Logistics

- Last class with materials today
  - Need to cover 3 big topics :-(
  - Will dedicate 1 hour each and cover as much as possible
  - You can read the slides home
  - Nothing related to the project
- Class schedule
- Class project
  - Deliverable 4 is out
  - Deliverable 5: Presentation
  - Upload slides on Gdrive
  - May 11 (next week) final presentation
    - 15 mins per team, 4 slides (one per person)
    - Unless everyone wants to move it, we will keep it May 11
- Hopefully you enjoyed the different approach to the class
  - More experiential / hands-on, rather than memorize-and-regurgitate
- Sorrentum project
  - Internships
  - Stay engaged
  - Start companies
  - Funding will be available

UMD DATA605 - Big Data Systems

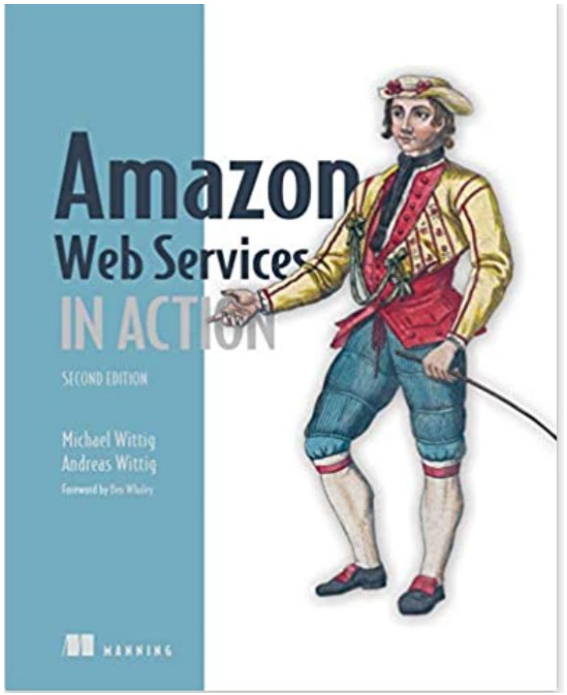
AWS Overview

Dr. GP Saggese

gsaggese@umd.edu

AWS: Resources

- Some basic info in the slides
- Many tutorials on-line
- Mastery
  - Amazon Web Services in Action
    3rd Edition

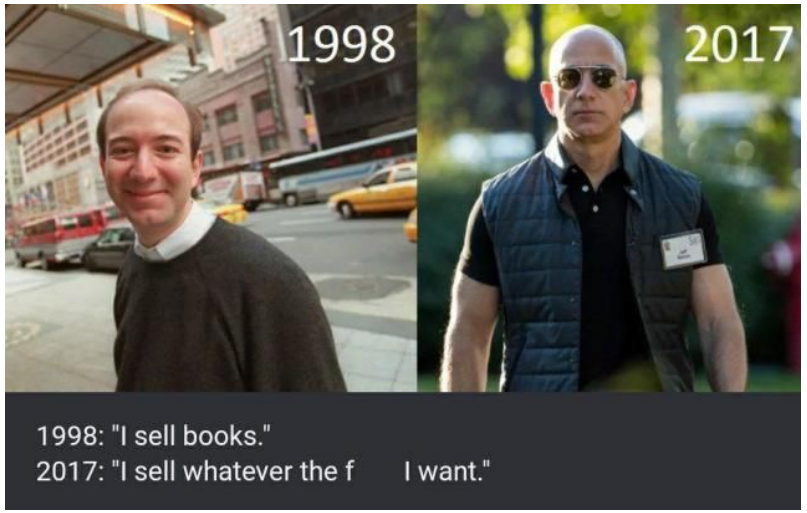


Amazon Web Services (AWS)

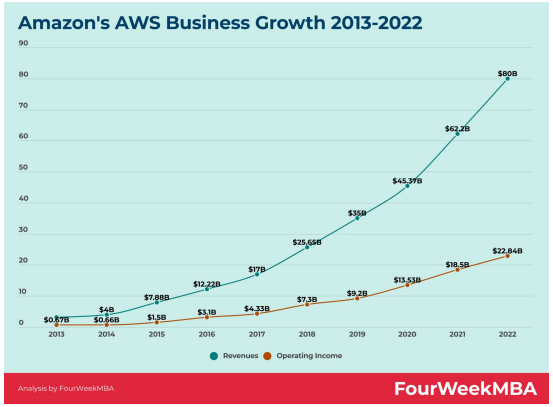
- **AWS is a platform offering complete cloud solutions**
  - Computing (e.g., EC2)
  - Storing (e.g., S3)
  - Networking
- **It offers different levels of abstractions**
  - IAAS, PAAS, SAAS
  - From virtual hardware to applications, e.g.,
    - Host web sites
    - Run enterprise software
    - Run machine learning applications
- **Services can be controlled in different ways**
  - A web interface (aka “console”)
  - CLI
    - `aws` command
  - Programmatically
    - Through language libraries, SDK (e.g., Python `boto3`)

AWS as Business

- Services are charged on a pay-per-use pricing model
- Data centers are distributed globally
  - US, Europe, Asia, South America
- 500 new services and features every year
- Insanely profitable
  - $62B dollar / year in revenue
  - Grows 42% year-over-year
  - Controls 30% of cloud business



