



# Amazon Web Services

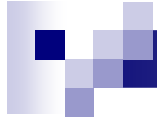
CSE 490H

This presentation incorporates content licensed under the Creative Commons Attribution 2.5 License.

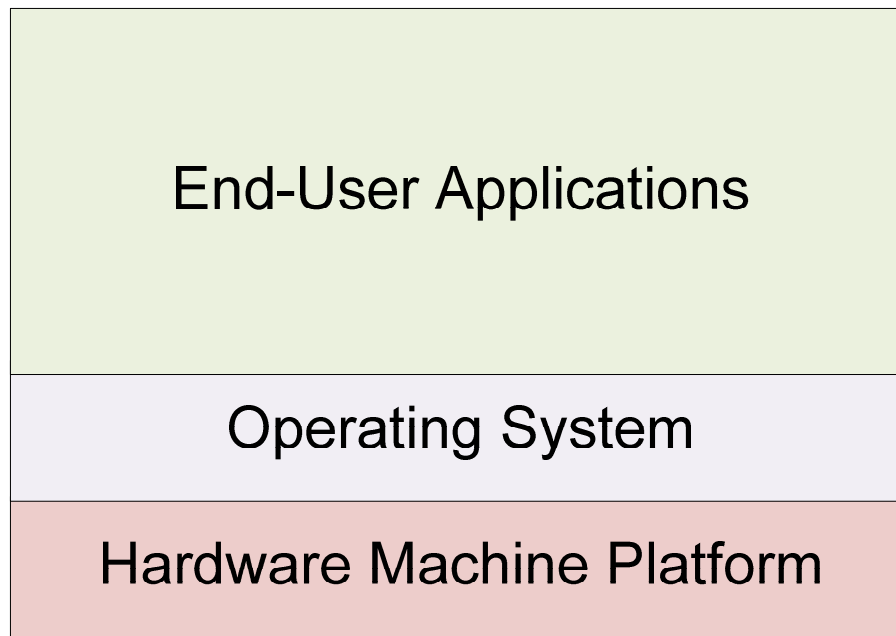


# Overview

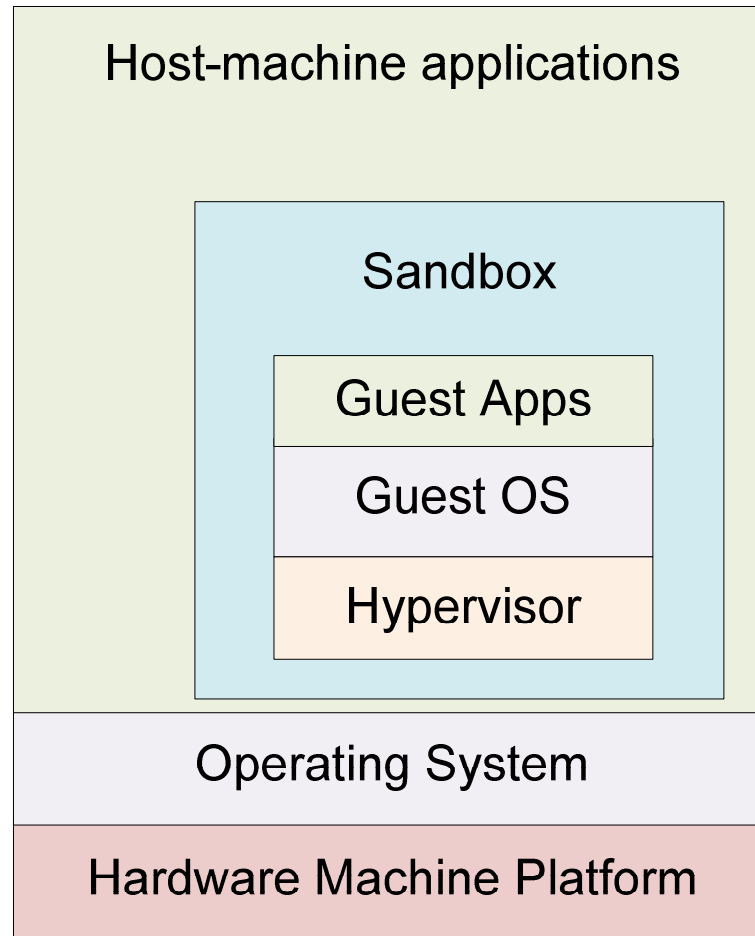
- Questions about Project 3?
- EC2
- S3
- Putting them together

