Cloud Computing
CS9243 Guest Lecture

Hiroshi Wada, PhD

Jorke Odolphi

Software Systems Research Group, NICTA

Yuruware

Agenda

- Introduction to cloud
  - Why Cloud Computing?
  - Look into Amazon Web Service and Google App Engine
- "Real" Cloud Architecture
- Cloud Architecture and NoSQL
- Application Architecture on Cloud
  - Scaling and fault tolerance

Operating infrastructure is not easy

NICTA

- Large upfront and running cost
  - Facility, hardware and configuration
  - 24h monitoring, backup, hardware maintenance
- Capacity planning is not easy
  - You cannot change infrastructure quickly
    - Not easy to change the facility design
    - Often takes days or weeks to obtain new hardware
  - Predict the peak load and provision based on it
    - e.g., 14:00 on Friday, Christmas, tax returns, …

http://www.flickr.com/photos/neospire/3594831687/

Over and under provisioning

- "overly" over provision is not uncommon due to the fear of under provisioning and lack of flexibility
- Issues
  - Waste of resources
    - In large data centers, average usage of servers is 5% – 15%
  - Cannot decommission easily to follow the workload
  - Cannot handle sudden workload spikes

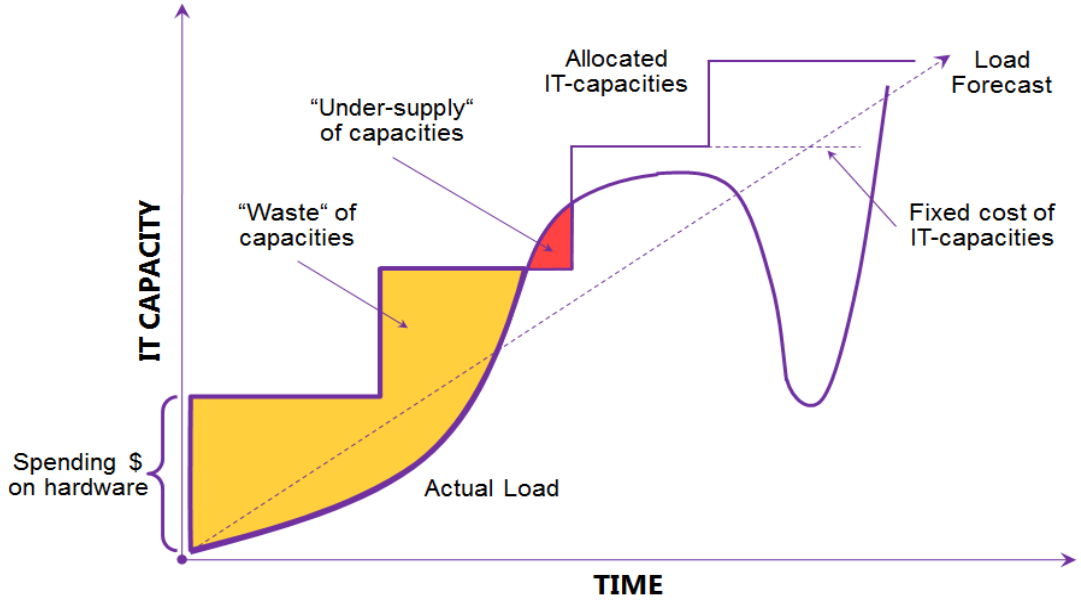
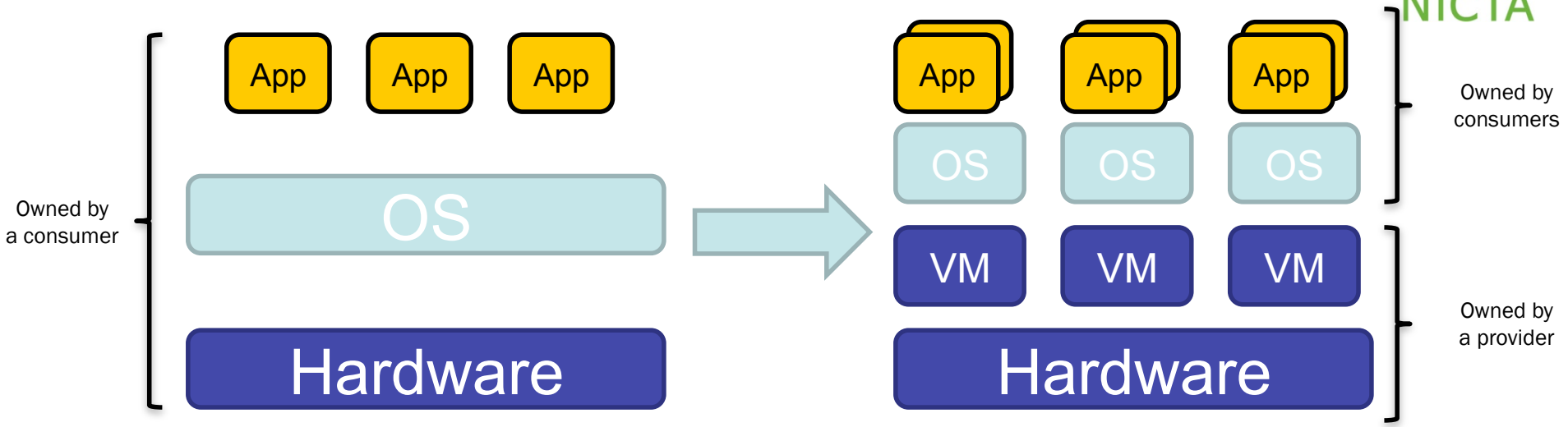


Fig1 in "Developing and Extending Applications for Windows Azure with Visual Studio"

Cloud Computing



- Basic idea is to share computing resources among multiple users with proper isolation
  - e.g., Split a physical server into multiple virtual machines
- Providers can reduce the cost per resource unit
  - Bulk purchase lowers IT equipment cost
  - Managing 1000 servers costs same as managing 20 servers
- Consumers can buy resources only when needed

Why Cloud Computing

- High Elasticity/Scalability
  - Virtually infinite amount of resources is available on demand
  - Cost of 1 machine for 100 hours = Cost of 100 machines for 1 hour
- (Possibly) Reduce cost and complexity
  - Pay per use and no up-font cost
  - No in-house data center management
- Ease-of-use
  - Obtain a server in 5 min, not days, and throw away in 10 sec
  - Various on-demand services (not only virtual machines)

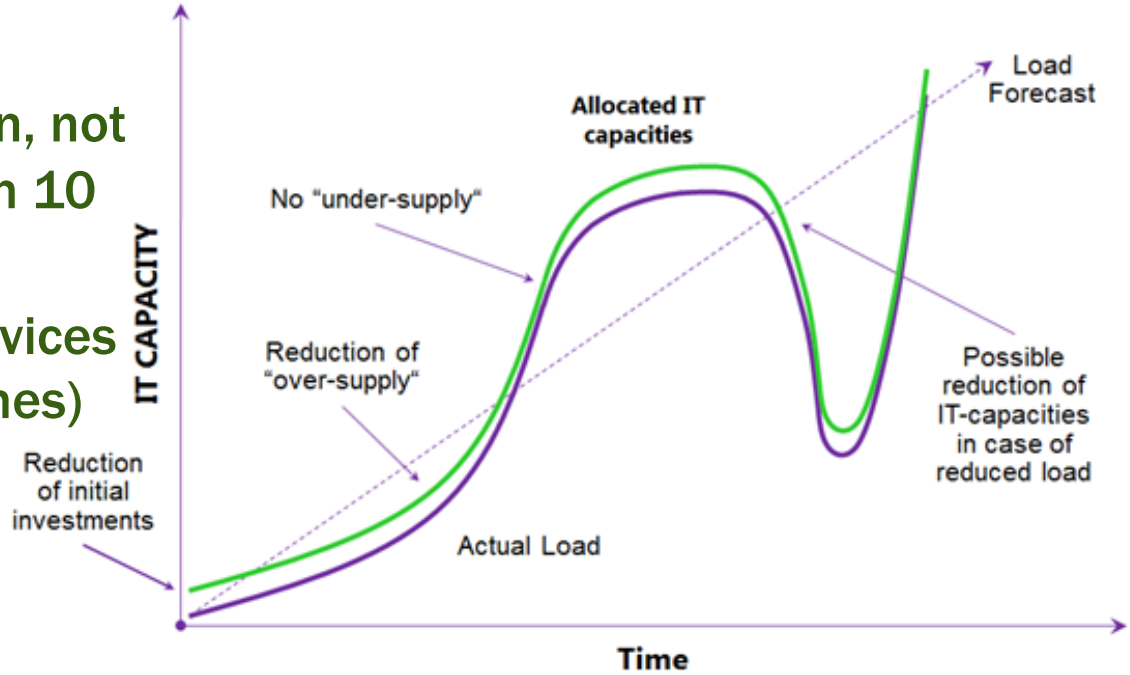


Fig2 in "Developing and Extending Applications for Windows Azure with Visual Studio"