Beef Jezos



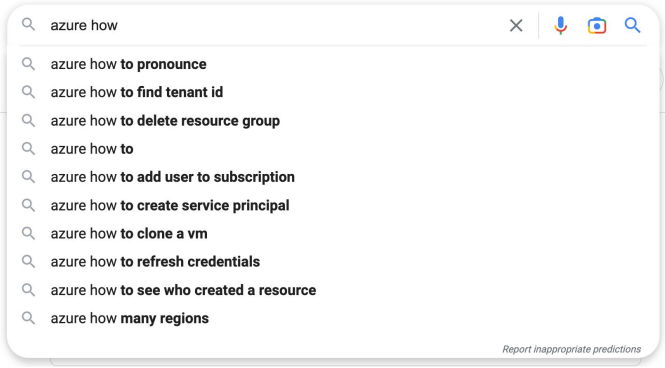
Types of Cloud Computing

- Cloud computing enables a shared pool of configurable computing resources
  - E.g., servers, storage, networks, applications, services

  to be used:
  - From everywhere
  - Conveniently
  - On-demand
- Clouds can be:
  - *Public*: open to use by general public (e.g., AWS)
  - *Private*: virtualize and share IT infrastructure within a single organization (e.g., government)
  - *Hybrid*: a mixture of public and private clouds

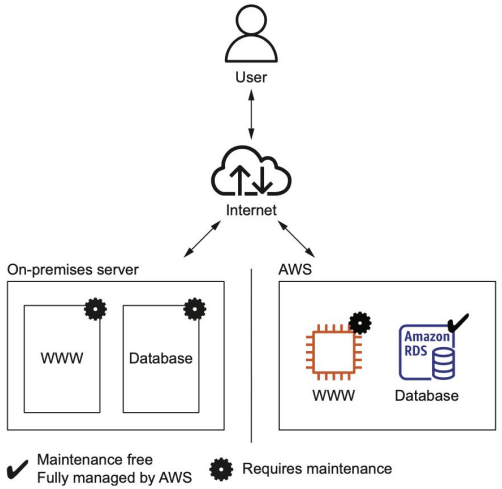
AWS vs Google Cloud vs Microsoft Azure

- **Similarities:**
  - Worldwide infrastructure
  - Provide computing, networking, storage as IAAS
    - AWS EC2 vs Google Compute Engine vs Azure VMs
    - AWS S3 vs Google Cloud Storage vs Azure Blob storage
  - Pay-as-you-go pricing model
- **Differences:**
  - AWS is market leader, most mature, and powerful
  - AWS (ab)uses a lot of open source technologies
  - Azure provides Microsoft stack in the cloud
  - Google seems more focused on cloud-native applications rather than migrating local applications to the cloud

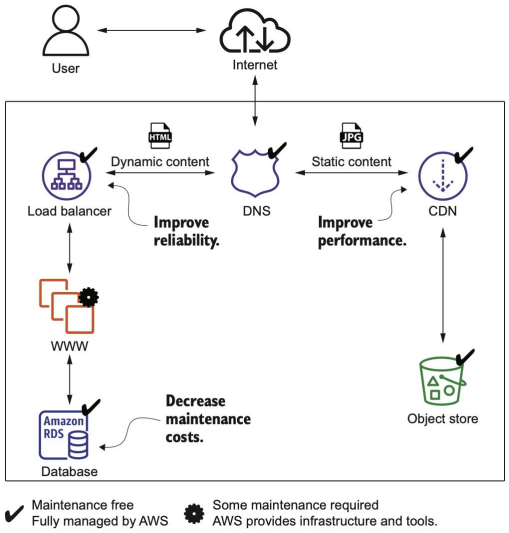


From On-premise to AWS

- **Move a medium-sized e-commerce site from on-premise to the cloud**
- **Architecture**
  - Web-server: handle requests from customers
  - DB: store product information and orders
  - Static content (e.g., JPEG image)
    - Delivered over a content delivery network (CDN)
    - Reduce load on company services
  - Dynamic content (e.g., HTML pages)
    - E.g., products and prices
    - Delivered by web server
- **Step 1: Move to cloud**
- **Step 2: Design for the cloud**
  - DNS
  - Database
  - Object store (S3)
  - Potentially managed solutions
  - Use multiple smaller virtual services using load balancer to increase reliability



Step 1: Move to cloud



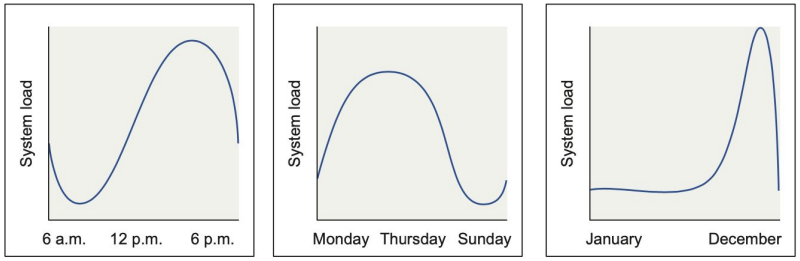
Step 2: Design for cloud

Low-Cost Batch Processing

- Batch jobs run on a schedule
  - E.g., analyze data once a day
  - E.g., tools generate reports from the DB
- AWS bills VMs per minute
  - Instead of paying for machines in traditional data center, pay only when you run your job
- AWS offers spare capacity at a discount
  - It's not crucial to execute the batch jobs at a specific time
  - Run when there is capacity, e.g., saving 50%