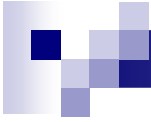


# Brief Virtualization Review

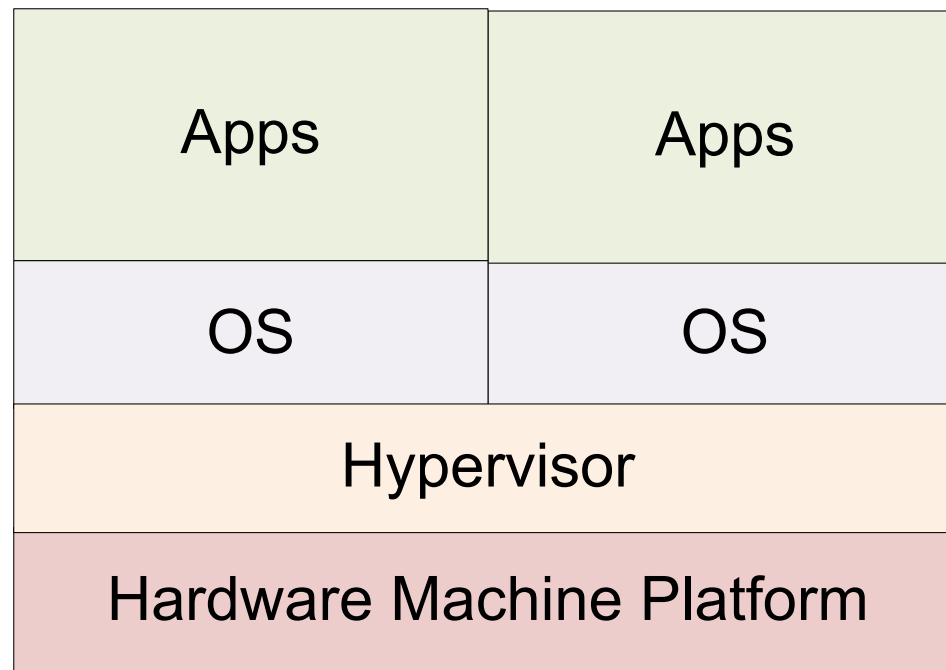


# Host and Guest Systems

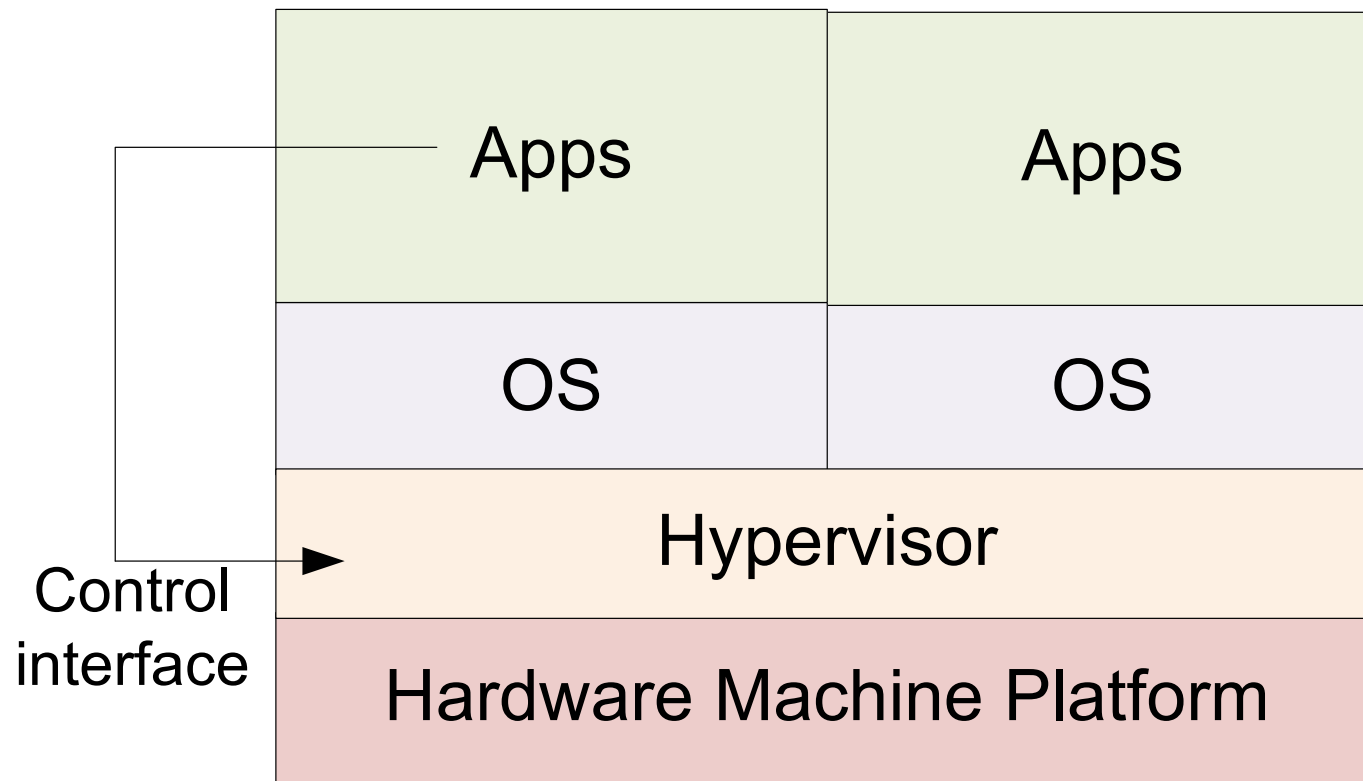




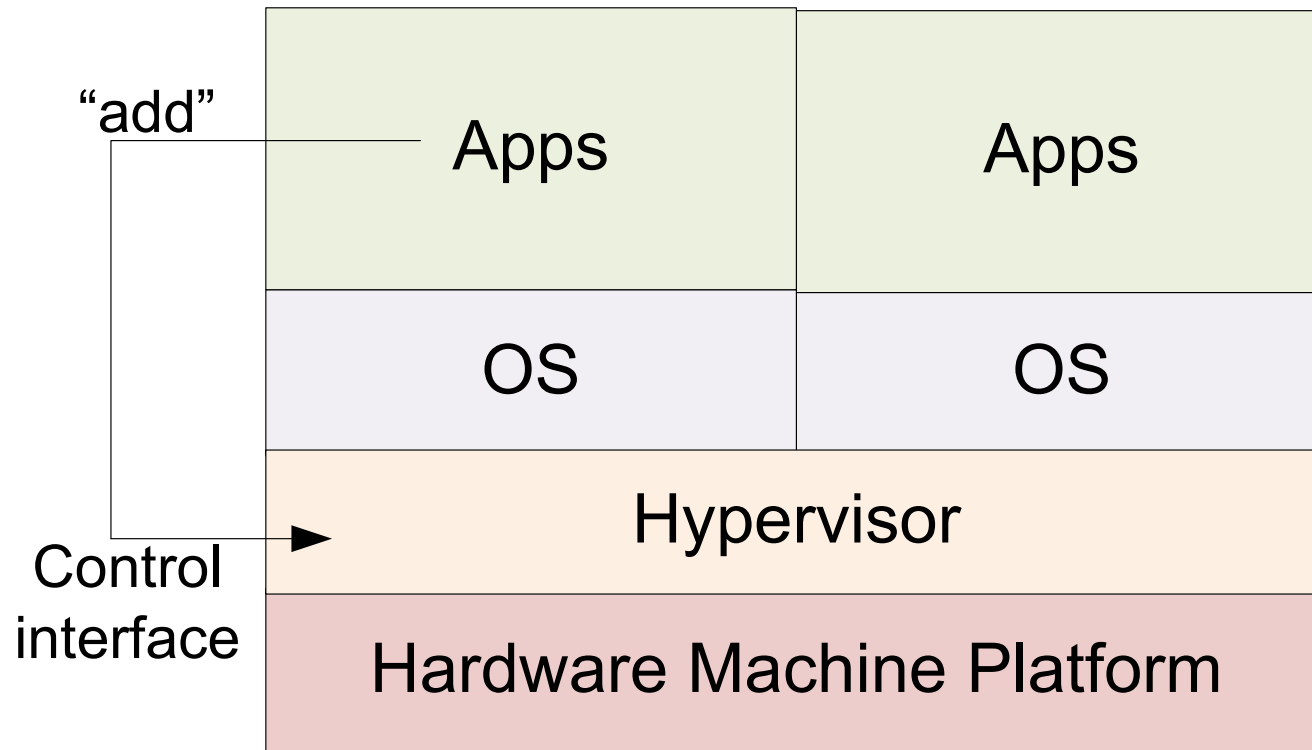