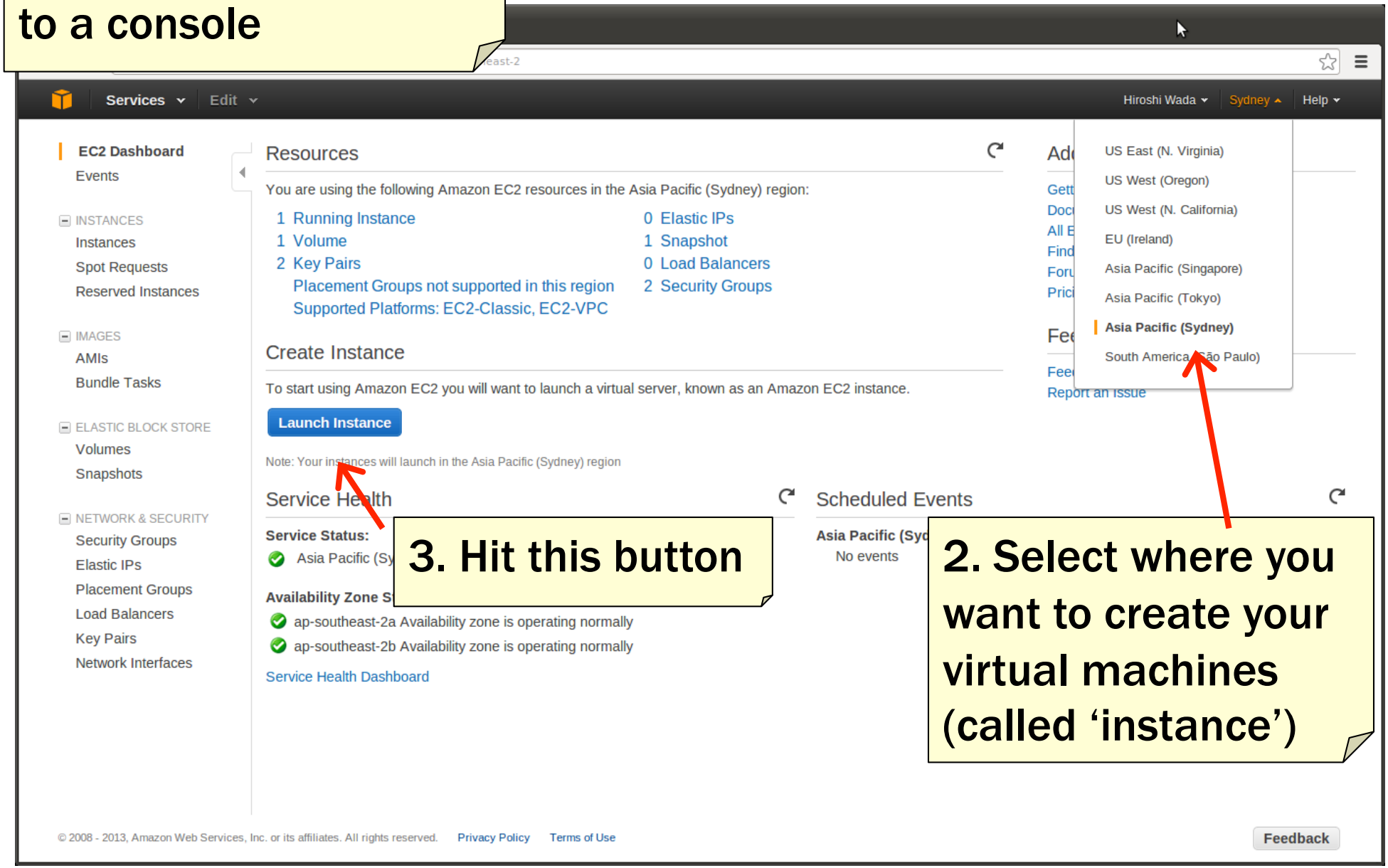
Leading Provider: Amazon EC2

NICTA

Let's see how Amazon EC2, a leading commercial cloud, looks

1. Grab your credit card and create an account. (10 min) Then, access to a console

NICTA

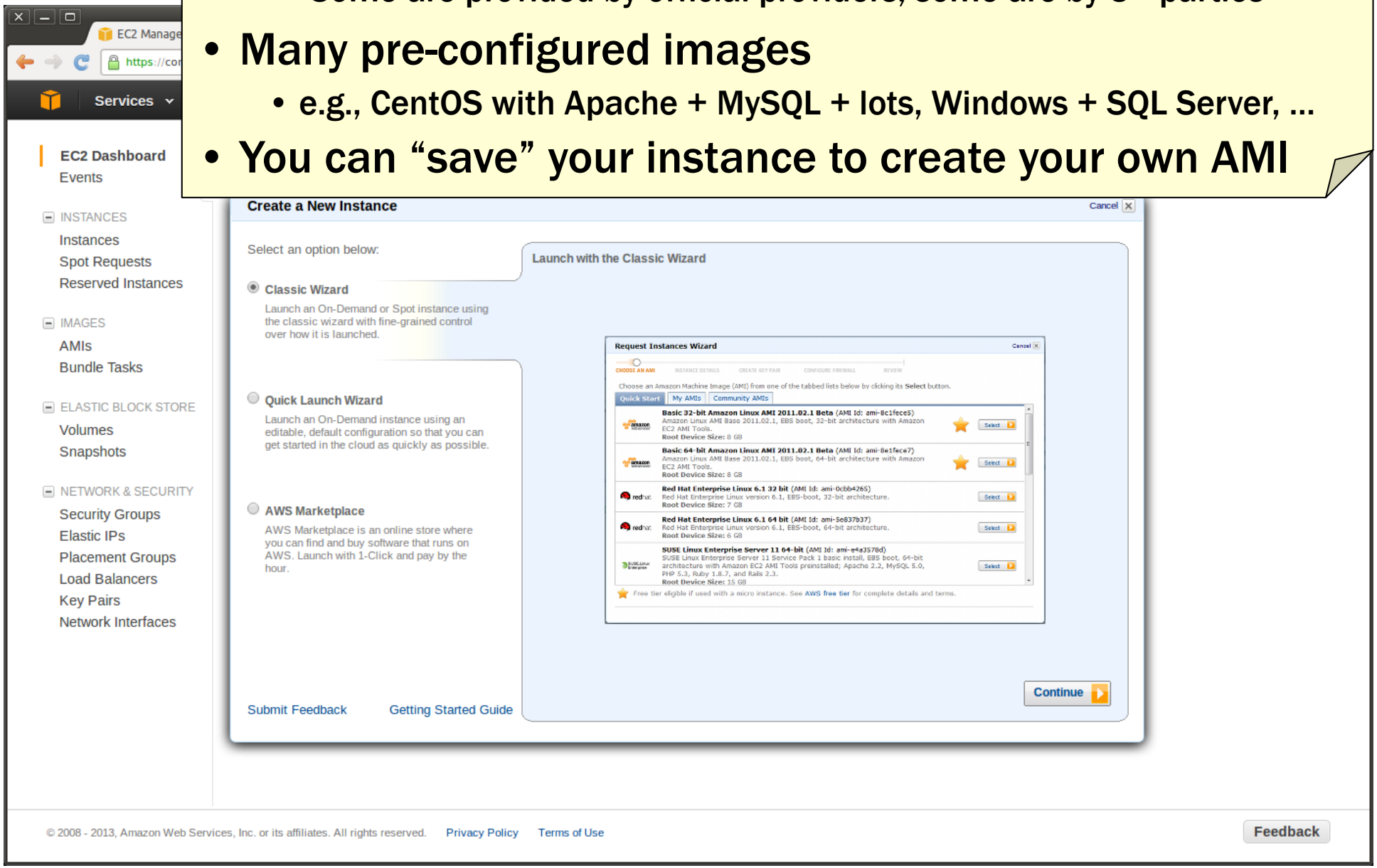


3. Hit this button

2. Select where you want to create your virtual machines (called ‘instance’)

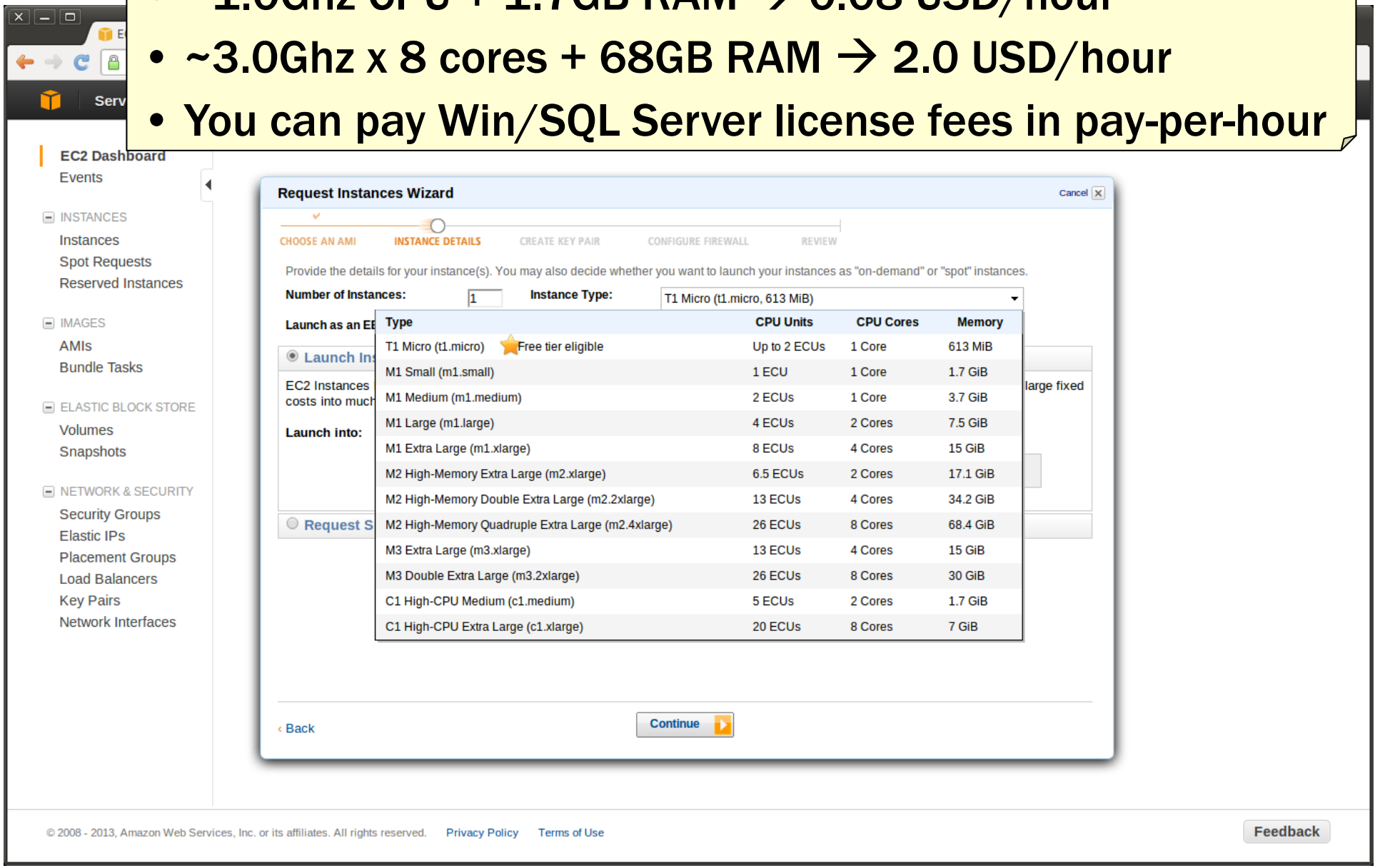
4. Select a machine image (called AMI) to use

- Various OS (Major distros, Redhat, Windows, …)
  - Some are provided by official providers, some are by 3rd parties
- Many pre-configured images
  - e.g., CentOS with Apache + MySQL + lots, Windows + SQL Server, …
- You can “save” your instance to create your own AMI



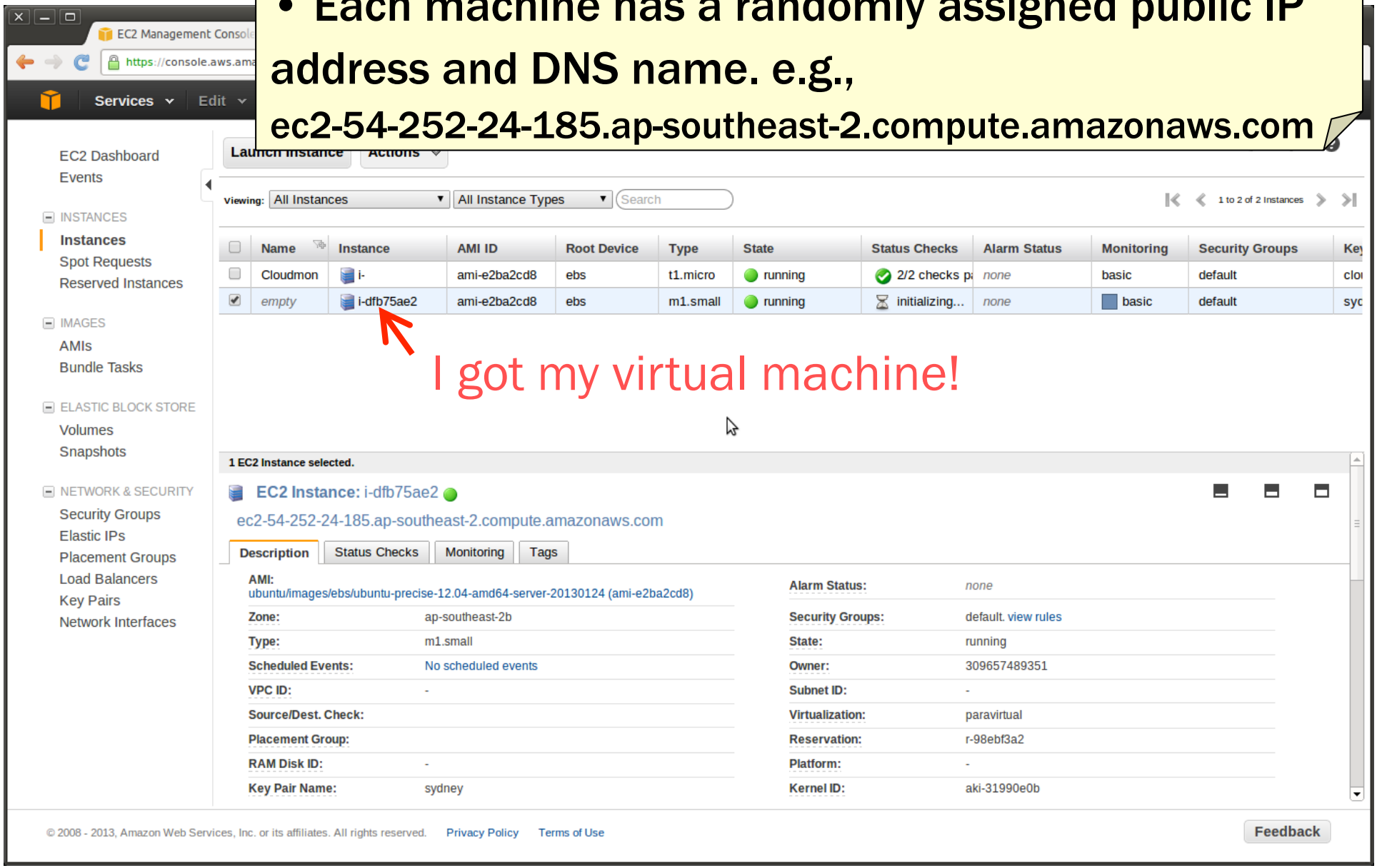
5. Determine the amount of resources to allocate

- <1.0Ghz CPU + 600MB RAM → 0.02 USD/hour
- ~1.0Ghz CPU + 1.7GB RAM → 0.08 USD/hour
- ~3.0Ghz x 8 cores + 68GB RAM → 2.0 USD/hour
- You can pay Win/SQL Server license fees in pay-per-hour