Capacity Scaling

- **No need to plan for capacity**
  - You can schedule capacity on-the-fly
  - You don't have to predict capacity
  - No need to think about rackspace, switches, power supplies
  - If your system grows, add more VMs (from 1 to 1000) and storage (from GB to PB)
- **You can handle seasonal traffic patterns by scaling up / down the, e.g.,**
  - Day vs night
  - Weekday vs weekend
  - Holiday
  - E.g., you don't need a test system when the team is off (at night and during the weekend)
- **Worldwide presence**
  - AWS has many data centers
  - You can deploy applications close to your customers



Analytics	Application integration	AR and VR
AWS cost management	Blockchain	Business applications
Compute	Containers	Customer enablement
Database	Developer tools	End-user computing
Frontend web and mobile	Game Development	Internet of Things
Machine learning	Management and governance	Media services
Migration and transfer	Networking and content delivery	Quantum technologies
Robotics	Satellite	Security, identity, and compliance
Storage		

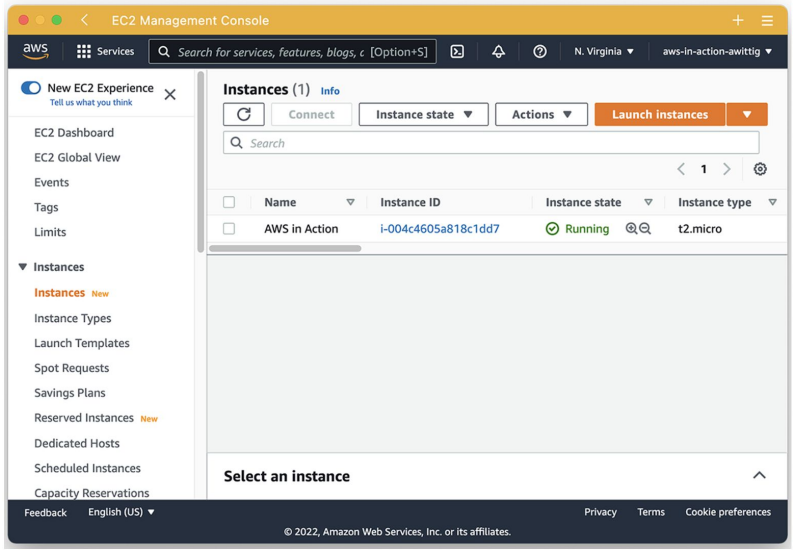
Pay-per-use

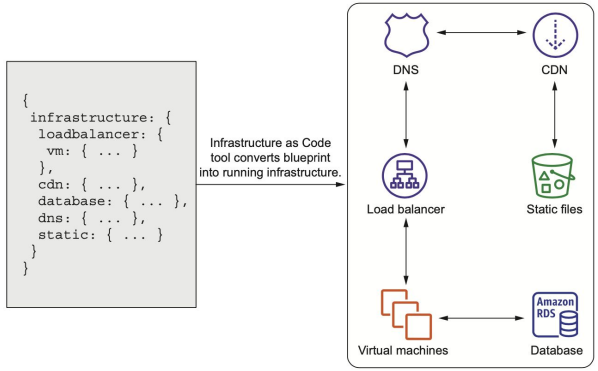
- **AWS bill**
  - Similar to an electric bill
  - Services are billed based on usage, e.g.,
    - Hours of virtual server (rounded up to hours)
    - Used storage in GB (allocated or real capacity)
    - Data traffic in GB or number of requests
- Free tier
  - Use some AWS services for free for 12 months after signing up
  - Get experience using services (e.g., EC2, S3)
  - If you exceed the limits, you start paying (without further notice)
    - Need to set an alarm
- **Advantages**
  - No need for upfront investments or commitment
  - The price to start a project is lowered
  - Easier to divide system into smaller parts, because the cost is the same
    - One big server or two smaller ones with the same capacity have the same price
  - Make fault tolerance / high performance affordable

Service	January usage	February usage	February charge
Visits to website	100,000	500,000	
CDN	25 M requests + 25 GB traffic	125 M requests + 125 GB traffic	$115.00
Static files	50 GB used storage	50 GB used storage	$1.15
Load balancer	748 hours + 50 GB traffic	748 hours + 250 GB traffic	$19.07
Web servers	1 virtual machine = 748 hours	4 virtual machines = 2,992 hours	$200.46
Database (748 hours)	Small virtual machine + 20 GB storage	Large virtual machine + 20 GB storage	$133.20
DNS	2 M requests	10 M requests	$4.00
Total cost			***$472.88***