# Fully Virtualized Machine



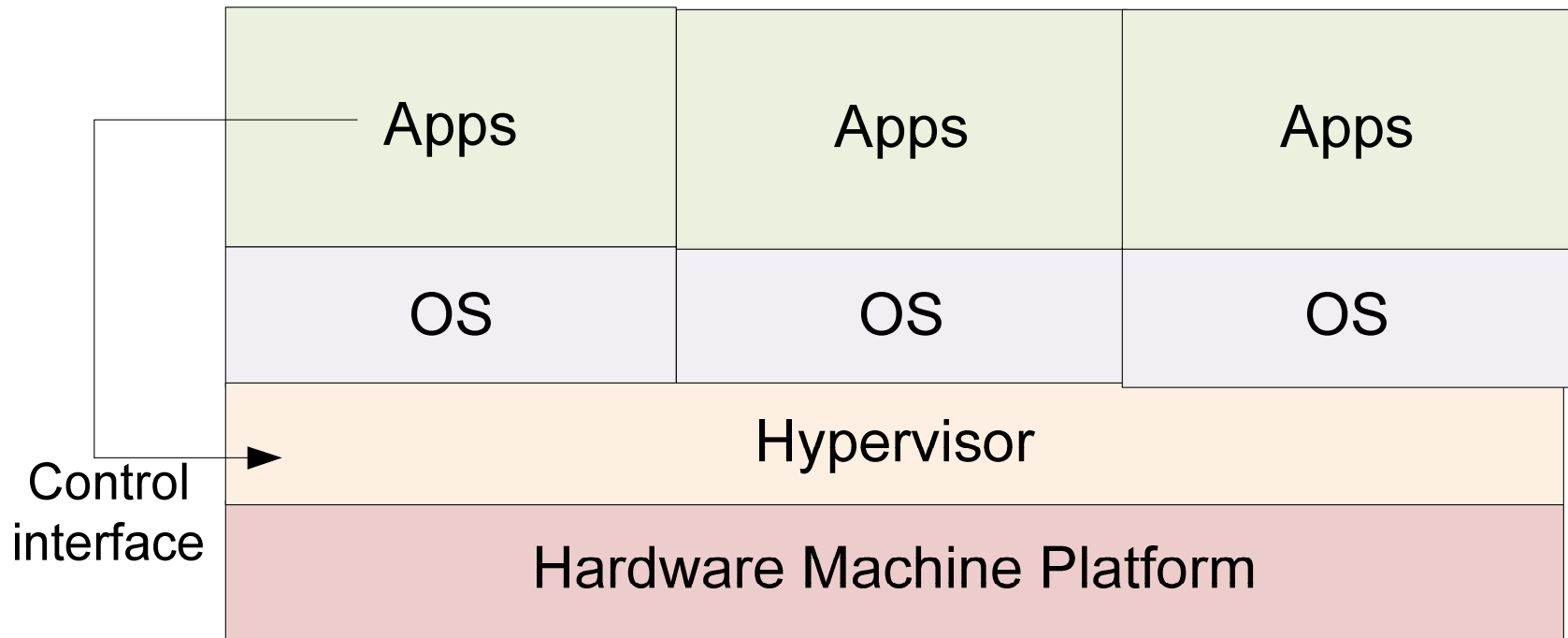
# Interacting with the Hypervisor



# “add machine”

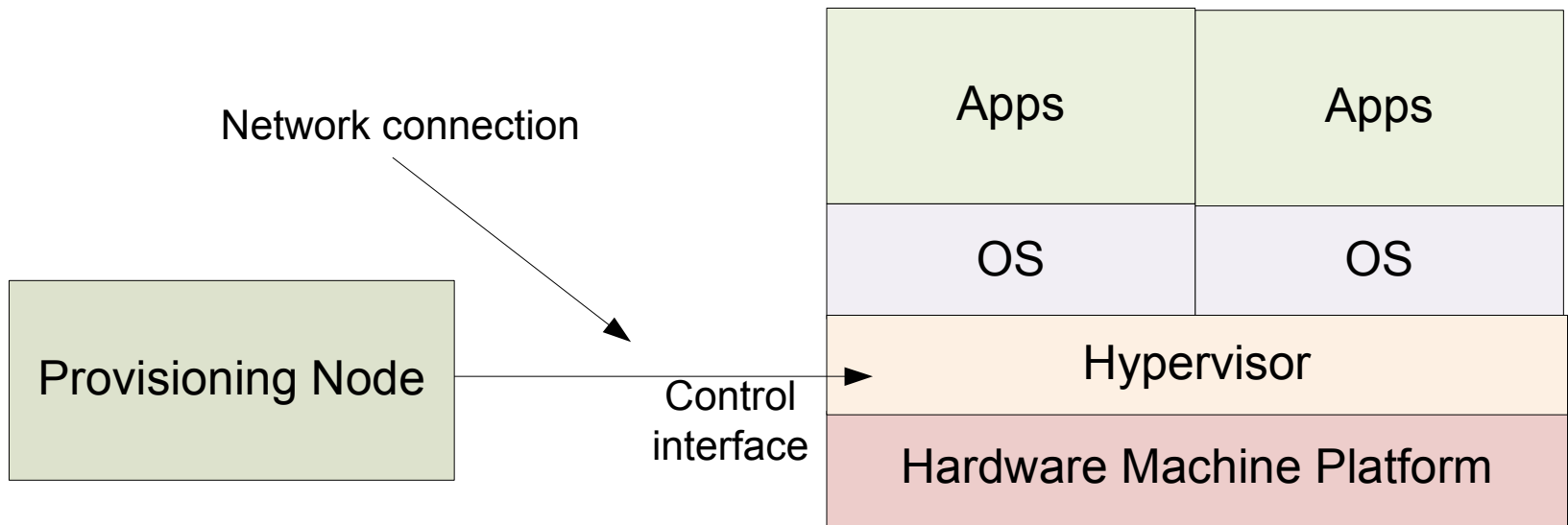


# New machine added





# Managing Large Deployments





# How Web Servers Work

- Interacting with a web servers has three stages
  - *Request* – A URL (and some data) is sent to the server
  - *Handler* – Some logic looks at the request
  - *Response* – Some data is sent back to the user