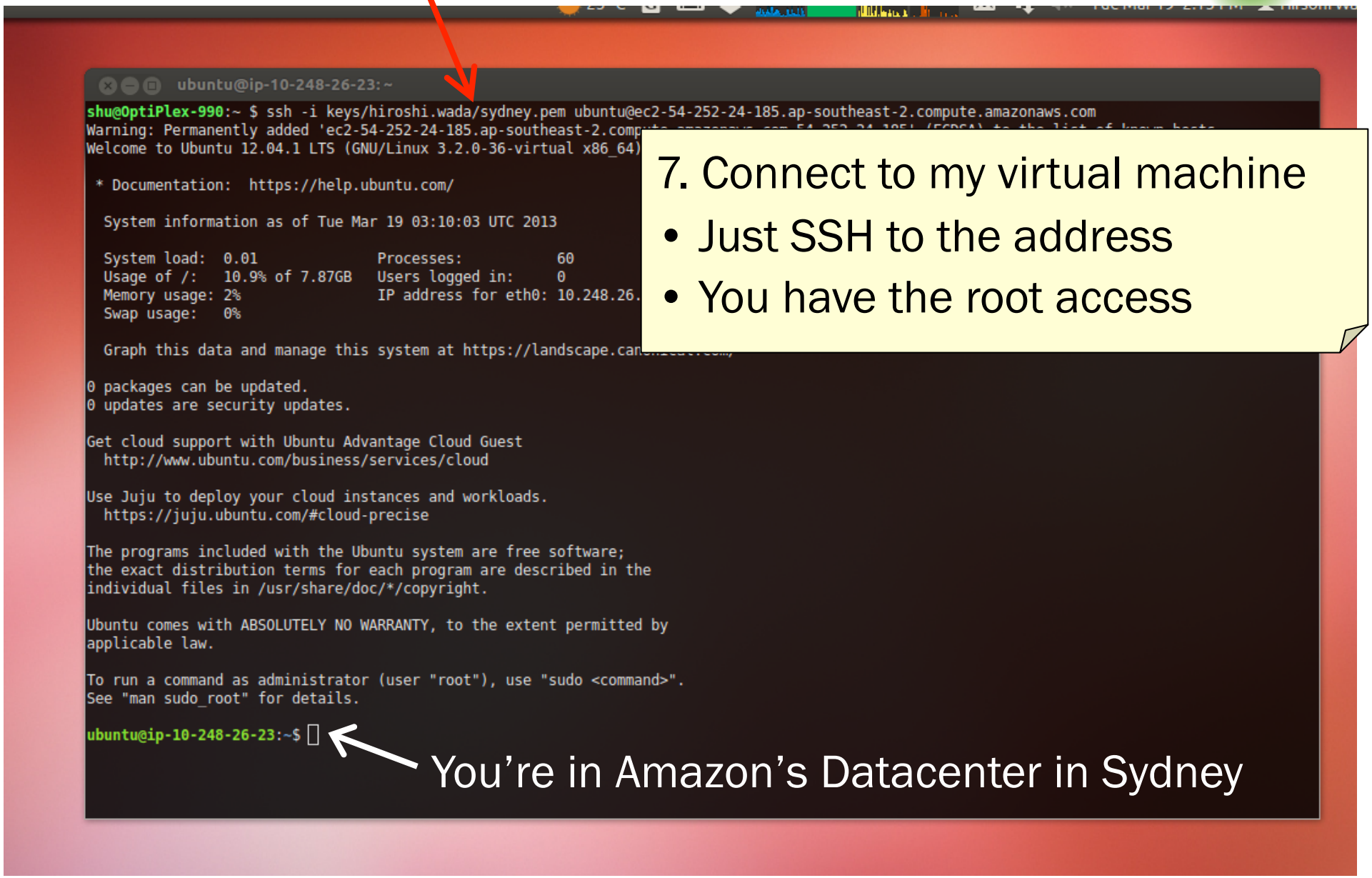


6. Done! (< 5 minutes in total)

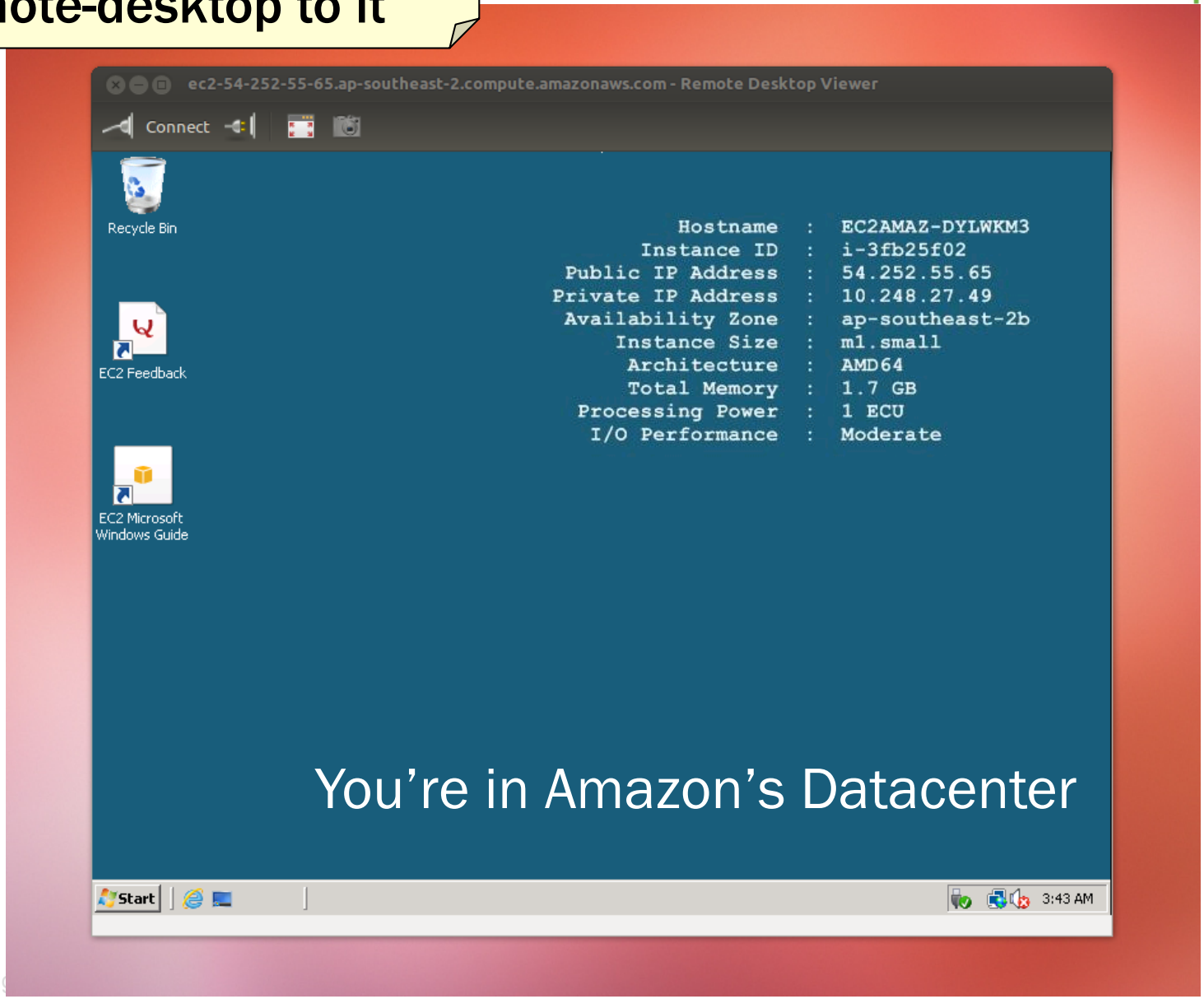
- Set SSH key, configure firewall, etc.
- Each machine has a randomly assigned public IP address and DNS name. e.g., ec2-54-252-24-185.ap-southeast-2.compute.amazonaws.com



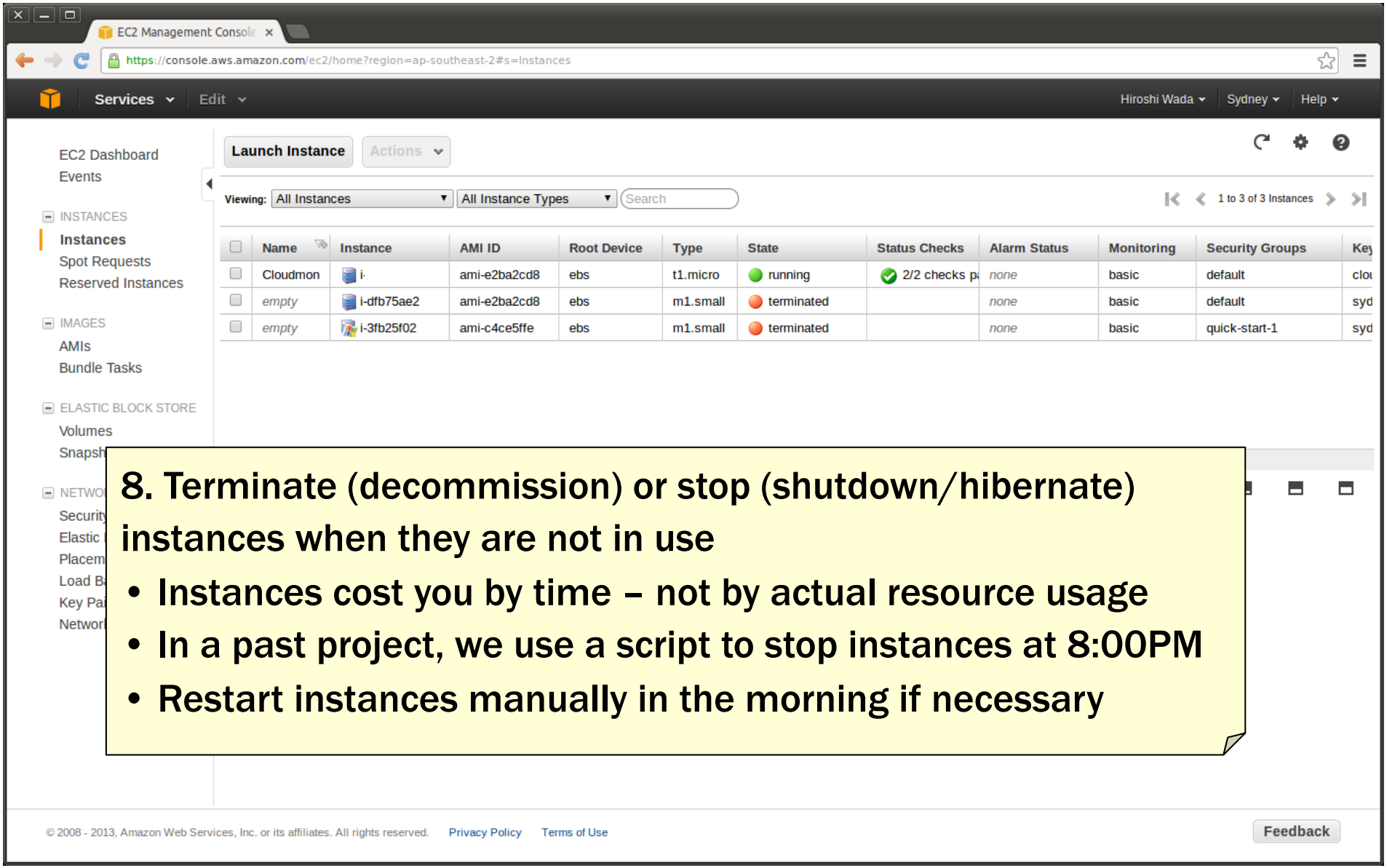
SSH from a desktop



If you need Windows, launch a Windows instance and remote-desktop to it

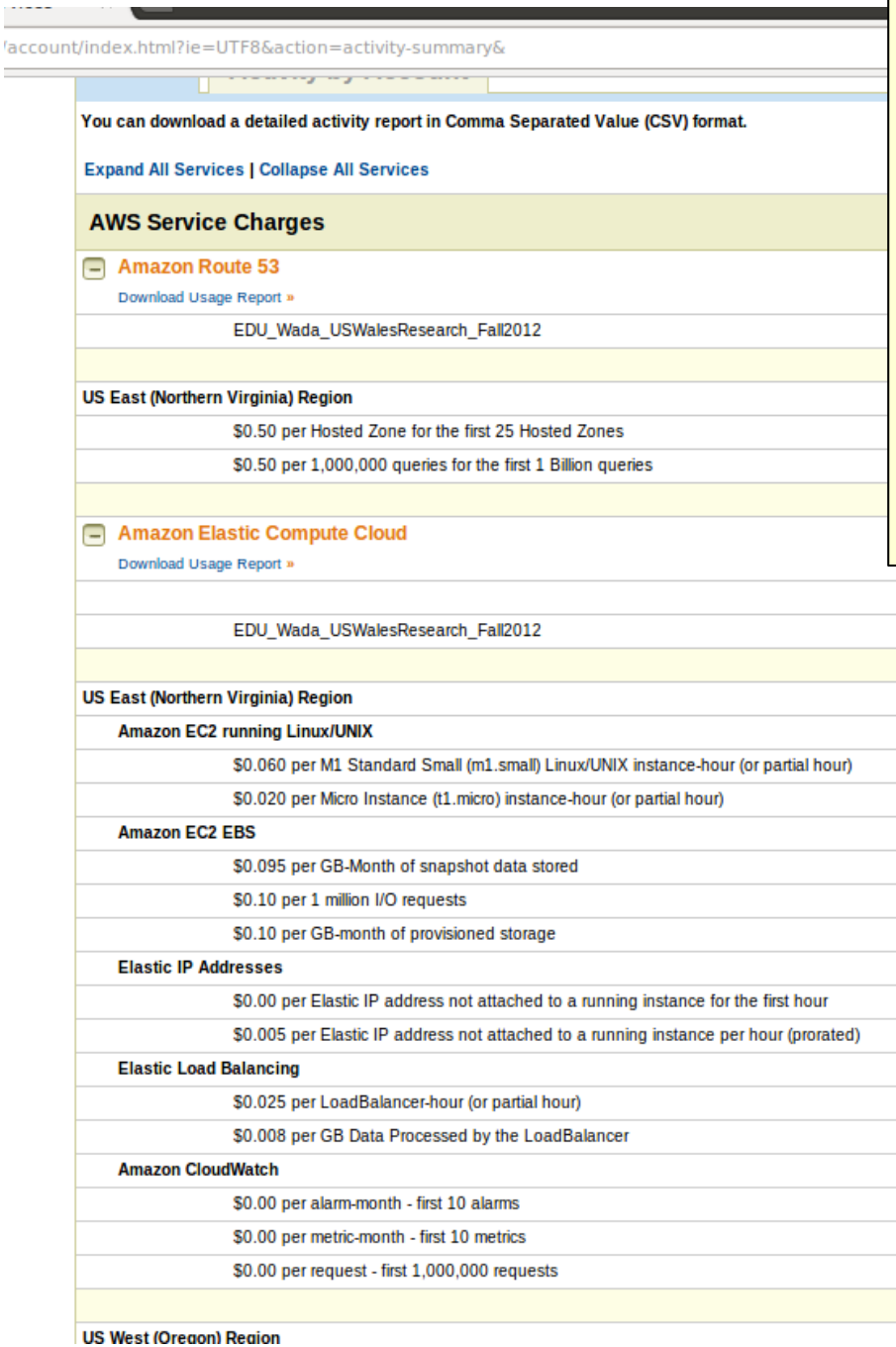


NICTA



8. Terminate (decommission) or stop (shutdown/hibernate) instances when they are not in use

- Instances cost you by time – not by actual resource usage
- In a past project, we use a script to stop instances at 8:00PM
- Restart instances manually in the morning if necessary



account/index.html?ie=UTF8&action=activity-summary&

You can download a detailed activity report in Comma Separated Value (CSV) format.