Interacting with AWS

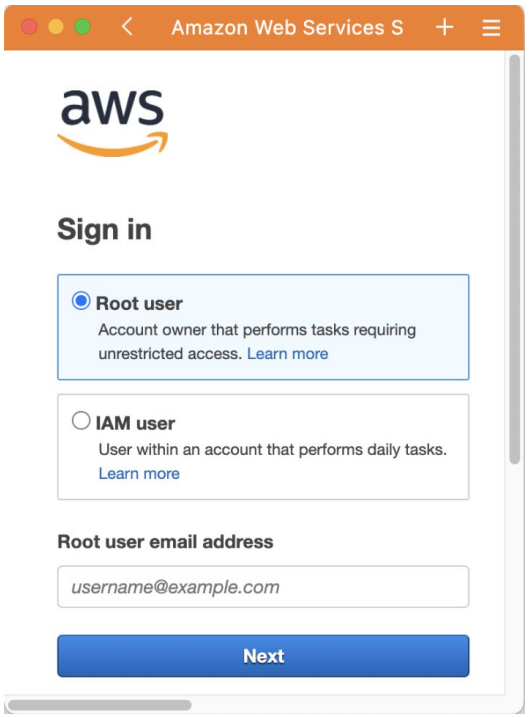
- **AWS internal API allows to interact with services**
- **GUI (Management Console)**
  - Easy to start interacting with services
  - Set up a cloud infrastructure for development and testing
- **Command-line tool (CLI)**
  - Manage and access AWS services
  - Automate recurring tools
- **SDKs**
  - Use library in almost any language to interact with AWS
  - Integrate applications with AWS
  - E.g., **`boto3`** for Python
- **Blueprints**
  - Description of your system containing all services and dependencies
  - It describes the system, it does not explain how to build it





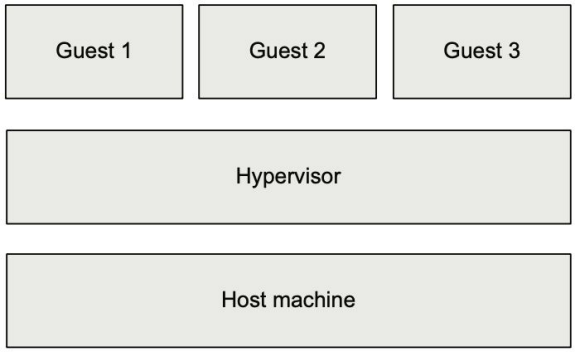
Accounts and Users

- **Users**
  - There is one root AWS account
  - You can attach multiple users to an account
    - With different privileges
    - To isolate different workloads
- **Be safe**
  - Never use root account to develop!
  - Always use 2FA!
  - Avoid my $40,000 mistake
- **Key pair**
  - To access a virtual server you need to create a key pair
  - Public key (in AWS and on virtual servers)
  - Private key is your secret
    - Don't lose it, you can't retrieve it



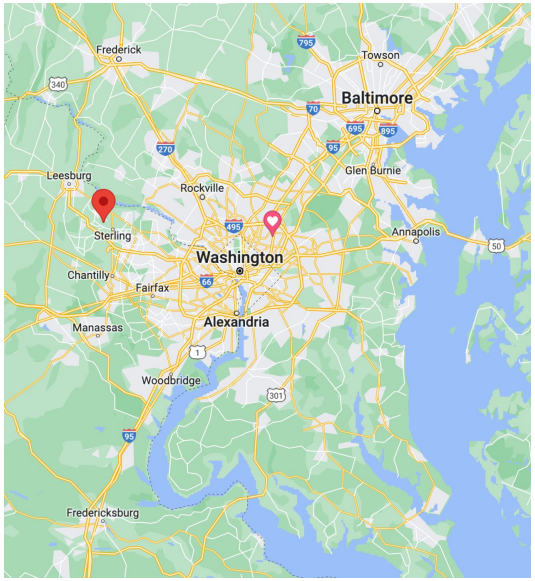
AWS VMs

- With virtualization multiple VMs can run on the same hardware
  - VMs can be started and stopped on-demand
- Physical server
  - Aka "host machine", "bare metal"
  - Each physical server consists of CPUs, memory, networking interfaces, and storage
- Hypervisor (software + CPU hardware)
  - Isolates guests
  - Schedules requests to the hardware
- Virtual servers (aka "guests") are isolated from each other on the same hardware
- AWS
  - Used to use Xen hypervisor (open-source)
  - Switched recently to AWS Nitro, hardware assisted virtualization
    - Performance close to bare metal



