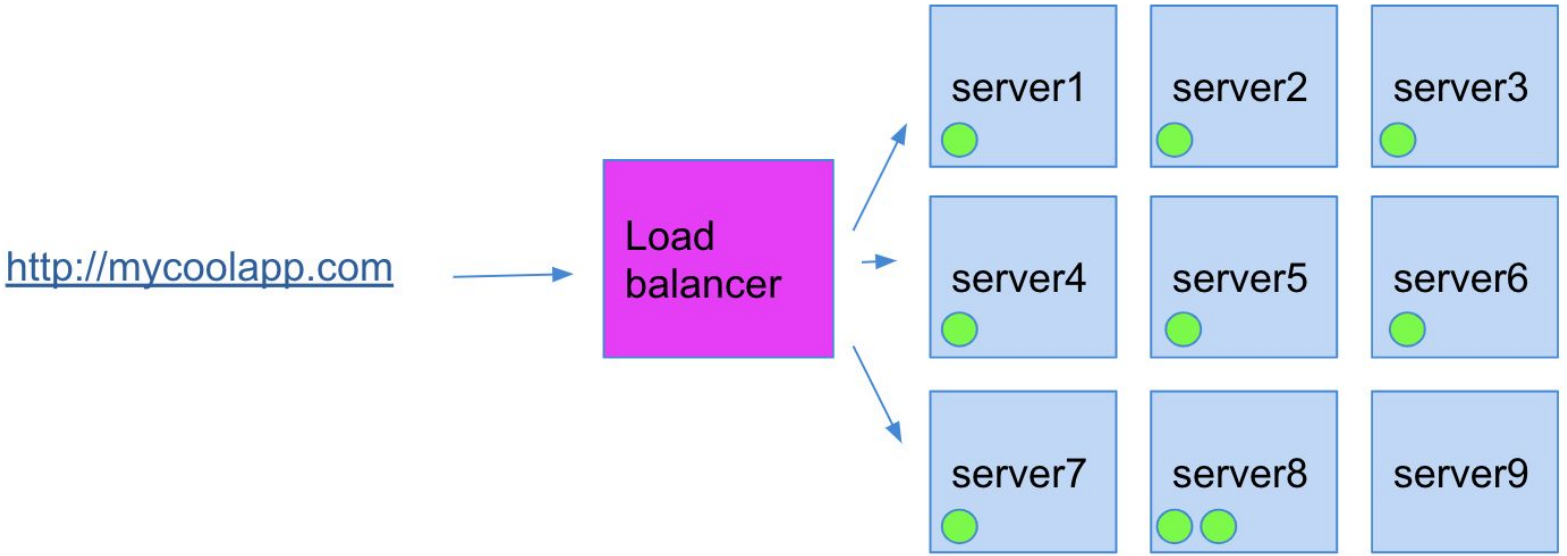
What is the most efficient way to run my containers?