Expand All Services | Collapse All Services

AWS Service Charges

Amazon Route 53

Download Usage Report »

EDU_Wada_USWalesResearch_Fall2012		
US East (Northern Virginia) Region		
$0.50 per Hosted Zone for the first 25 Hosted Zones		
$0.50 per 1,000,000 queries for the first 1 Billion queries		

Amazon Elastic Compute Cloud

Download Usage Report »

EDU_Wada_USWalesResearch_Fall2012	Credit	-142.21
		-142.21
US East (Northern Virginia) Region		
Amazon EC2 running Linux/UNIX		
$0.060 per M1 Standard Small (m1.small) Linux/UNIX instance-hour (or partial hour)	879 Hrs	52.74
$0.020 per Micro Instance (t1.micro) instance-hour (or partial hour)	248 Hrs	4.96
Amazon EC2 EBS		
$0.095 per GB-Month of snapshot data stored	0.856 GB-Mo	0.08
$0.10 per 1 million I/O requests	3,245,306 IOs	0.32
$0.10 per GB-month of provisioned storage	35.480 GB-Mo	3.55
Elastic IP Addresses		
$0.00 per Elastic IP address not attached to a running instance for the first hour	1 Hr	0.00
$0.005 per Elastic IP address not attached to a running instance per hour (prorated)	485 Hrs	2.43
Elastic Load Balancing		
$0.025 per LoadBalancer-hour (or partial hour)	524 Hrs	13.10
$0.008 per GB Data Processed by the LoadBalancer	0.004 GB	0.01
Amazon CloudWatch		
$0.00 per alarm-month - first 10 alarms	1.494 Alarms	0.00
$0.00 per metric-month - first 10 metrics	4.961 Metrics	0.00
$0.00 per request - first 1,000,000 requests	21,709 Requests	0.00
		77.19
US West (Oregon) Region		

9. Check the cost in near real-time

- Hours to run virtual machines
- Network in/out
- VPN
- Disk access
- # of requests made to services

...

Programmable Infrastructure


```
https://ec2.amazonaws.com/?Action=RunInstances
&ImageId=ami-xxxxxxx&InstanceType=m1.large
&MaxCount=1&MinCount=1
&KeyName=web&AUTHPARAMS
```

- All actions have REST APIs
- What you can do on the web interface can be done through REST as well
- Use of APIs opens the possibility of flexible operations
  - Obtain resources on demand
  - Release unused resources
  - Automate the environment building
  - Micro-rebooting of apps for error handling
  - … will discuss later

```
$ ec2-run-instances ami-xxxxx –g default –k web
$ ec2-run-terminate i-xxxxx
```

```
try {
  RunInstancesRequest req = new RunInstancesRequest();
  req.setImageId("ami-xxxx");
  req.setPlacement(new Placement().withAvailabilityZone(az));
  req.setSecurityGroupIds(securityGroupIds);
  RunInstancesResult res = ec2client.runInstances(req);
} catch ()…
```

- **SDKs are available for major programming languages**
  - iOS, Java, PHP, Ruby, …
  - It's possible to write code with command line tools and sh
  - You can use curl to send REST messages ☺

Five Characteristics – NIST Definition

- On-demand Self-Service
  - A consumer can provision computing capabilities without human interaction
- Broad network access
  - Computing capabilities are available over the network and accessed through standard mechanisms
- Resource pooling
  - Provider's computing resources are pooled to serve multiple consumers with different resources dynamically assigned according to consumers' demands
- Rapid elasticity
  - Computing capabilities can be rapidly and elastically provisioned to quickly scale out and rapidly released to scale in
- Measured service
  - Resource usage can be monitored, controlled, and reported. Providing transparency for both the provider and consumer

Three Service Models – NIST definition

NICTA

Technology exposed to customers

Commercial providers

Software as a Service

salesforce.com
Success On Demand.

Platform as a Service

PostgreSQL

Microsoft ASP.net

MySQL

GRAILS

mongoDB

Google App Engine

Windows Azure

heroku

CLOUD FOUNDRY

Infrastructure as a Service

vmware

Windows Server Hyper-V

openstack

Xen

amazon web services

GOGRID

Google Compute Engine

Windows Azure

rackspace

Datacenter Infrastructure

EQUINIX

GOGRID

at&t

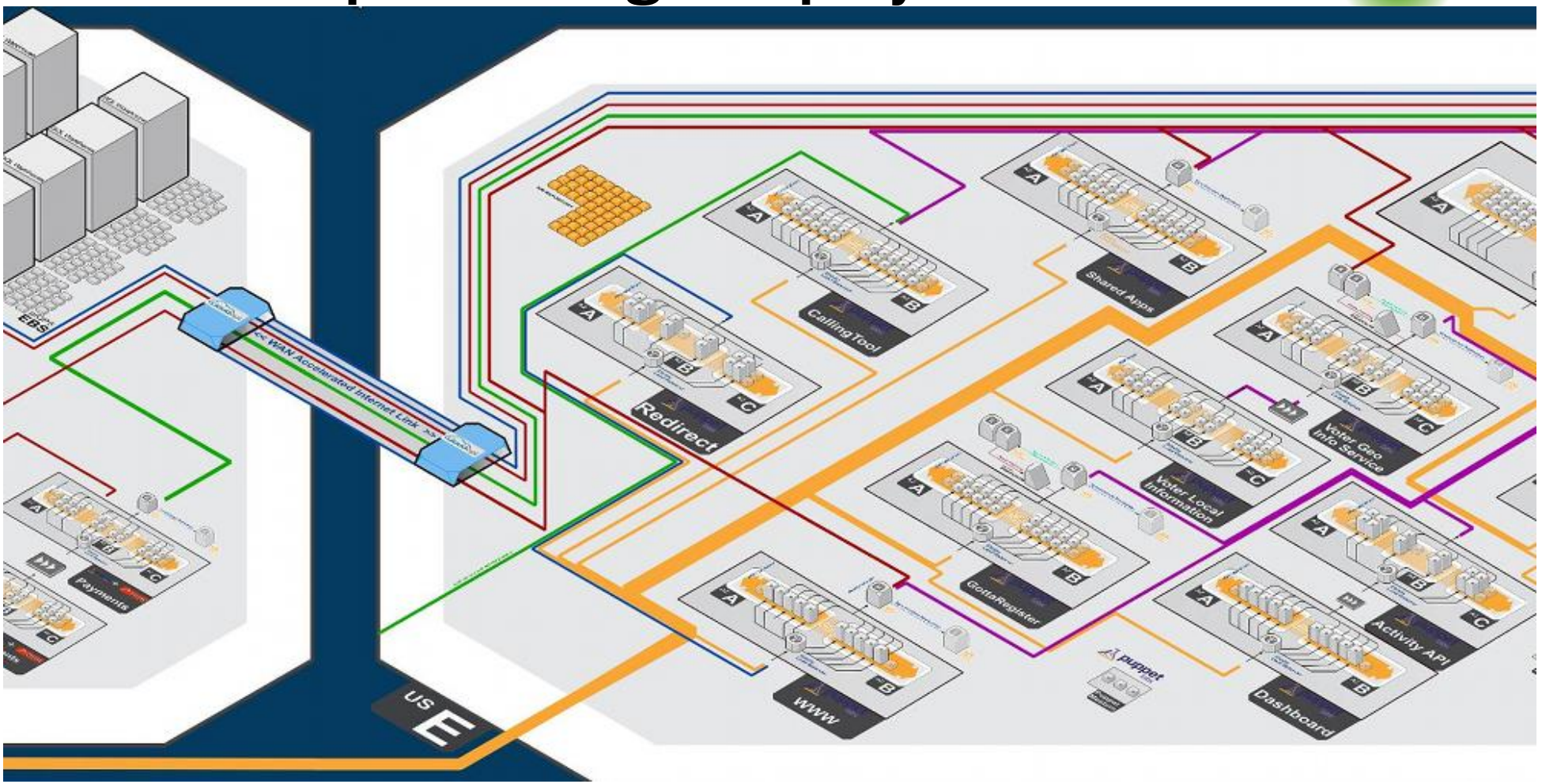
rackspace

verizon

IaaS is more than virtual machines

- Today's IaaS provides wide range of basic services used to build web/enterprise apps in pay-per-use
  - Load balander, DNS, object storage, CDN, email, queue, SQL/NoSQL database, big data analytics, payment, …
- Building and operating those services "properly" was often a huge cost for small/mid enterprises
- In IaaS, infrastructure operation is more or less to figure out a "right" combination of given services
  - Less need to worry about fault tolerance of those services

An Example of Large Deployment in AWS

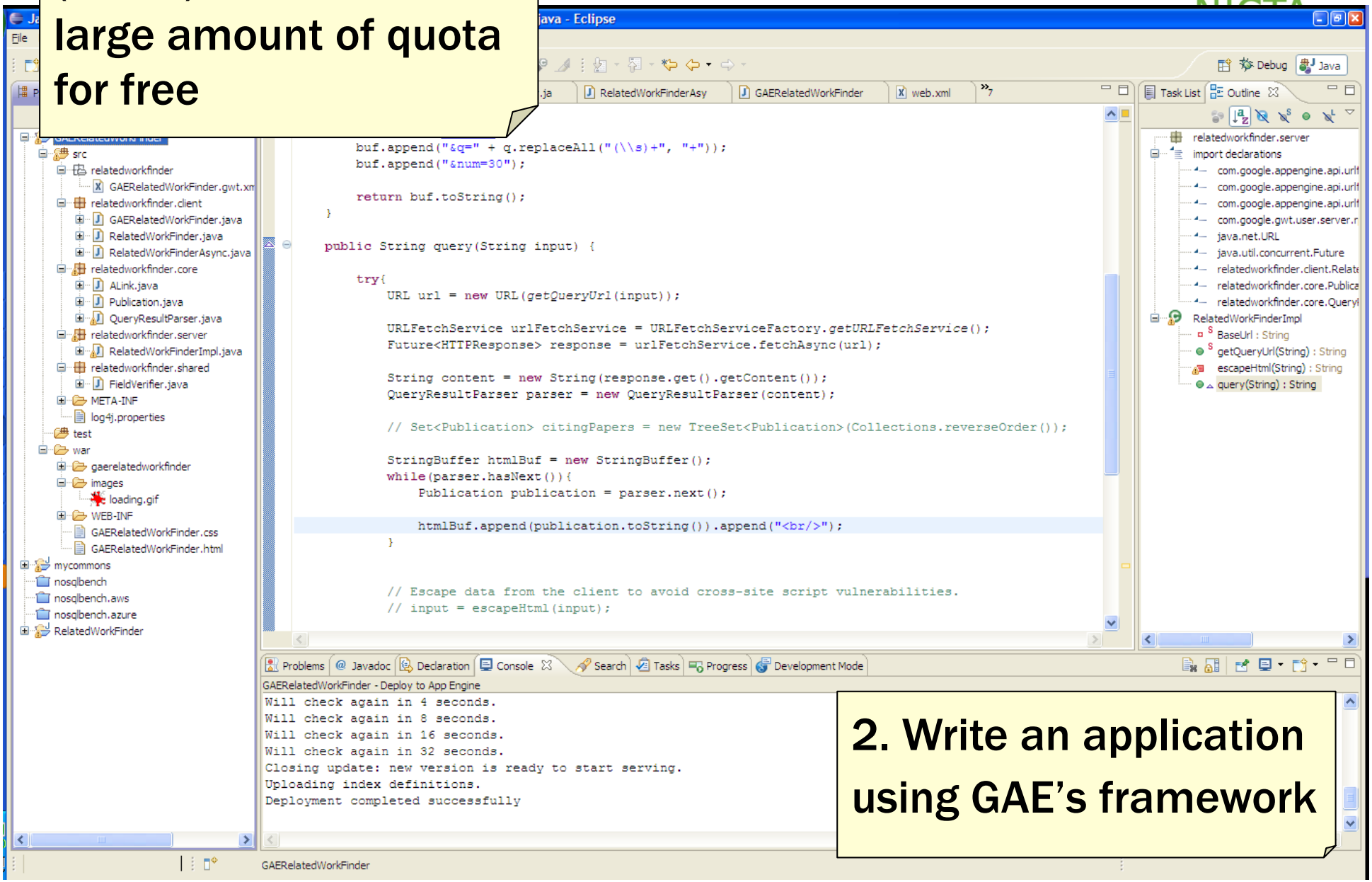


- aws.amazon.com/solutions/case-studies/obama/, awsofa.info
- 200 apps, 2k VMs, 180TB, 10k req/sec
- Very short lifetime
- Use IaaS services as much as possible (queue, LB, DB, NoSQL, Analytics, ...)