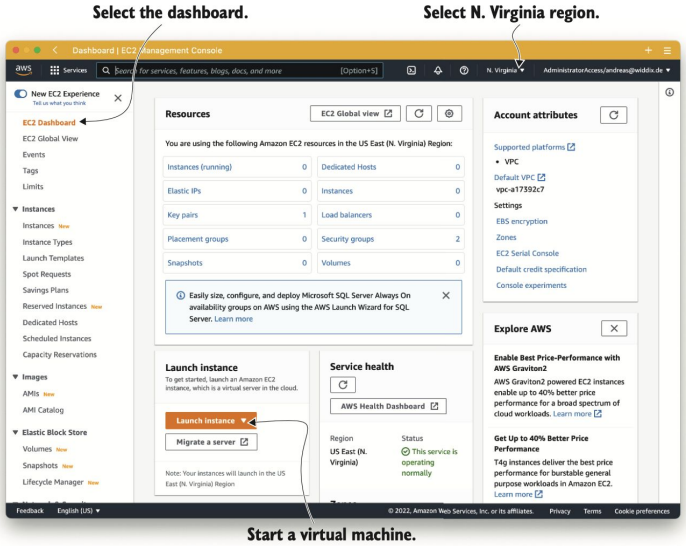
Starting an EC2 Instance

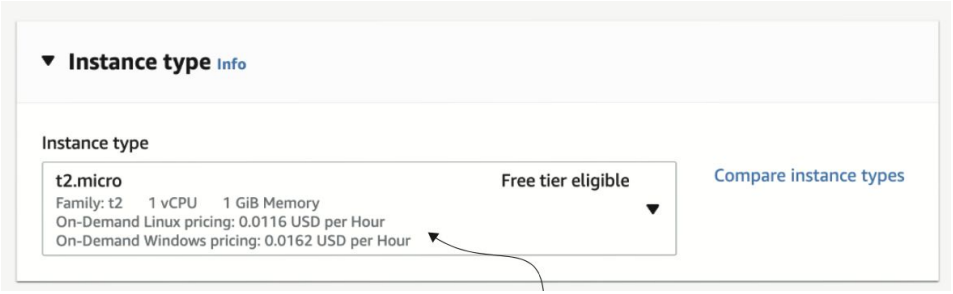
- **Select region**
  - E.g., **us-east-1**
  - Where is it?
    - 21155 Smith Switch Road, Ashburn, VA, USA

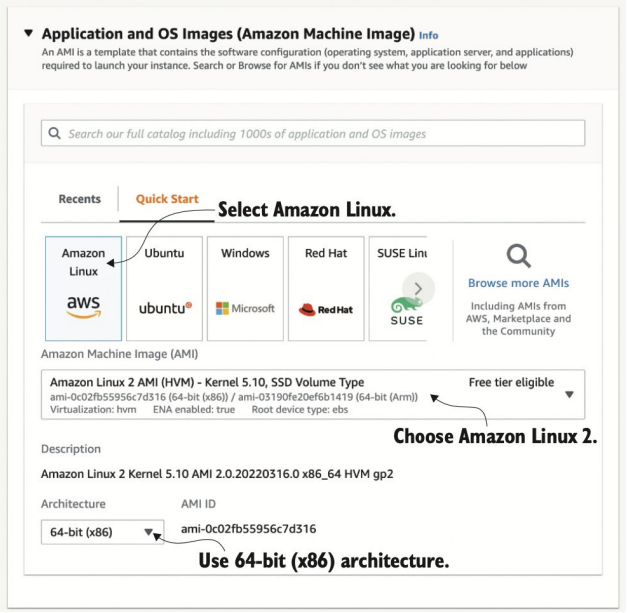


Starting an EC2 Instance

- **Select OS**
  - Amazon Machine Image (AMI)
  - Contains OS + pre-installed software
  - No need to spend (dev and machine) time installing OS, packages, software
- **Choose instance parameters**
  - E.g., `t2.micro`
  - More info later
- **Configure instance**
  - Network, shutdown behavior, termination protection, monitoring