# Serving a Web Page

- Request: “GET /index.html”
- Handler: The server itself reads the `$wwwroot/index.html` file
- Response: The contents of the file are sent back to the user



# Web Applications

- Request: “GET  
/buyItem.php?itemId=414&customerId=2000”
- Handler: The server invokes the buyItem.php script and runs the code
- Response: Whatever output is sent back from the script gets sent back to the end user’s web browser



# CGI Scripts

- This sort of “Web page that does something” is referred to as CGI (the Common Gateway Interface)
- Typically a script that takes in parameters, does some processing, and returns a new web page to view in your browser



# REST Interfaces

- ... Buy why the focus on “pages?”
- Request: “GET  
/launchMissiles.exe?authCode=12345”
- Handler: launchMissiles program works
- Response: “Boom!”
  
- ...This is a “web service”



# REST Interfaces

- Well-defined “URLs” perform operations
- Web server is connected to programs specific to each of those operations
- Typically work with XML-formatted data
- Designed for connections to be self-contained and non-persistent

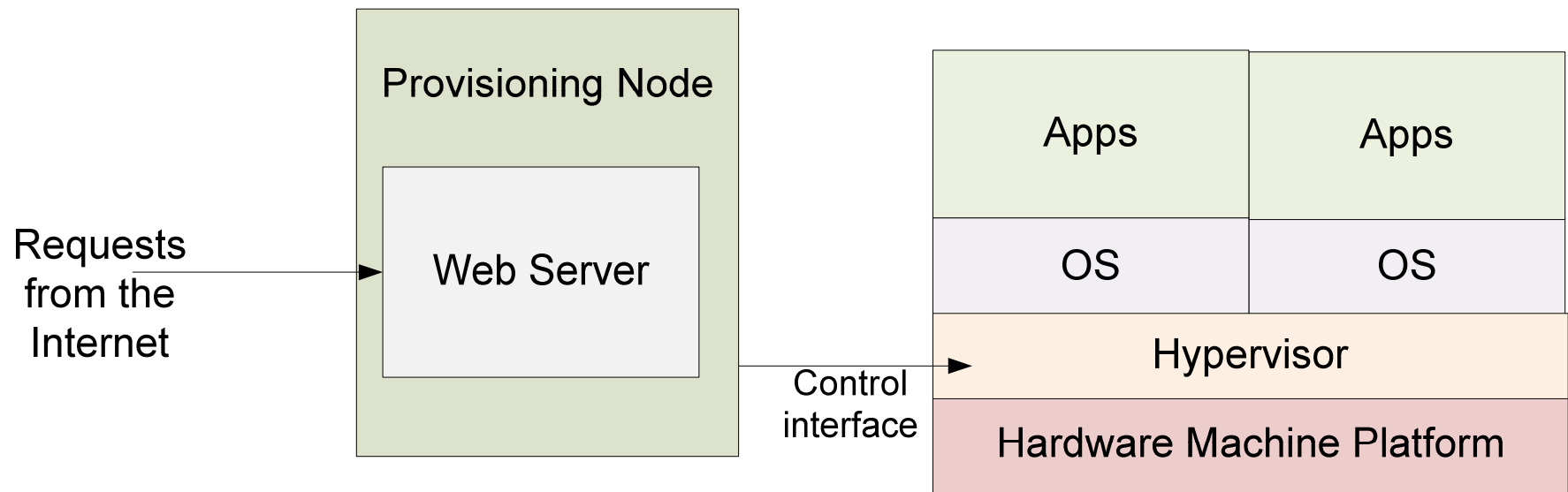


# Web without the Web Browser

- Any application can send/receive data with the HTTP protocol
- Requests can be sent by command-line utilities, other GUI apps, etc
- They then parse the XML response, display data as is appropriate



# Put them together...





# EC2 Terminology

- Instance – A virtual machine
- Image, AMI – The initial state for a VM
- Security Group – A set of instances with shared firewall settings



# Launching Instances

- ec2-run-instances
  - Requires AMI id (e.g., ami-1a2b3c4d)
  - User key, security group, instance type, count
- Doesn't run immediately – instances start in “pending” state; later transition to “running”