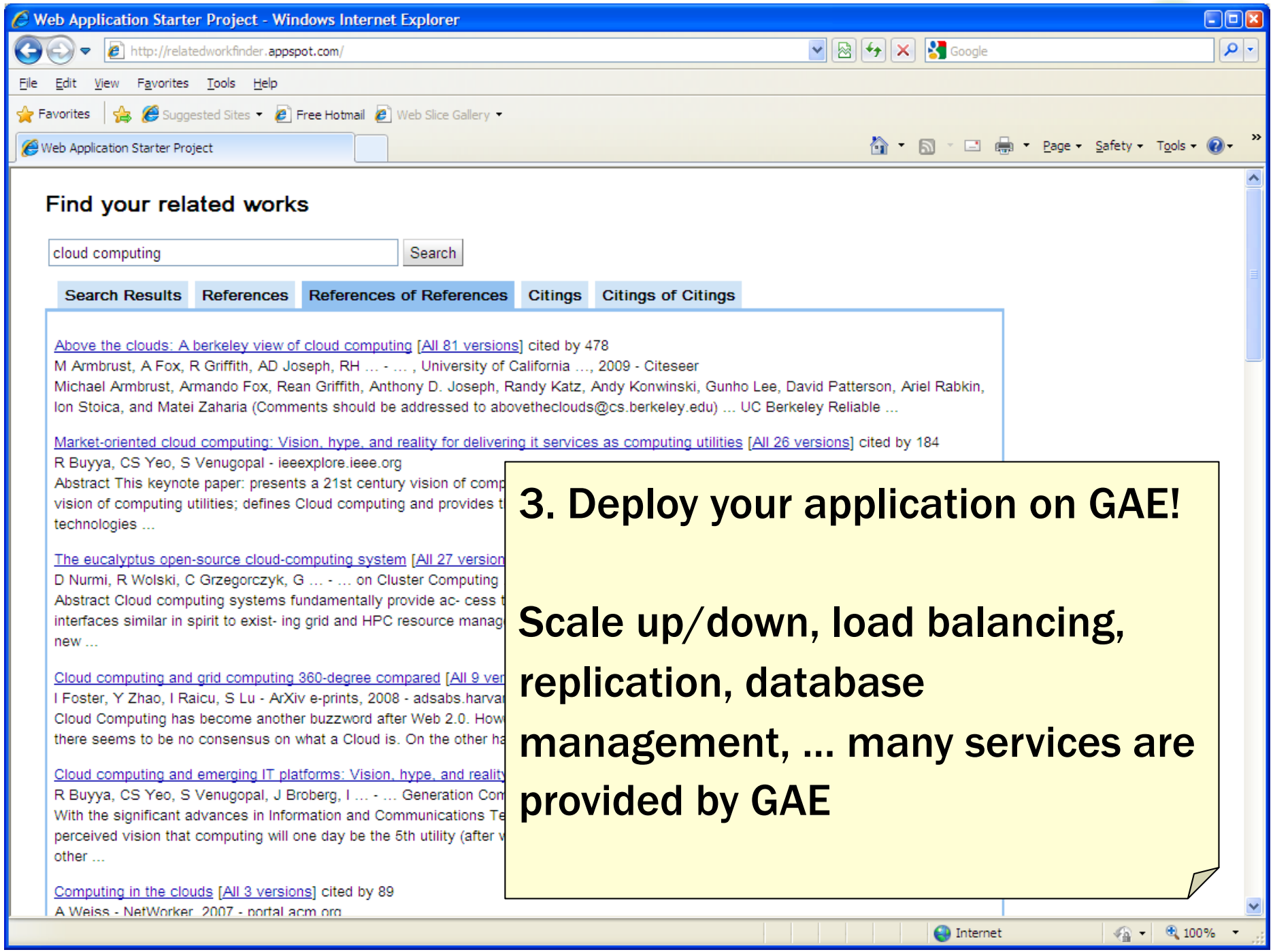
Example of PaaS: Google App Engine

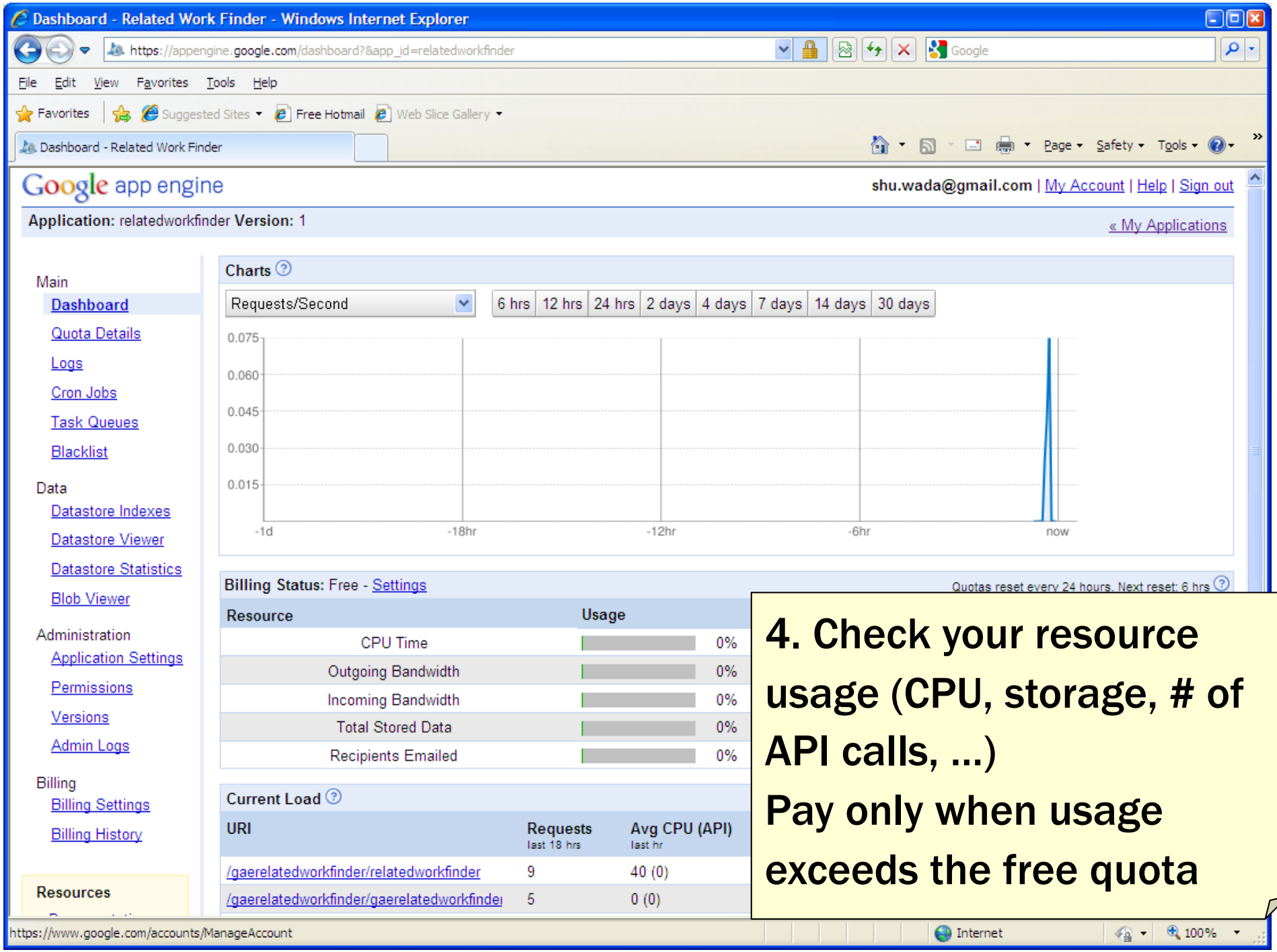
- "Platform as a Service" provides a solution stack
- Consumers "deploy" an application (e.g., uploading a war file) – do not touch VM/OS
- Platform manages scaling/backup/security
- Let's look into Google App Engine

1. Create an account. (5 min) GAE offers a large amount of quota for free



2. Write an application using GAE's framework


Web Application Starter Project - Windows Internet Explorer
http://relatedworkfinder.appspot.com/
Google
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
Web Application Starter Project
Page Safety Tools
Find your related works
cloud computing
Search
Search Results
References
References of References
Citings
Citings of Citings
Above the clouds: A berkeley view of cloud computing [All 81 versions] cited by 478
M Armbrust, A Fox, R Griffith, AD Joseph, RH ... - ... , University of California ..., 2009 - Citeseer
Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia (Comments should be addressed to abovetheclouds@cs.berkeley.edu) ... UC Berkeley Reliable ...
Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities [All 26 versions] cited by 184
R Buyya, CS Yeo, S Venugopal - ieeexplore.ieee.org
Abstract This keynote paper: presents a 21st century vision of comp
vision of computing utilities; defines Cloud computing and provides t
technologies ...
The eucalyptus open-source cloud-computing system [All 27 version
D Nurmi, R Wolski, C Grzegorczyk, G ... - ... on Cluster Computing
Abstract Cloud computing systems fundamentally provide ac- cess t
interfaces similar in spirit to exist- ing grid and HPC resource manag
new ...
Cloud computing and grid computing 360-degree compared [All 9 ver
I Foster, Y Zhao, I Raicu, S Lu - ArXiv e-prints, 2008 - adsabs.harva
Cloud Computing has become another buzzword after Web 2.0. How
there seems to be no consensus on what a Cloud is. On the other ha
Cloud computing and emerging IT platforms: Vision, hype, and reality
R Buyya, CS Yeo, S Venugopal, J Broberg, I ... - ... Generation Com
With the significant advances in Information and Communications Te
perceived vision that computing will one day be the 5th utility (after v
other ...
Computing in the clouds [All 3 versions] cited by 89
A Weiss - NetWorker, 2007 - portal.acm.org
3. Deploy your application on GAE!
Scale up/down, load balancing, replication, database management, ... many services are provided by GAE
Internet
100%



4. Check your resource usage (CPU, storage, # of API calls, ...)

Pay only when usage exceeds the free quota

Under the covers of

“REAL” CLOUD ARCHITECTURE

Based on “Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: the Consumers’ Perspective”, CIDR 2011

CLOUD ARCHITECTURE OVERVIEW AND NOSQL

Principle of (Large) Cloud Platform

- Cloud = “big computing resource” on top of large number of commodity servers
  - Commodity servers are cheap but not reliable
  - Many “cloud” technologies are designed to achieve high reliability by leveraging massive redundancy
    - Advancement in P2P research in 2000s made a significant contribution to the foundation of cloud

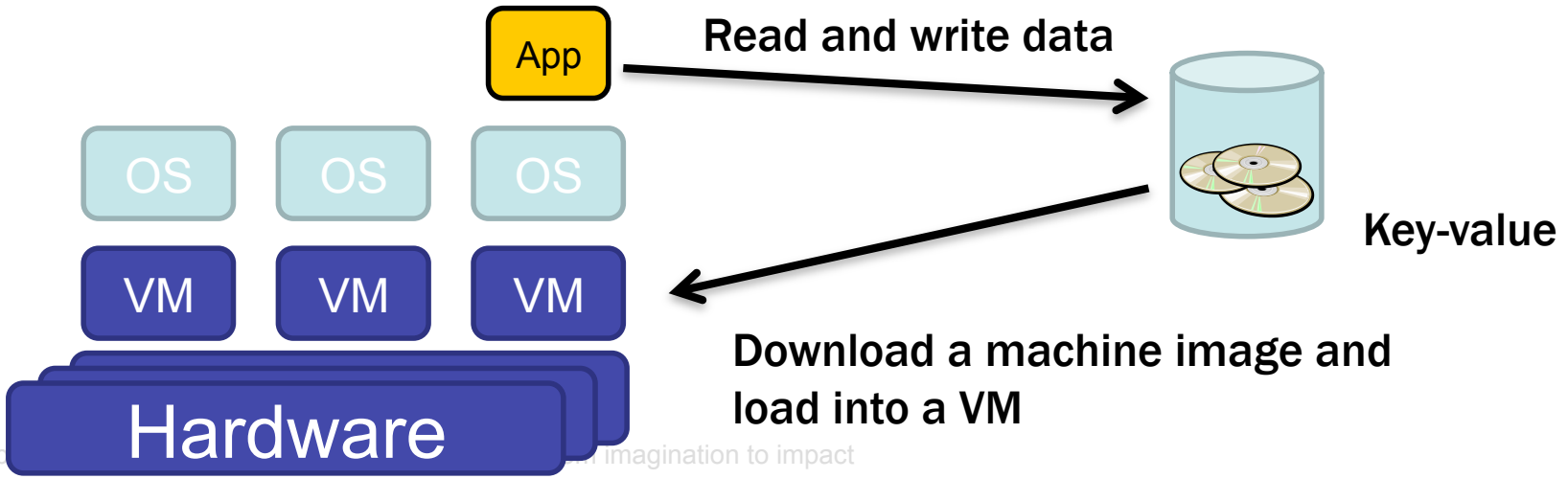
Computing and Storage

- CPU virtualization enables the sharing CPU resource
  - Relatively easy part since each virtual machine is stateless
- Storage/database technology in cloud has been evolving quickly
  - Today's cloud offer various data storage / database / content-distribution network / distributed cache
  - Common to mix them in one system
  - Critical to understand their pros and cons
- Network virtualization is hot now

~2008

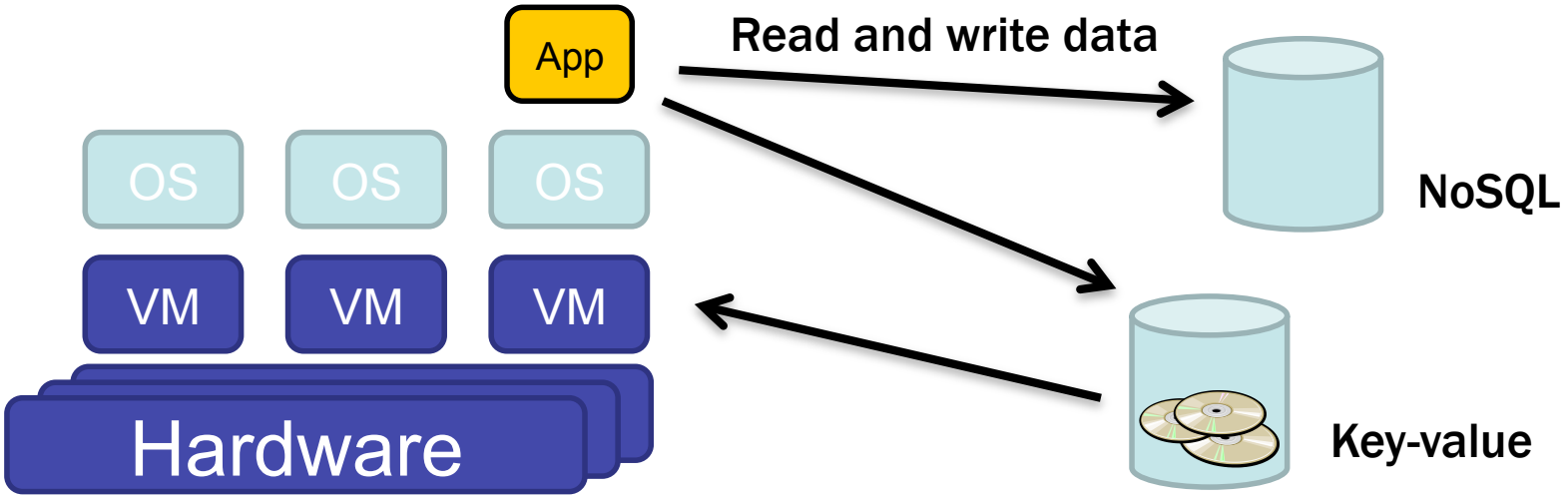
NICTA

- "Volatile" VMs and highly available key-value storage
- Each VM can use local non-persistent disks
  - On each boot a VM may "jump" to different physical server
  - On a server failure, a VM restart on a different server
- Replication and P2P (e.g., distributed hash) enables reliable key-value storage



2008 - 2009

- Vendors provided NoSQL (Not-only SQL) databases as “official” cloud storage
- As cloud computing was getting popular, “NoSQL” became a hot buzzword
  - Many product had emerged



Why no distributed block storage?