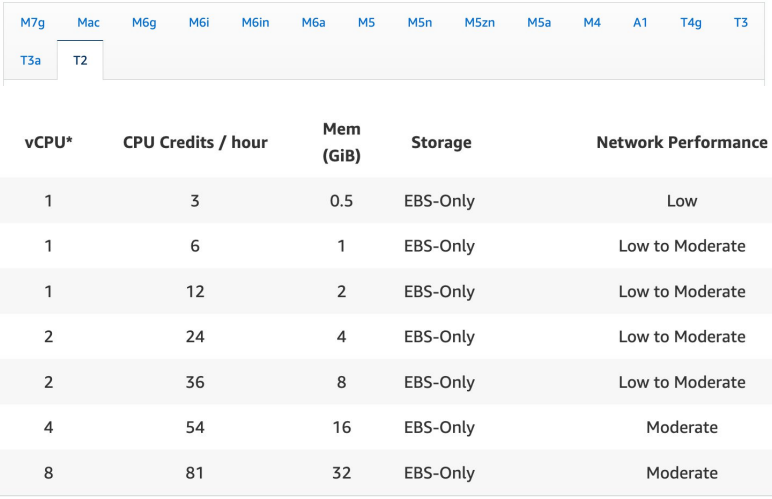






AWS Instance Type

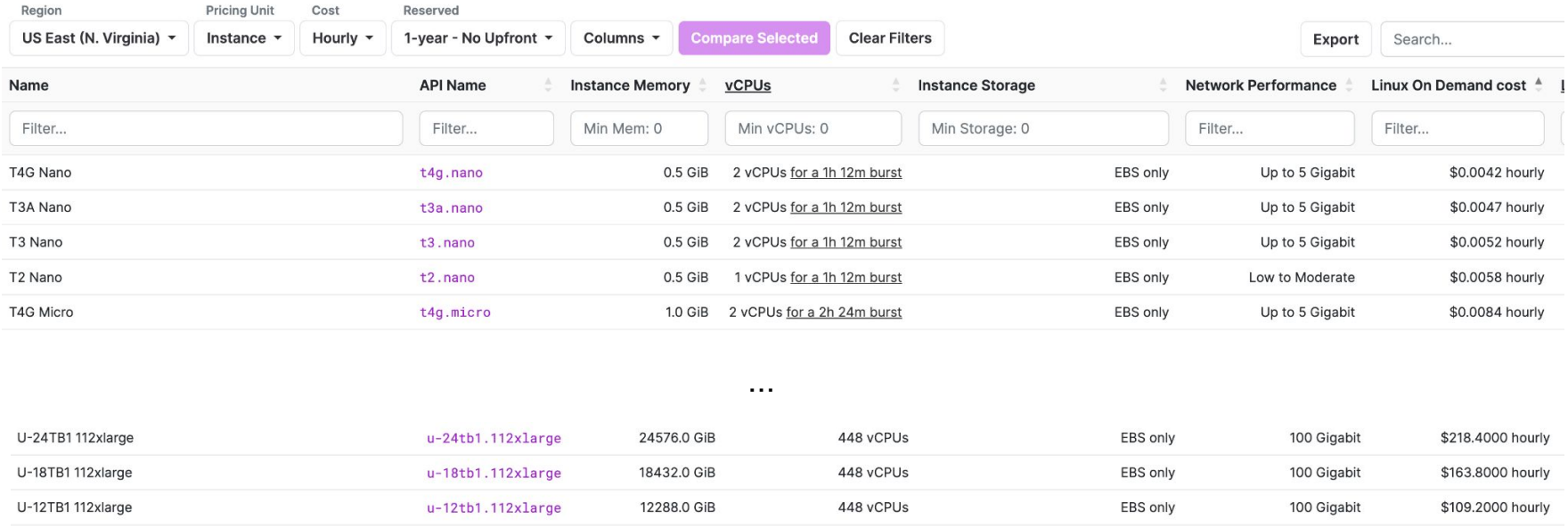
- An "instance type" describes the amount of computing power available
- https://aws.amazon.com/ec2/instance-types
- Instance family
  - T: cheap, baseline
  - M: general purpose
  - C: compute optimized
  - R: memory optimized
  - D: storage optimized for HDD
  - I: storage optimized for SSD
  - F: with FPGAs
  - P, G, CG: with GPUs
- E.g., **`t2.micro`**
  - `t`: instance family (small, cheap)
  - `2`: generation (second)
  - `micro`: size (1 vCPU, 1GB memory)
  - 0.013 USD / hr
- E.g., **`m4.large`**
  - `m`: general purpose
  - large (2 vCPUs, 8GB memory)
  - 0.14 USD / hr



Instance	vCPU*	CPU Credits / hour	Mem (GiB)	Storage	Network Performance
t2.nano	1	3	0.5	EBS-Only	Low
t2.micro	1	6	1	EBS-Only	Low to Moderate
t2.small	1	12	2	EBS-Only	Low to Moderate
t2.medium	2	24	4	EBS-Only	Low to Moderate
t2.large	2	36	8	EBS-Only	Low to Moderate
t2.xlarge	4	54	16	EBS-Only	Moderate
t2.2xlarge	8	81	32	EBS-Only	Moderate

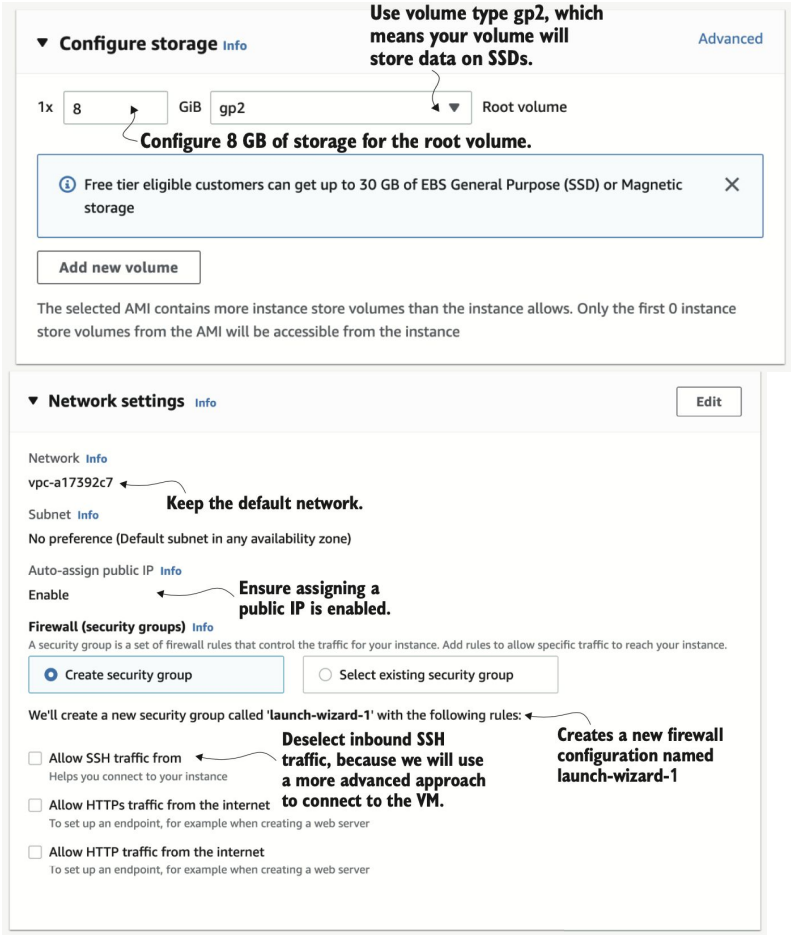
AWS Instance Type

- Price on AWS website is somehow unclear (duh really?)
  - Burst mode
  - vCPUs
  - Multi-tenancy
  - On-demand vs spot vs reserved vs pre-paid
  - 642 types of machines
  - From $37 / yr to $1.91M / yr (500 CPUs and 24TB of mem)
- Alternative sites: https://instances.vantage.sh