# Where's my instance?

- ec2-describe-instances

```
RESERVATION      r-b27edbdb 726089167552  tom
INSTANCE  i-90a413f9  ami-4715f12e
            ec2-67-202-10-48.compute-1.amazonaws.com
            ip-10-251-22-143.ec2.internal
            running tom 0          m1.large
            2008-11-11T17:23:39+0000
            us-east-1c  aki-b51cf9dc  ari-b31cf9da
```



# Firewall rules

- `ec2-describe-group (groupname)`

```
GROUP          726089167552          aaron          aaron
PERMISSION 726089167552 aaron  ALLOWS
      tcp      22      22      FROM  CIDR  0.0.0.0/0
PERMISSION          726089167552          aaron  ALLOWS
      tcp      80      80      FROM  CIDR  0.0.0.0/0
```

- Create a group with `ec2-add-group`
- Control permissions with `ec2-(de)authorize`



# A new instance, a blank slate

- How do you log in to an instance?
- How does an instance know what it should do?
  - Per-instance metadata



# ssh keypairs

- ssh lets you log in to a remote machine with a username
  - Authentication can be done by password
  - Also can be done with public/private keys
- EC2 will let you register a key pair in db
  - Injects public key into instance on boot
  - You have the private key, you can log in



# Shutting down instances

- `ec2-terminate-instance (instance id)`
- Terminates a running instance
- Use `ec2-describe-instances` to get the instance id (`i-XXXXXXXXXX`)





# Using Instance Metadata

- You can create an AMI to do anything you want
- Very specific AMI may already have full application stack already loaded
- More generic AMI may run a bootstrap script
  - Can download more programs, data from another source



# S3 – The Simple Storage Service

- S3 is an infinitely-large, web-accessible storage service
- Data is stored in “buckets” as (key, value) pairs
  - Effectively (server, filename) → file mapping



# S3 has a REST API too

- PUT request to a URL with data uploads the data as the value bound to the key specified by the URL
- GET request to the URL retrieves the value (file) or “404 Not Found”




# S3 Buckets

- Names must be globally unique
  - (Since they are addressable as DNS entries)
- Can hold an unlimited number of keys
- Each key can have up to 5 GB of value



# Starting a Server

- ec2-run-instances can specify metadata
- A new server is provisioned and boots
- Boot process runs a script that reads metadata
  - This specifies location of another program
  - Retrieves the program, runs it
  - Retrieves data, starts more services, etc...



# Project 4 And You

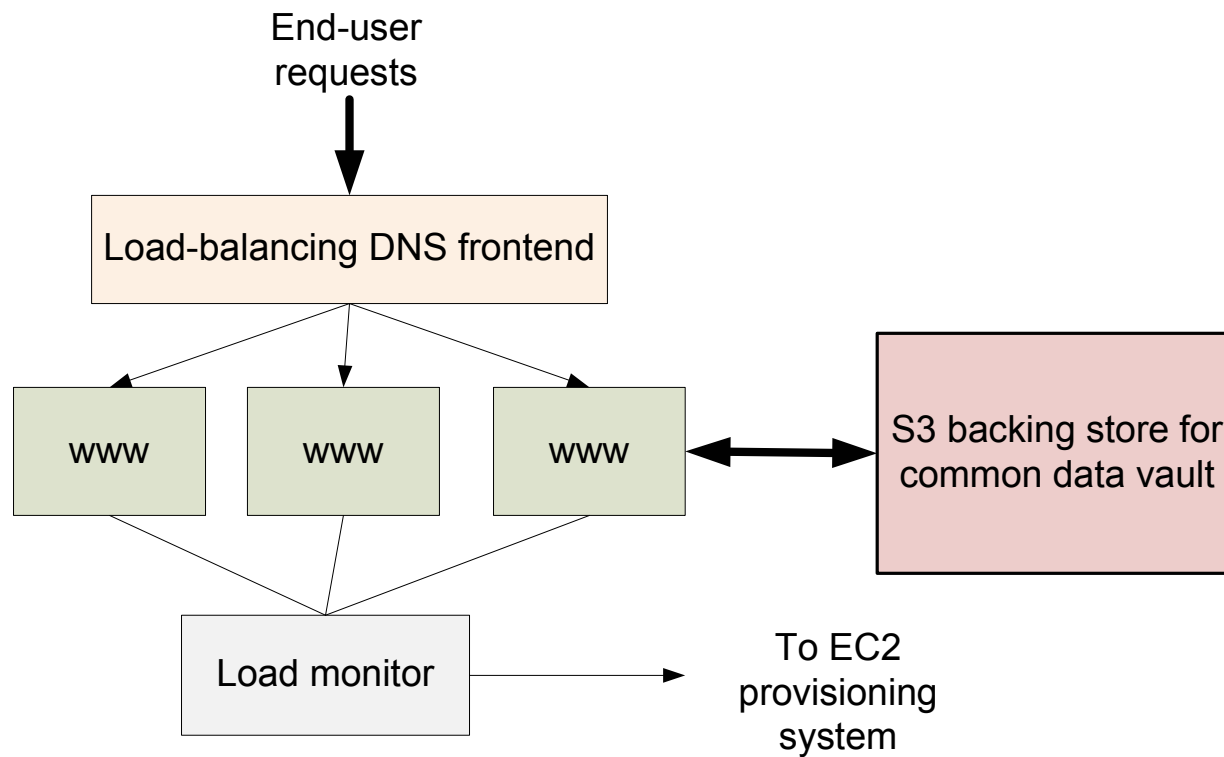
- Project 3 will provide you with map tiles and an index from (address → lat, lon)
- In project 4, you will:
  - Upload this into S3
  - Write a web server handler applet to do address lookups
  - Write the bootstrap scripts to retrieve data from S3 into your instance and launch your server