- Storage Area Network (SAN) was expensive!
  - Disks and Fibre Channel were not cheap

- Cloud = massive scale and throughput; however, making disk fast and reliable is hard
  - Scaling up is very expensive
  - Scaling out (RAID) is not cheap

Why no block storage? What's NoSQL?

- Designed to achieve high scalability and high availability for *essential* functionalities, trading off for some features of traditional DBMS
  - No relational data model: key-value or extension of key-value
  - Limited query languages: get/put by key, no or limited join (on same node), range query, …
  - No or limited transaction
  - Weak consistency: to achieve high availability and throughput on (possibly less reliable) distributed servers
- People wondered "how to build systems on NoSQL?"

Eventual Consistency

- A model originally proposed for disconnected operation (e.g., mobile computing)

- Different nodes keep replicas and each update is "eventually" propagated to each replica
  - DNS is the most well-known system implementing eventual consistency

- Usual definition is *counterfactual:*
  "once updating ceases, and the system stabilizes, then after a long enough period, all replicas will have the same value

Consistency from the Consumer's Perspective

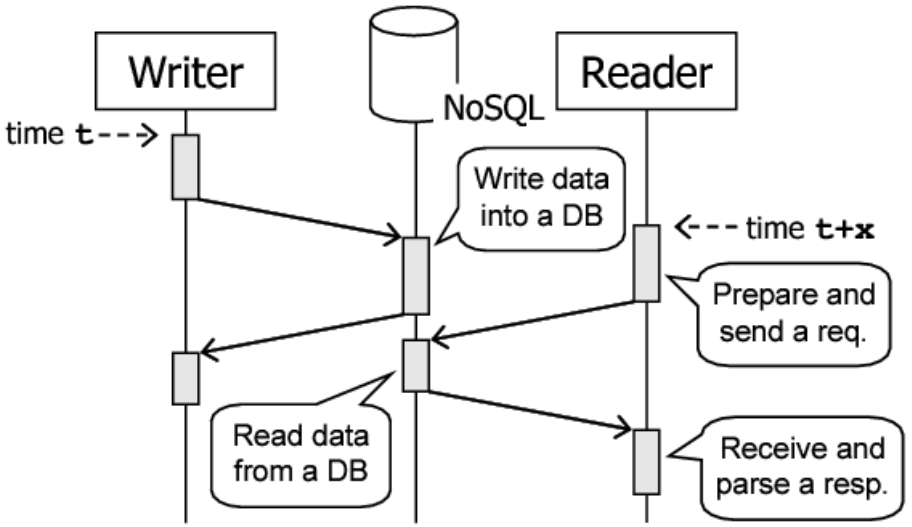
- Work done in "Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: the Consumers' Perspective", CIDR 2011
- Investigate consistency models provided by commercial NoSQLs in cloud
  - If weak, which extra properties supported? How often and in what circumstances is inconsistency (stale values) observed?
  - Any differences between what is observed and what is announced from the vendor?

Platforms we observed

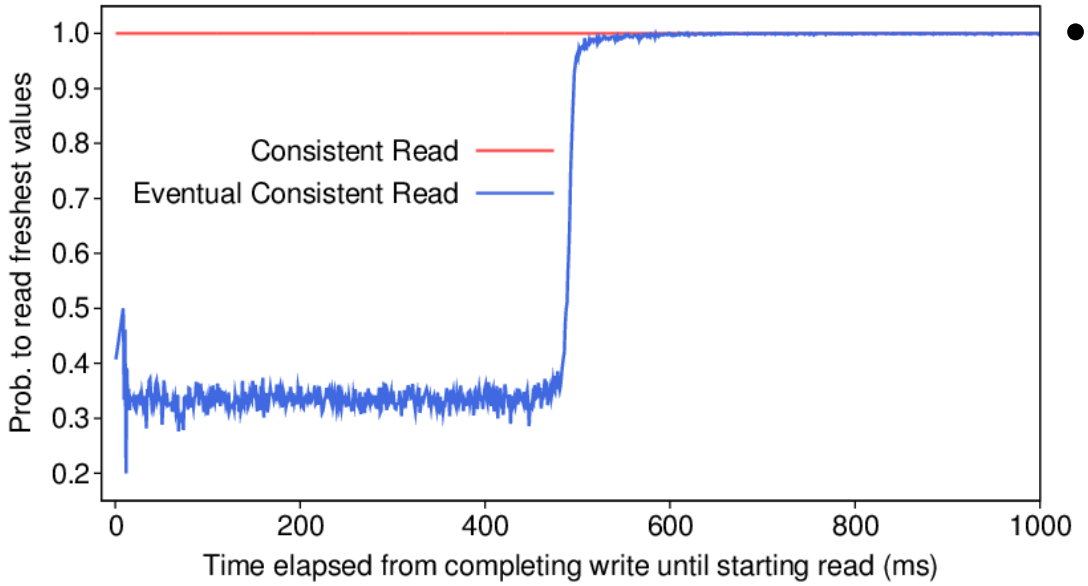
- A variety of commercial cloud NoSQLs that are offered as storage service
  - Amazon S3
    - Two options: Regular and Reduced redundancy (durability)
  - Amazon SimpleDB
    - Two options: Consistent Reads and Eventual Consistent Reads
    - (DynamoDB was introduced in 2012)
  - Google App Engine datastore
    - Two options: Strong and Eventual Consistent Reads (added “High Replication” in 2011)
  - Windows Azure Table and Blob
    - No option available in data consistency

Frequency of Observing Stale Data

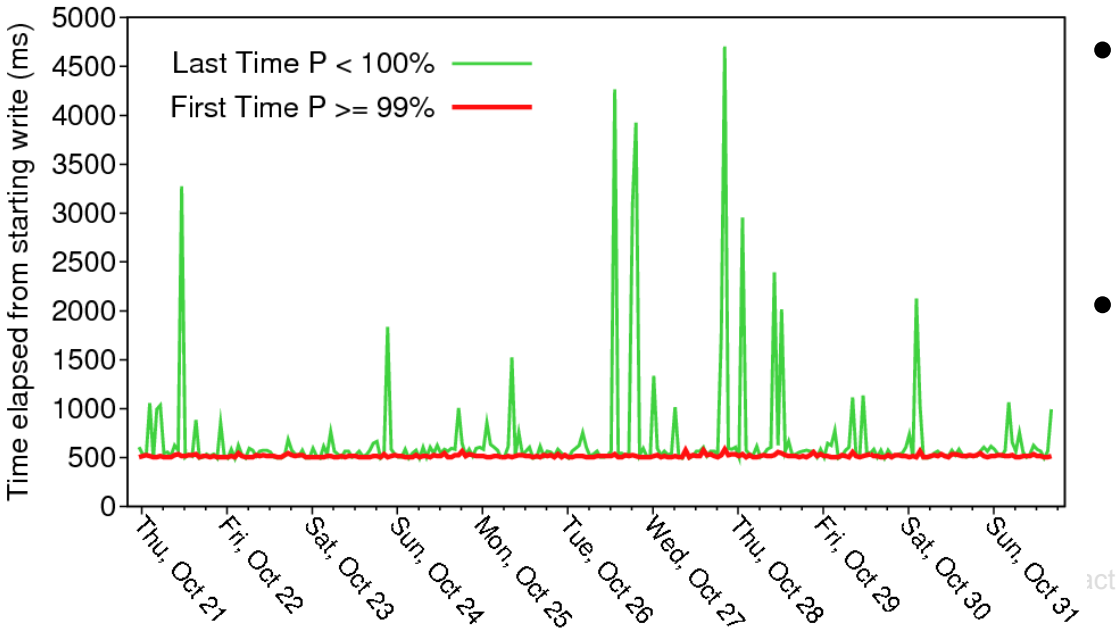
- Experimental Setup
  - A thread (or process) writes timestamp in storage
  - Another thread (or process) keeps reading from the storage
    - 100 times/sec
    - Check if the data is stale by comparing value seen to the most recent value written
    - Plot against time since most recent write occurred



SimpleDB: Read and Write from a Single Thread



- **With eventual consistent read, 33% of chance to read freshest values within 500ms**
  - **Perhaps one master and two other replicas. Read takes value randomly from one of these?**



- First time for eventual consistent read to reach 99% “fresh” is stable 500ms
- Outlier cases of stale read after 500ms, but no regular daily or weekly variation observed

Stale Data in Other Cloud NoSQLs

Cloud NoSQL and Accessing Source	What Observed
SimpleDB (access from one thread, two threads, two processes, two VMs or two regions)	Accessing source has no affect on the observable consistency. Eventual consistent reads have 33% chance to see stale value, till 500ms after write.
S3 (with five access configurations)	No stale data was observed in ~4M reads/config. Providing better consistency than SLA describes.
GAE datastore (access from a single app or two apps)	Eventual consistent reads from different apps have very low ($3.3E^{-4}$ %) chance to observe values older than previous reads. Other reads never saw stale data.
Azure Storages (with five access configurations)	No stale data observed. Matches SLA described by the vendor (all reads are consistent).

Additional Properties: Read-Your-Writes?

- Read-your-writes: a read always sees a value at least as fresh as the latest write from the same thread/session
  - Once you write x, you will read x
- SimpleDB with eventual consistent read: does not have this property
- GAE with eventual consistent read: may have this property
  - No counterexample observed in ~3.7M reads over two weeks

Additional Properties: Monotonic Reads?

- Monotonic Reads: Each read sees a value at least as fresh as that seen in any previous read from the same thread/session
  - Write x then y. Once you read y, you never see x.
- Our experiment: check for a fresh read followed by a stale one
- SimpleDB: Monotonic read consistency is not supported
  - In staleness, two successive eventual consistent reads are almost independent

	2nd Stale	2nd Fresh
1st Stale	39.94% (~1.9M)	21.08% (~1.0M)
1st Fresh	23.36% (~1.1M)	15.63% (~0.7M)

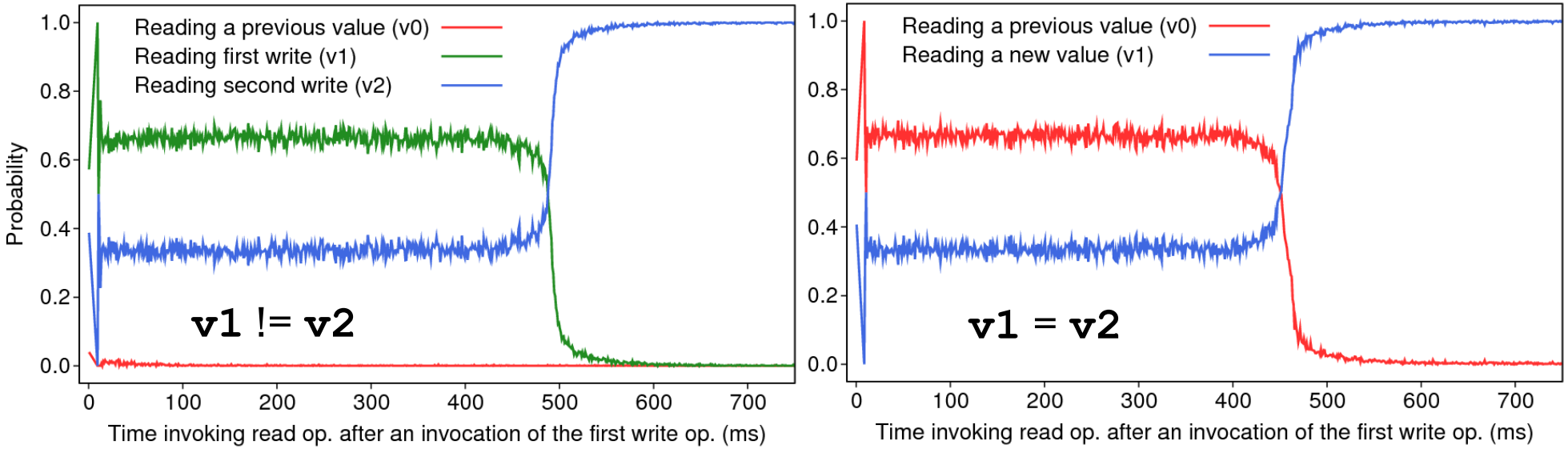
- GAE with eventual consistent read: not supported
  - 3.3E-4 % chance to read values older than previous reads

Additional Properties: Monotonic Writes?

- Monotonic Writes: Each write is completed in a replica after previous writes have been completed
  - Write x then y. you may read y then read x sometime
- Programming is “notoriously hard” if monotonic write consistency is missing
  - W. Vogels. Eventually consistent. Commun. ACM, 52(1), 2009.