Starting an EC2 Instance

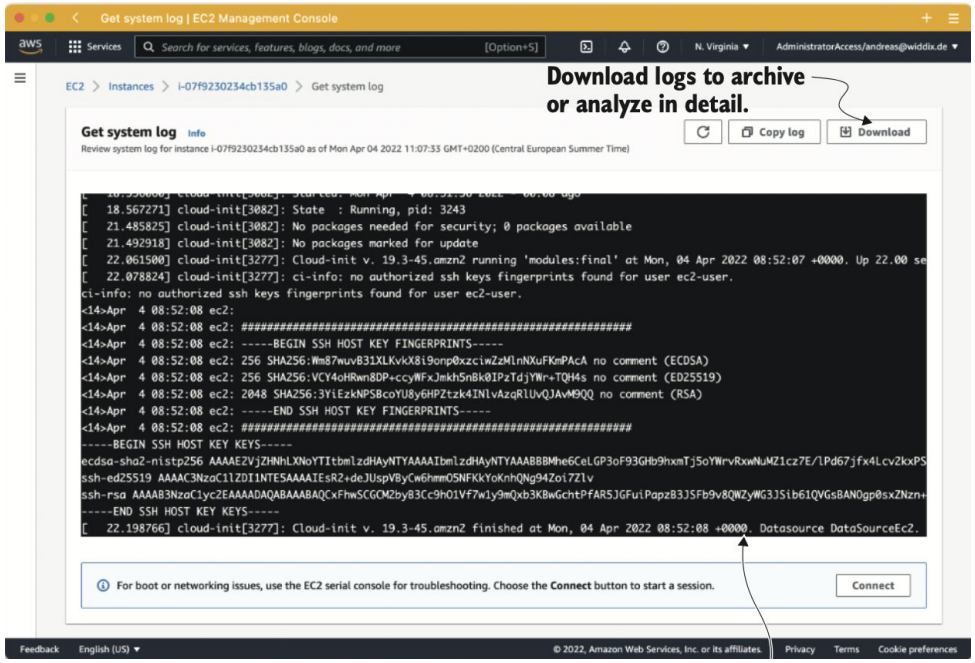
- Add storage
  - Volume size
  - Volume type (SSD or magnetic HDDs)
- Tag
- Configure firewall
  - How to access using SSH
  - Select key-pair
- How to monitor the instance
  - E.g. CloudWatch



Starting an EC2 Instance

- Start instance
- Find public IP
- Connect to the machine

```
> ssh -i $PATH/mykey.pem ubuntu@$PUBLIC_UP
> cat /proc/cpuinfo
> free -m
> sudo apt-get update
> sudo apt-get install ...
```



States of a VM

- **Start**
  - You can start a stopped VM
- **Stop**
  - A stopped VM is not billed
  - Attached resources (e.g., network HDD) will persist and still incurs in charges
  - Your data storage (local disk) will persist
  - The VM can be restarted later, but on a different host (with different IP)
- **Reboot**
  - Attached resources will persist
  - Your data storage will persist
  - All software still installed after a reboot
  - The VM restarts but on a different host
- **Terminate**
  - It means delete: you cannot restart
  - Data storage is wiped out
  - Attached volumes persist

Moving / Upgrading EC2 Instances

- **Scale up / down**
  - If you need more computing power you can increase the size of the VM
  - Stop the VM
  - Change instance type (e.g., m3.large)
  - Start the VM
  - Public and private IP addresses change
- **AWS regions**
  - Each region is a collection of data centers located in the same area
  - Regions are independent from each other
  - Data is not transferred between regions
  - Some AWS services (e.g., IAM, CDN, DNS) act globally
- **Why moving across AWS regions**
  - Different regions have different distance from your users
  - Compliance
    - Are you allowed to store and process data in that country?
  - Service availability
    - Some AWS services are not available in certain regions
  - Redundancy
  - Costs
    - Service costs vary by region
- **Often you want an Elastic IP address to get a fixed public IP**

Optimizing Costs

- **On-demand instances**
  - Maximum flexibility, no restrictions
  - You start and stop VMs when you want
  - Pay by hour
- **EC2 / Compute saving plans**
  - 1 yr vs 3 yrs
  - Commit to a certain number of hours
  - Payment options: all, partial, no upfront
  - Discount: up to 3x cheaper than on-demand price
  - Useful for dev servers
- **Capacity reservation**
  - Get machines even in peak hours
- **Spot instances**
  - Bid for unused capacity
  - Price based on supply / demand
  - Discount: up to 10x cheaper than on-demand price
  - Useful for running asynchronous tasks

Programming the Infrastructure

- On AWS *everything* can be controlled via an API over HTTPS
  - Start a VM
  - Create 1TB of storage
  - Start Hadoop cluster
- Jeff Bezos 2002 API mandate
  - HackerNews
  - An eye-witness from Google about it
- You can use:
  - AWS console
  - HTTP requests to the API
  - CLI (Command Line Interface)
    - Call AWS API from your terminal
  - SDK (Software Development Kit)
    - Call AWS API from your code
  - CloudFormation: templates that describe the state of the infrastructure and are translated into API calls

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols – doesn't matter.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.
7. Thank you; have a nice day!