Shipping Containers



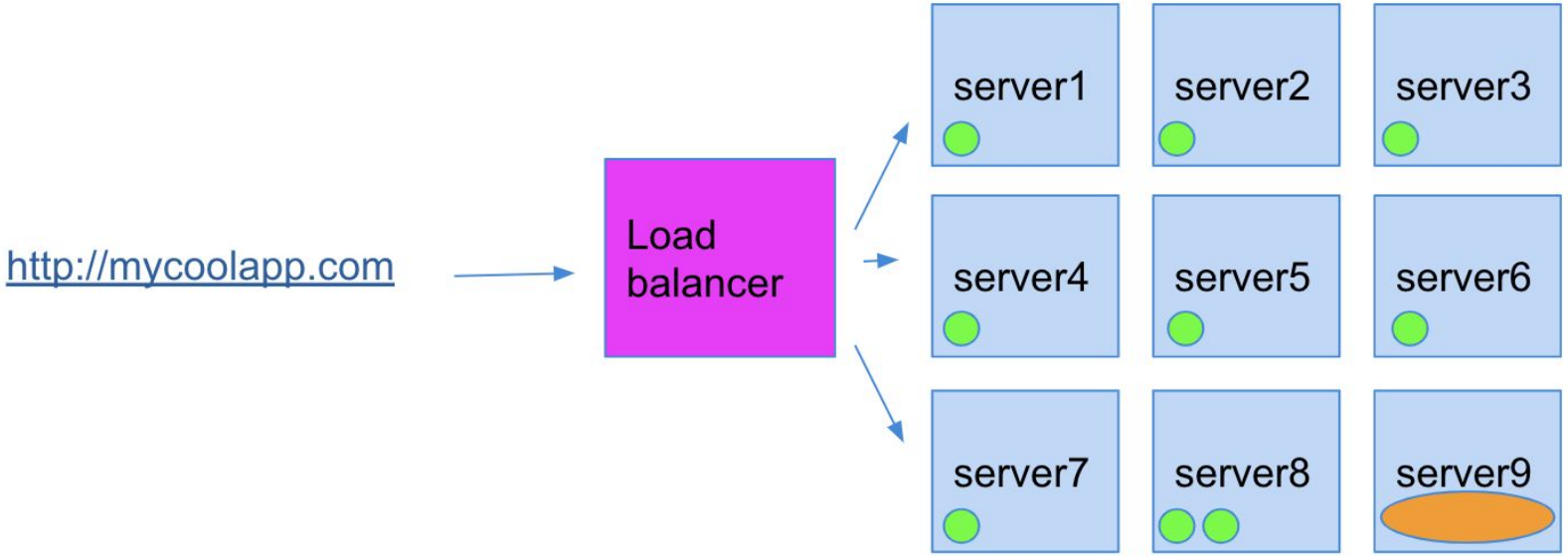
What is the most efficient way to run my containers?

Shipping Containers



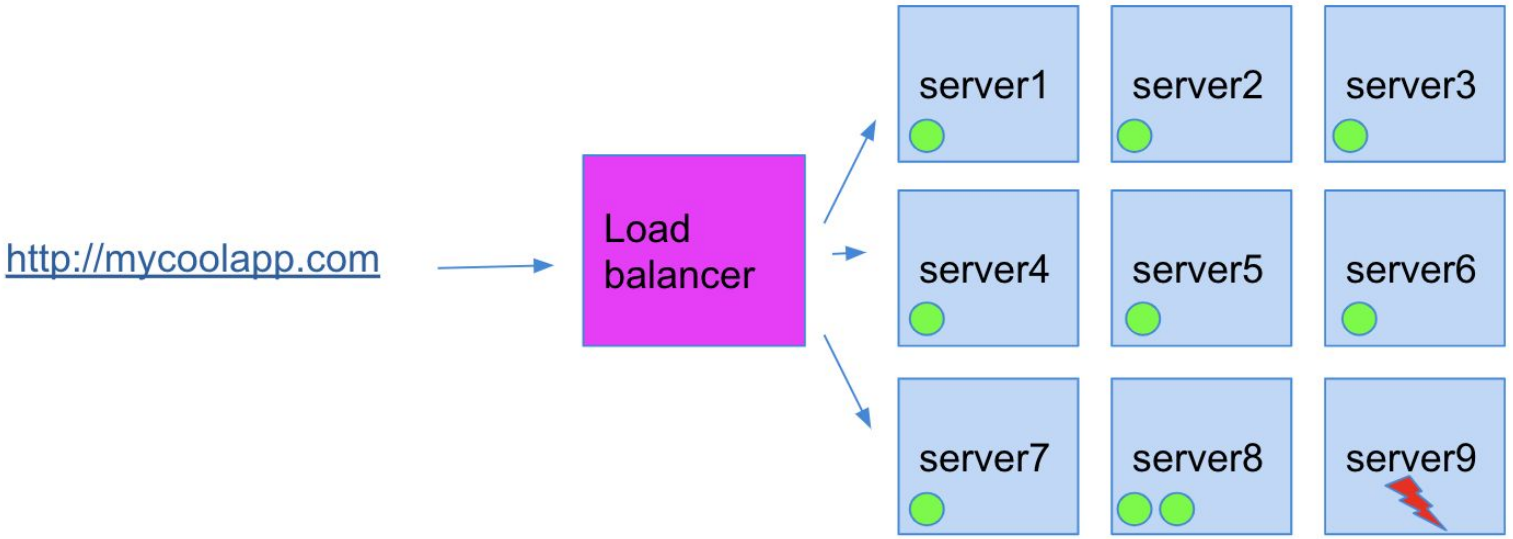
What is the most efficient way to run my containers?