SimpleDB's Eventual Consistent Read: Monotonic Write

- A data has value v0 before each run. Writing value v1 and then v2 there, then read it repeatedly



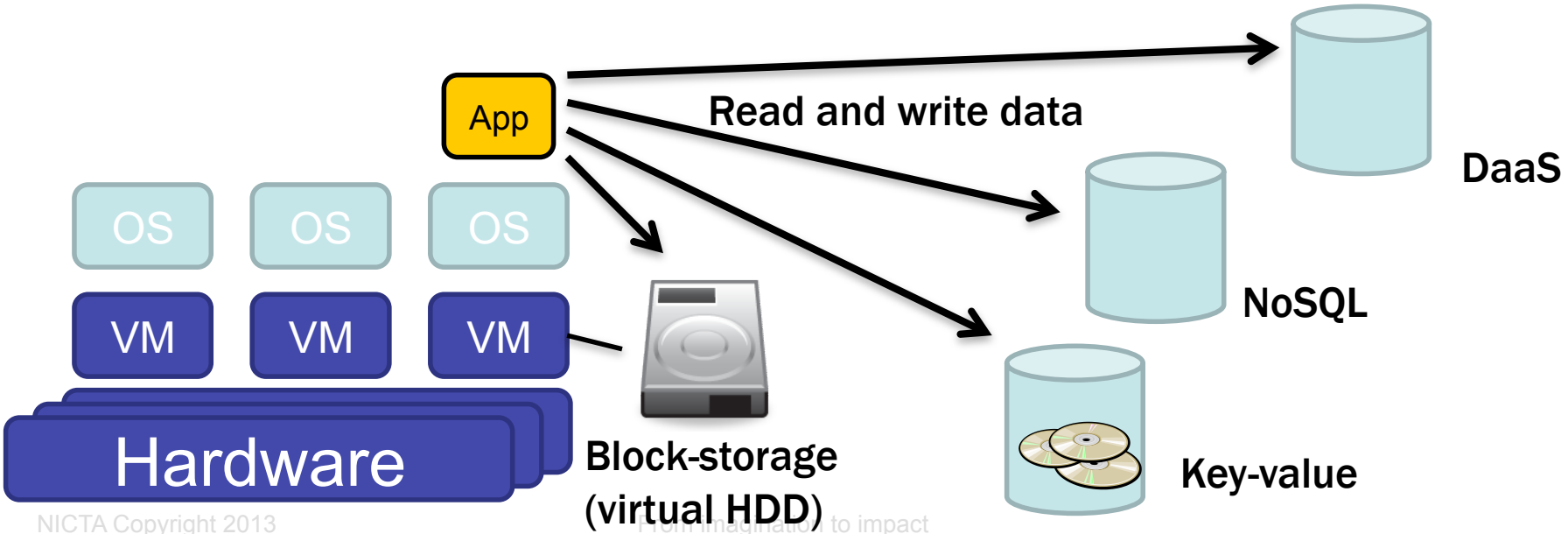
- When **v1 != v2**, writing **v2** "pushes" **v1** to replicas immediately (previous value **v0** is not observed)
- When **v1 = v2**, second write does not push (**v0** is observed)
  - Same as the "only writing one value" case

What Consumers Can Observe

- SimpleDB platform showed frequent inconsistency
- It offers option for consistent read. No extra costs for the consumer were observed from our experiments
  - At least under the scale of our experiments (few KB stored in a domain and ~2,500 rps)
- Some platforms gave (mostly) better consistency than they promise
  - Consistency almost always (5-nines or better)
  - Perhaps consistency violated only when network or node failure occurs during execution of the operation

2010 to 2011

- Vendors started providing virtual persistent block storage in cloud
  - on Storage Area Networks (SANs)
- Also started providing RDBMS services in cloud
  - Database-as-a-service
  - RDBMS pre-installed VMs and managed by vendors



Trends in NoSQL after 2011

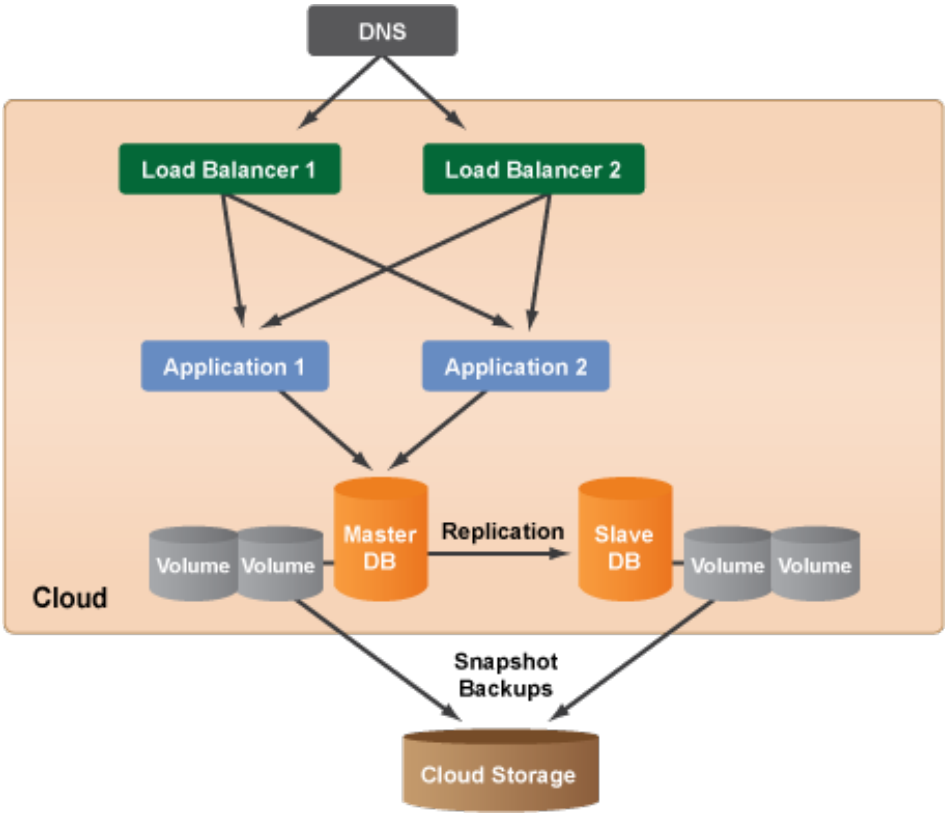
- "NoSQL" is accepted as a new class of storage
  - High throughput, less system administration, easy to build fault tolerance over geo-replication, schema less and easy to write client code, …
- Popular NoSQL implementations are everywhere
  - DynamoDB, MongoDB, CouchDB, Redis, …
  - But no sign of "common NoSQL model/API" yet
- Many NoSQL products offer consistent view
  - Some offer tunable consistency
- Research effort to "revive missing features" on NoSQL

APPLICATION ARCHITECTURE AND OPERATIONS IN CLOUD

Design for Scale

- Scale horizontally (out/in)
  - Adding/removing nodes to/from a system
  - Architecture could be complex
  - Highly available
  - Good for stateless layers (e.g., web servers and computing array)
- Scale vertically (up/down)
  - Adding/removing resource to/from a node
  - Easy to implement but potentially expensive
  - Hard to avoid downtime on scale, less availability
  - Good for stateful layers (e.g., relational database)
- For small-mid applications, scale up/down is not a bad choice!

3-Tier Architecture for Scale

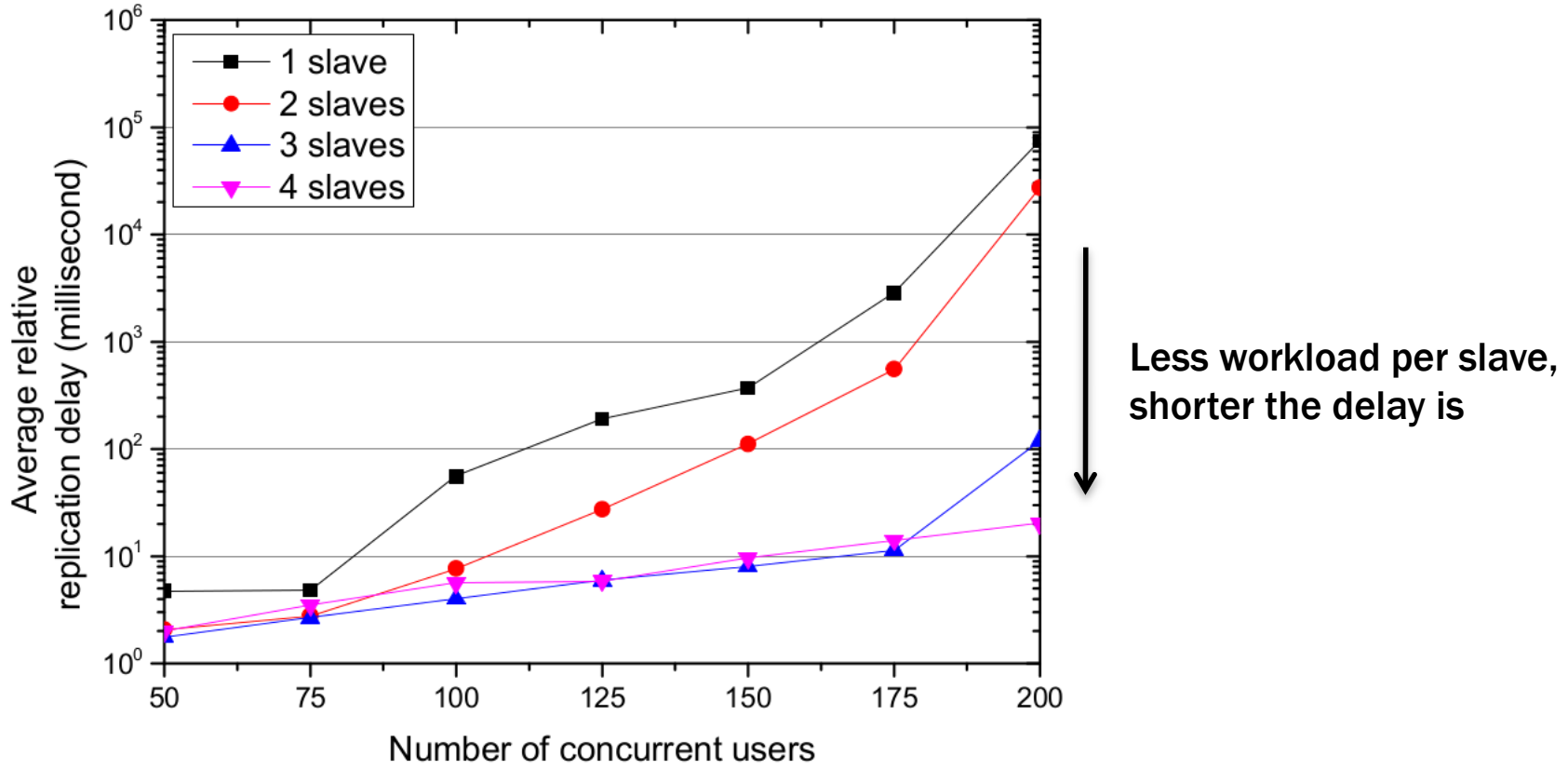


support.rightscale.com/12-Guides/
EC2_Best_Practices/EC2_Site_Architecture_Diagrams

- Dynamic DNS
  - Distribute traffic among public-facing endpoints
- Load-balancer
  - Can handle large traffic
  - Easy to scale out
- Application layer
  - Stateless and easy to scale out
  - Cache efficiency, session stickyness, ...
- Persistent layer
  - Stateful and hard to scale out
  - Master-slave for scaling and fault tolerance

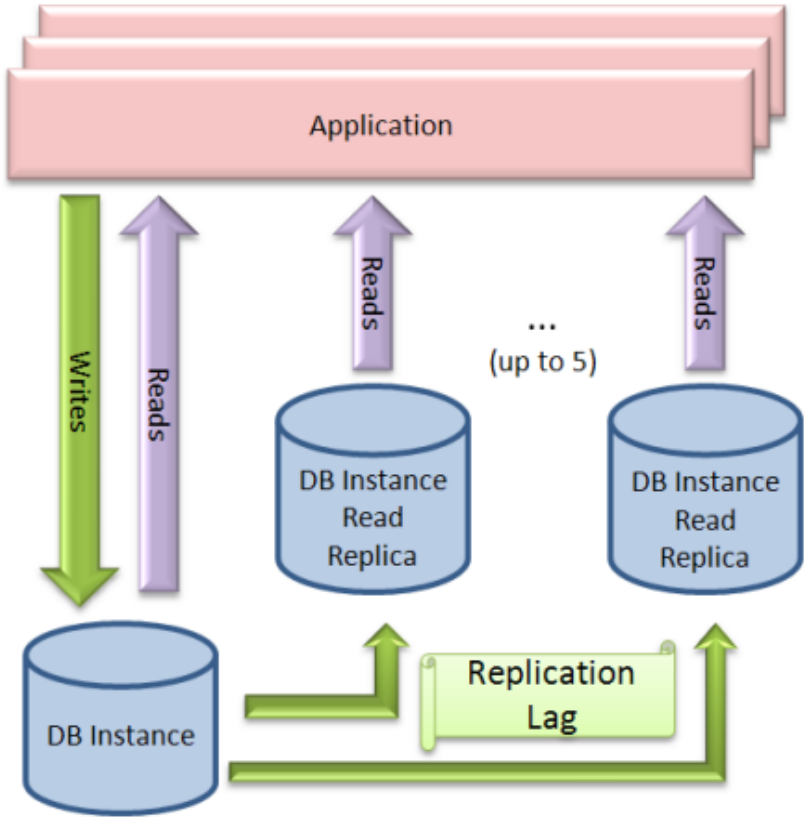
Master-Slave Architecture (Persistent Layer)