# More Web Services

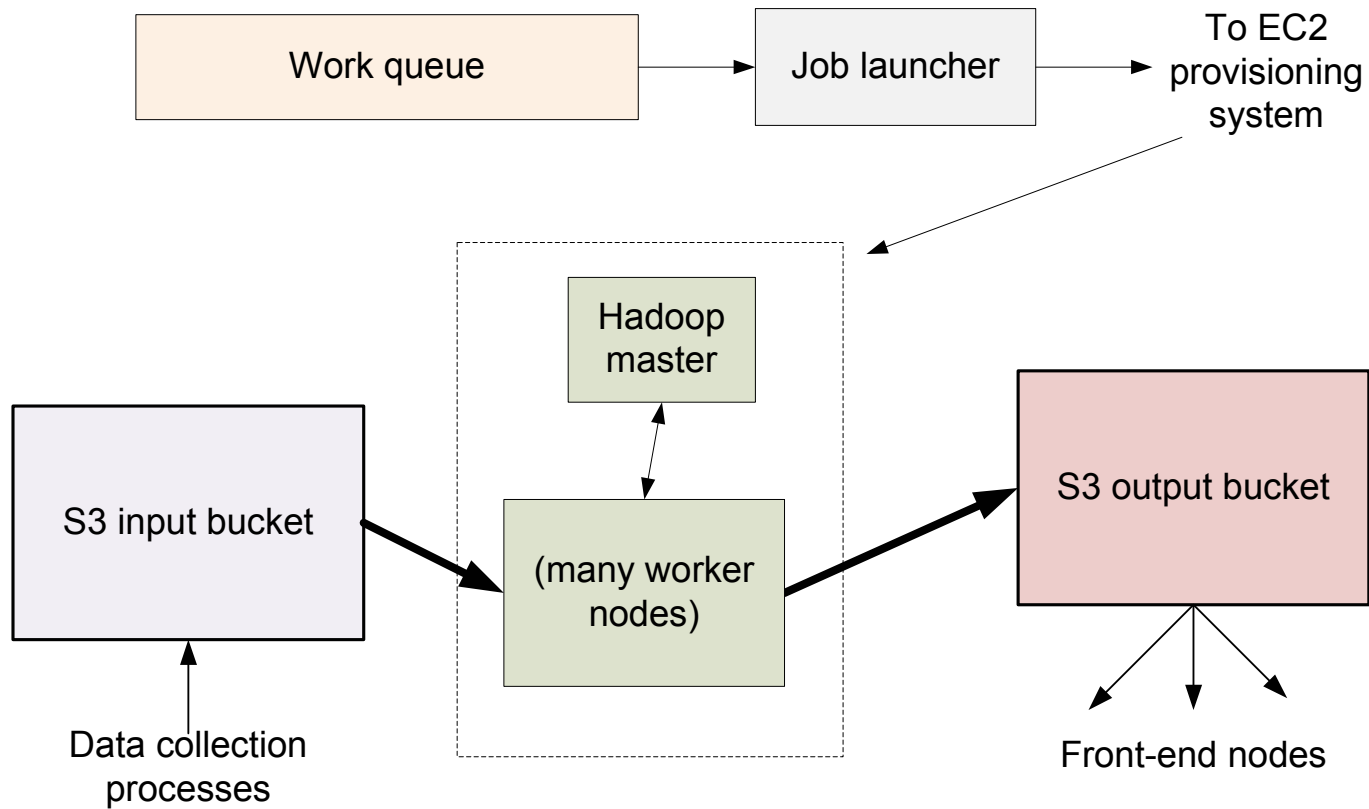
- Simple Queue Service (SQS)
  - Reliable producer—consumer queues that hold millions of queue entries, with hundreds of servers connecting...
- Simple Database Service (SDB)
  - A lot like BigTable