Shipping Containers



What is the most efficient way to run my containers?

Shipping Containers



What is the most efficient way to run my containers?

Serverless

Why use servers at all?

Serverless

The “serverless” movement takes the idea of virtualization even further...

...what if your app is just a small function?

(...or more realistically, a collection of a bunch of small functions)

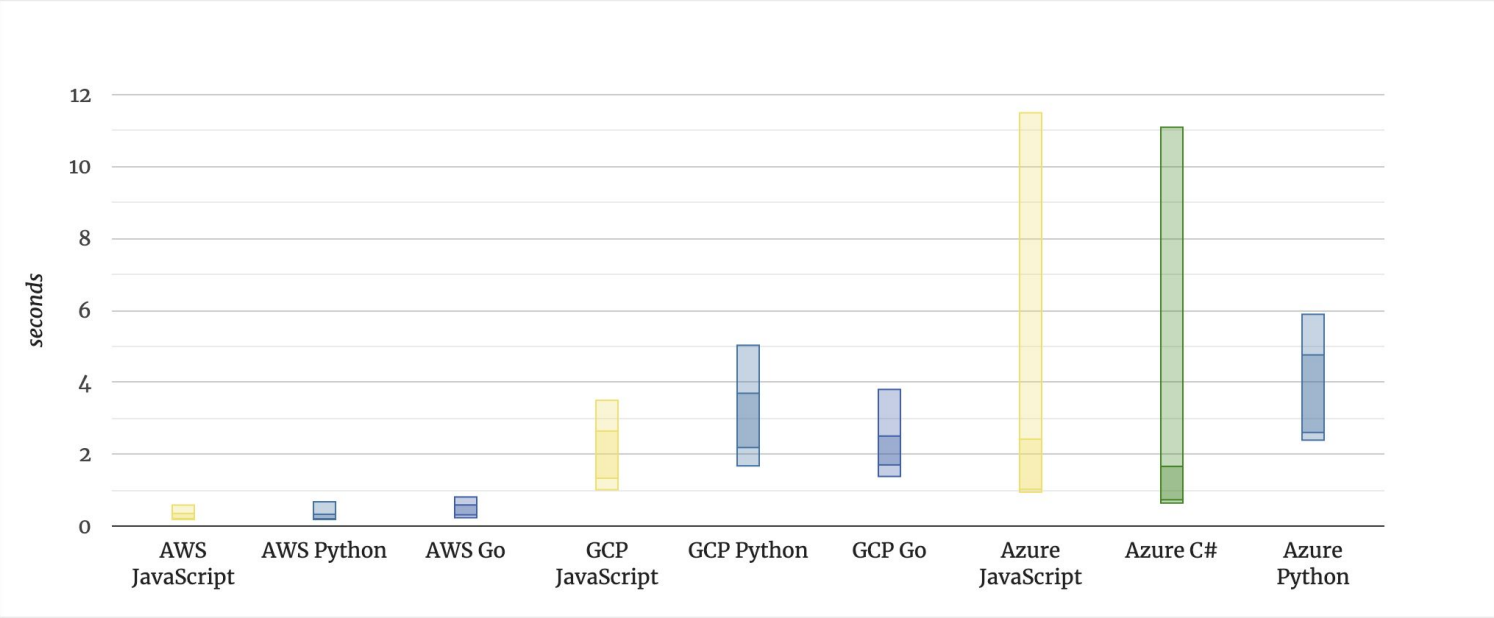
Serverless

The “serverless” movement takes the idea of virtualization even further...

...what if your app is just a small function?

- You configure AWS to receive HTTP requests, forward them to your small function
- AWS allocates a VM to run your function ***only if necessary***
  - If it can use an existing VM, it will
  - If existing VMs are over-utilized, AWS will dynamically allocate a new VM
  - Automatic scaling!
    - You have 0 control over how many VMs are allocated running your functions.
    - If there are 0 VMs currently allocated to run your function, you get a “cold start”

Serverless



Typical cold start durations per language

source

Serverless

Pros:

- Automagic scaling
- Pay for exactly how much load your function handles: $0 if unused!
- Stateless programming model

Cons:

- Potential to pay cold start cost
- Lack of control over scaling can lead to problems
  - How many DB connections should each serverless function get?
- Programming restrictions (15min maximum request duration, no background work, etc)