- Create a slave(s) for two purposes: scaling and fault tolerance
- Asynchronous replication is used often ➔ replication delay between a master and the slave



"Application-managed database replication on virtualized cloud environments", ICDEW 2012

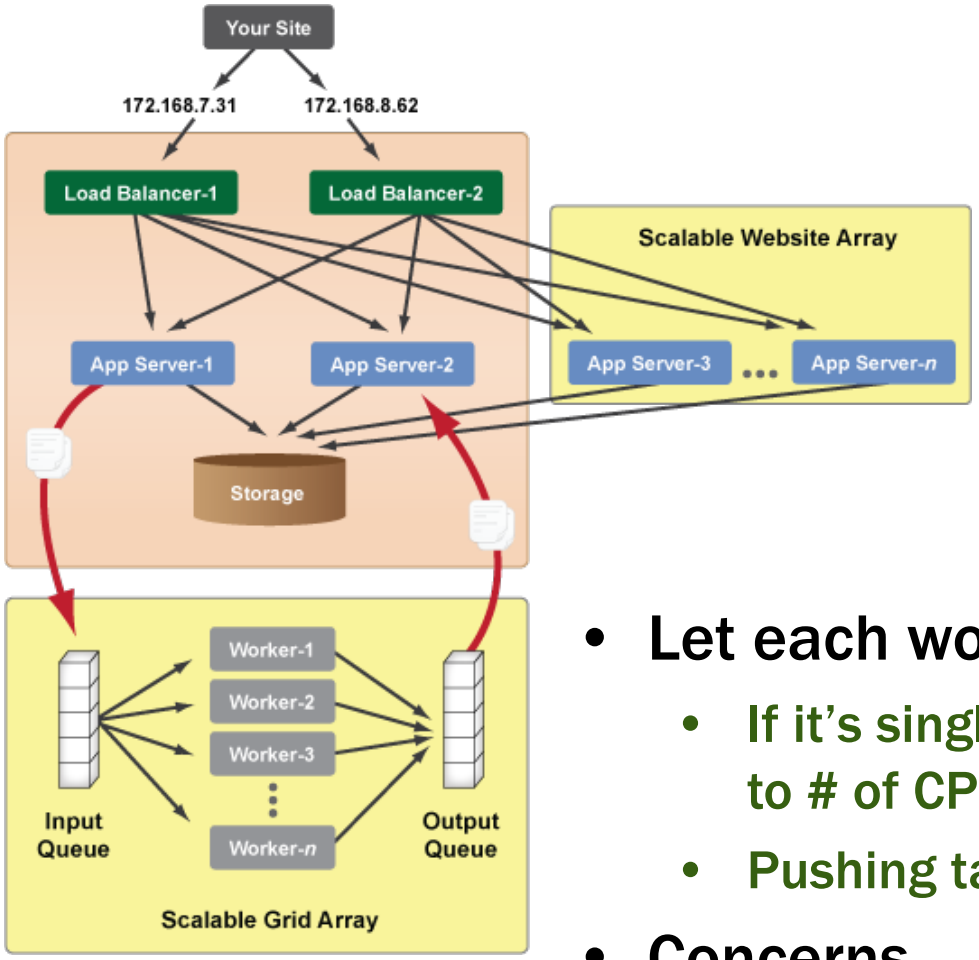
Read-Replica



aws.typepad.com/aws/2010/10/amazon-rds-announcing-read-replicas.html

- Using a slave as a read-only copy is a common technique
- Achieving FT and scaling
  - Typical read-write ratio is 90:10
  - Due to replication delay, it's good to read only from read replicas
  - Only one slave and forward 90% transactions to it does not scale out
  - Using a master and the slave over 50% of capacity ➔ Two "single point of failures"

Queue-based Architecture



support.rightscale.com/12-Guides/EC2_Best_Practices/EC2_Site_Architecture_Diagrams

- Long running tasks should be executed asynchronously
  - Holding HTTP connections wastes resource
- Easy to scale in/out
  - Adjust the # of workers (VMs) based on the queue length

- Let each worker adjust the concurrency
  - If it's single threaded, each worker can process up to # of CPUs
  - Pushing tasks poses the risk of node crash
- Concerns
  - Timing of scaling out/in, restarting jobs upon a node failure or scaling in, prevent nodes from popping the same request, task priority, ...

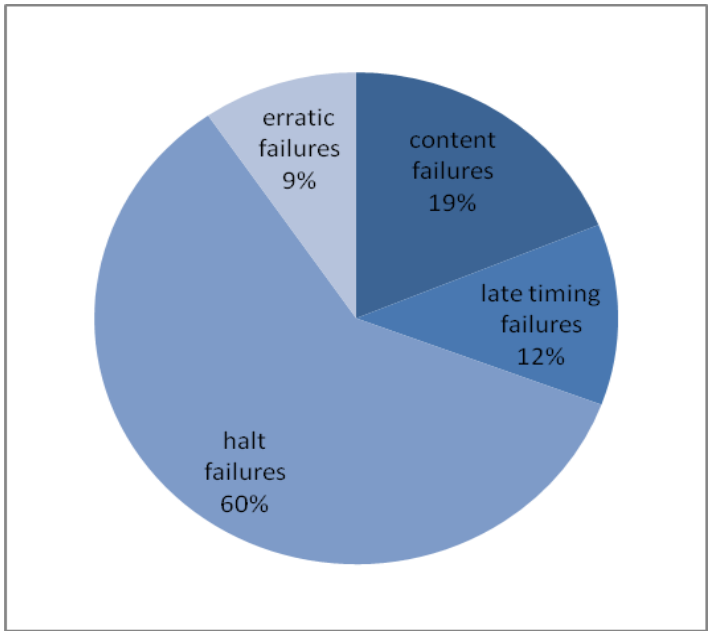
Reliability of Cloud

- (Public) cloud computing is a "best-effort" service
- E.g., Amazon's SLA on availability is 99.95%
  - AWS will use commercially reasonable efforts to make Amazon EC2 available with an Annual Uptime Percentage of at least 99.95%
  - "Annual Uptime Percentage" is calculated by subtracting from 100% the percentage of 5 minute periods during the Service Year in which Amazon EC2 was in the state of "Region Unavailable."
  - "Region Unavailable" and "Region Unavailability" means that more than one Availability Zone in which you are running an instance, within the same Region, is "Unavailable" to you.
- Availability of each VM could be much lower than 99.95%

You have to prepare for failures

NICTA

- Failures are common in cloud
  - No responding VM due to an underlying hardware (network) failure
  - Wrong result from an API call
  - Takes too long to process an request (not a “failure”)
  - An API call fails
  - A resource gets ‘stuck’ in an error state
  - A data center wide failure (e.g., network connection)
  - Reboot request



“API Issues: an Empirical Study and Impact”, QoSA 2013

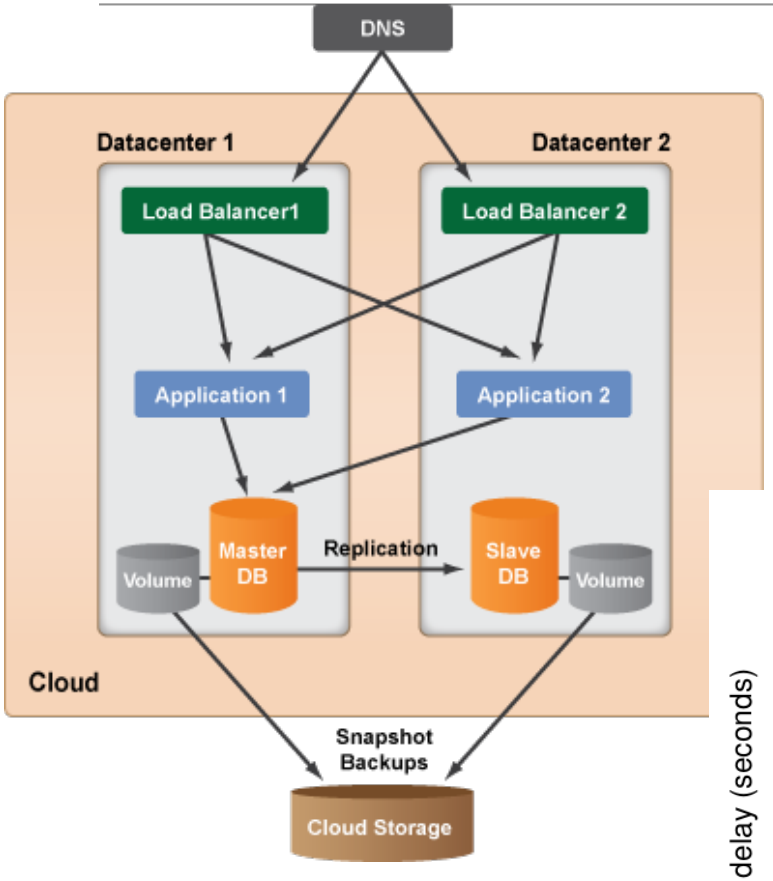
Design for Failure

- Cloud providers explicitly say they do not guarantee 100% availability
  - They provide redundancy instead
- You must architect your system to survive random failures

http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html

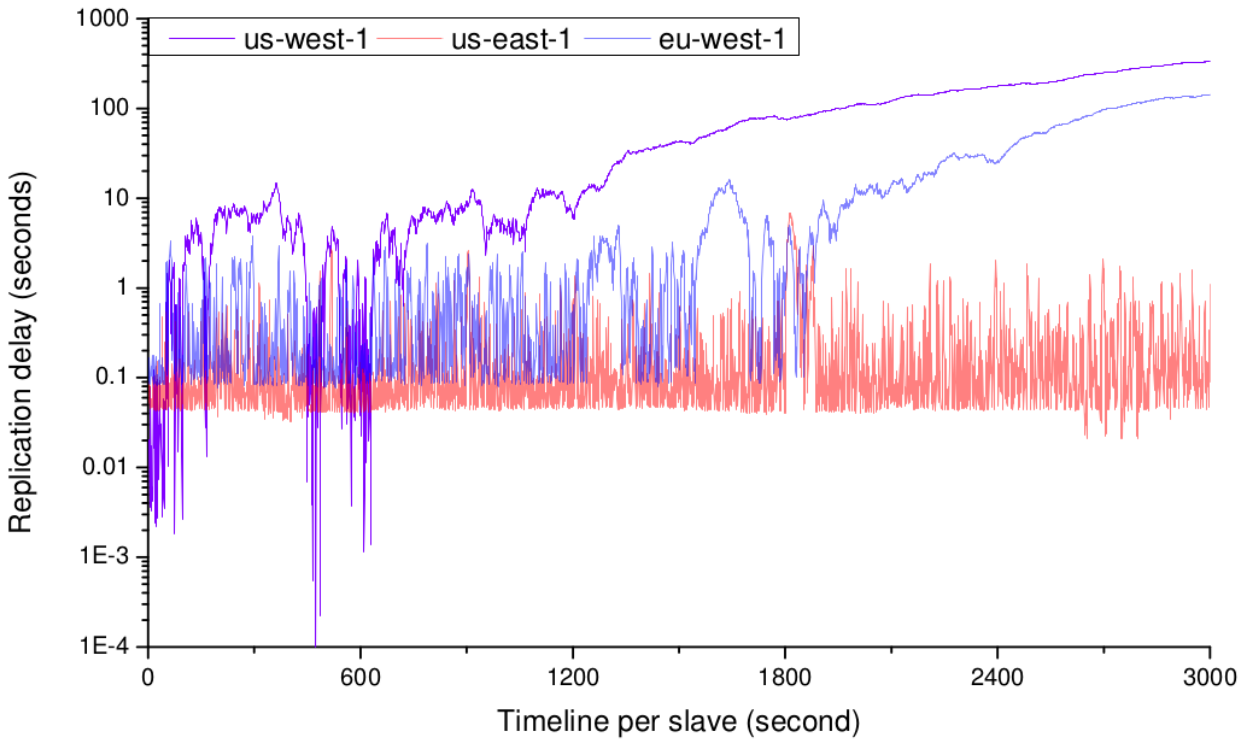
- Chaos monkey: test your system against random failures
  - "The best way to avoid failure is to fail constantly" by Netflix
  - Randomly shutdown virtual machines, services, …

Geographically-distributed architecture



support.rightscale.com/12-Guides/EC2_Best_Practices/EC2_Site_Architecture_Diagrams

- Cloud allows for using geographically dispersed data centers
  - > 50km away, on different utility infrastructure (e.g., power, water, road, ...)
  - Traditionally very expensive solution
  - The speed of light is your enemy!



"A framework for consumer-centric SLA management of cloud-hosted databases", IEEE Trans. Serv. Comp. 2013

Failure Handling at  Yuruware

- Chaos monkey is not only for random failures
  - Strange errors occur everywhere: cloud, os, app, …
- Easiest error handling = discard and get new one
- What we're doing
  - Worker VM kills itself if detecting a fatal app error
  - Worker VM kills itself if not executing jobs or not receiving HB from management server for an hour
  - The mgmt server kills workers if not responding
  - Let processes/VMs clean up the mess and restart from start
- Key design: error detection, autonomous decision making, loose coupling and async process
- "micro-rebooting" in cloud makes your app robust

Configuration Management

- Config management system is a great help
  - Puppet, Chef, …

```
node basenode {
        $botl_opt_dir = "/opt/ybolt.net"
        include env::swapfile
        include env::hostname
        class { logwatch :
                logwatch_mail_to => $ops_email_addr
        }
}
node 'prod.ybolt.net' inherits boltserver {
        mysql::db { "$mysql_bolt_db" }
        class { mysql_s3_backup : s3_bucket => "…"}
}
```

  - Make all environments consist
  - Restart an environment quickly
    - Micro-rebooting, moving from local to cloud, …