# Self-Scaling Applications





# Self-Scaling Backends





# GrepTheWeb

- Large web crawl data is stored in S3
- Users can submit regular expression to the GTW program
  - GTW uses Hadoop to search for data
  - Puts your results in an output bucket and notifies you when it's ready

Figure 3: Phases of GrepTheWeb Architecture

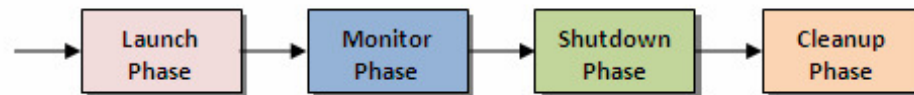
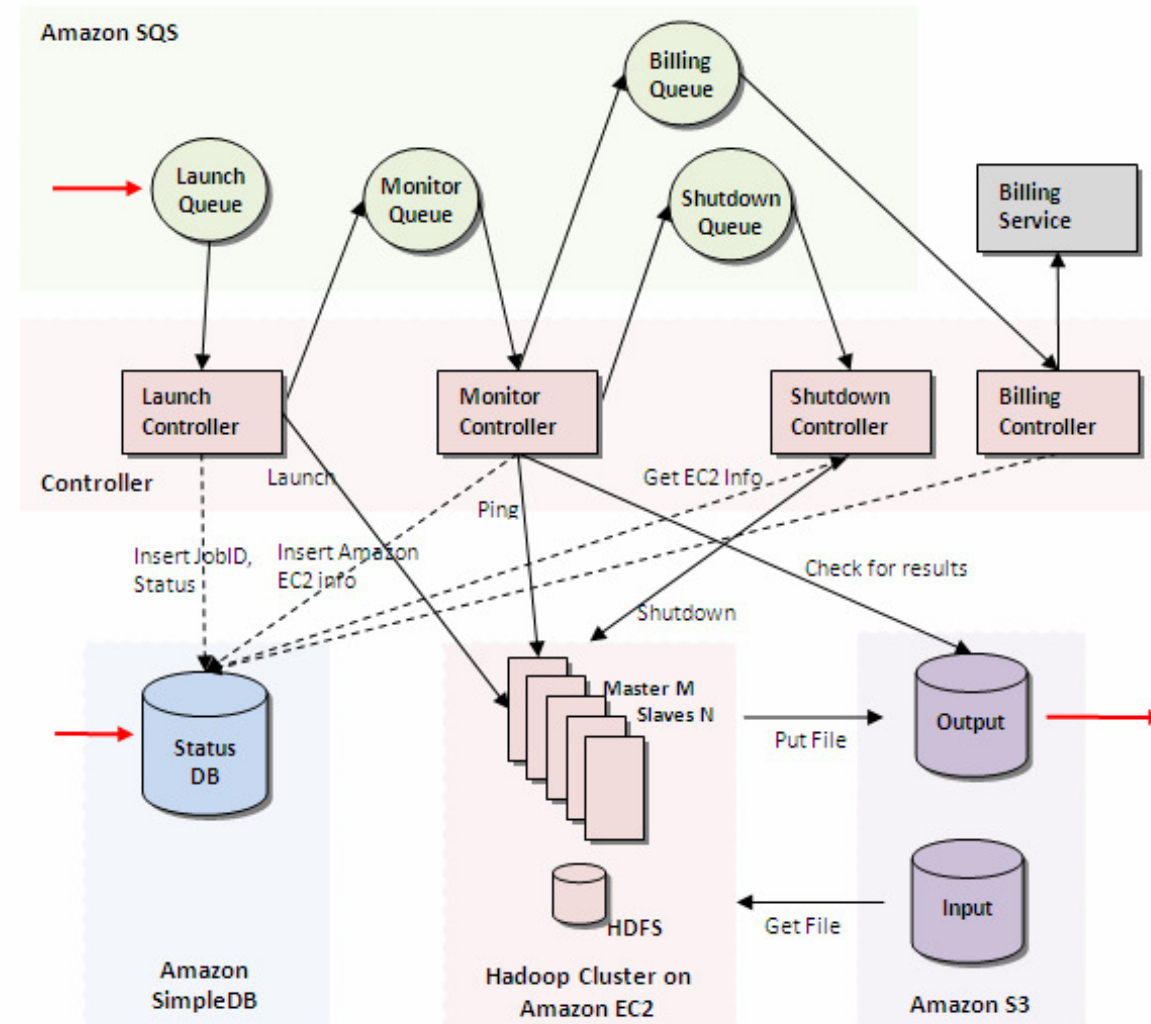


Figure 4: GrepTheWeb Architecture - Zoom Level 3





# Conclusions

- Web Services make for clean couplings between systems
- Hardware as a Service (EC2/S3) allows applications to use physical resources dynamically
- The two put together allow for very scalable application design