Jeff Bezos 2002 API mandate

Infrastructure-As-Code

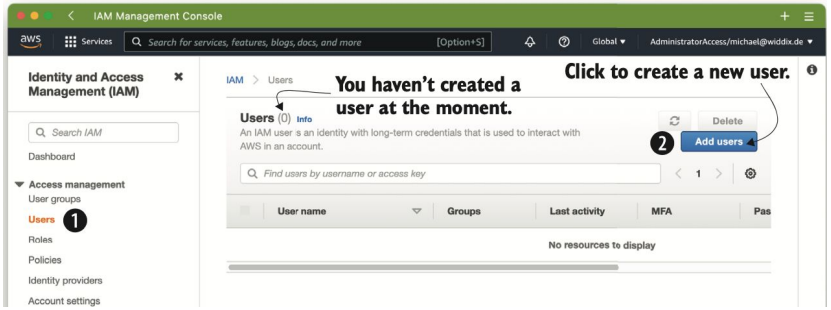
- Use high-level programming language to control IT systems
- You can apply software development to infra
  - Code repository
  - Automated tests
  - Continuous integration
- DevOps (SRE in Google parlance)
  - Mix devs and ops in the same team
  - Use software to bring development and operations closer together
  - Devs are responsible for operational tasks (e.g., being on-call)
  - Ops are involved in software development, making system easier to operate
  - Foster communication and collaboration

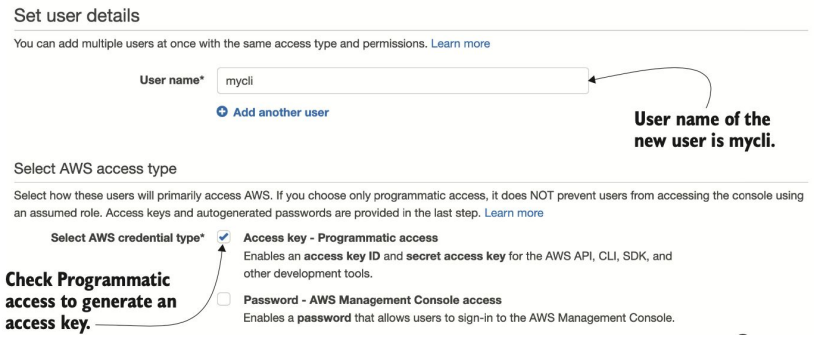
Infrastructure-as-Code: Advantages

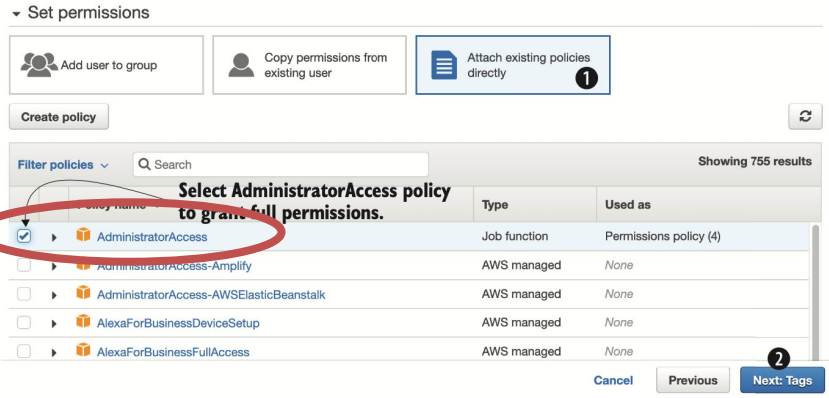
- **Save time**
  - Reusing scripts or ready-to-use blueprints
  - Automating tasks done regularly
  - Copy-paste vs click-click
- **Fewer mistakes**
  - Push button flow
- **Consistency of actions**
  - Multiple deploys per day
- **Deployment pipeline**
  - Commit to the repo
  - Source code is built
  - Automatic tests (e.g., integration tests)
  - Build testing environment
  - Run acceptance tests in isolation
  - Changes are propagated to production
  - Monitor production
- **The script is a detailed documentation**
  - It explains what and how, but not why (not a design document)

Create User Account

- Do not to use AWS root account for all development
- Create a new user
  - IAM = Identity and Access Management
- Access key ID + secret access key
- Access control
  - Enable programmatic access
  - Enable console access
  - Limit what an user can do through policies







Bad idea!

AWS Command Line Interface (CLI)

- Provide unified interface to all AWS services
- Output is in JSON format

  `> apt-get install awscli`
- Authenticate

  `> aws configure`
  - AWS access key ID
    - E.g., `AKIAIRUR3YLPOSVD7ZCA`
  - AWS secret access key
    - E.g., `SSKIng7jkAKERpcT3YphX4cD87sBYgWVw2enqBj7`
  - Default region name
    - E.g., `us-east-1`
- Execute command

  `> aws <service> <action> --key value`

```
$ aws ec2 describe-regions
{
  "Regions": [
    {
      "Endpoint": "ec2.eu-north-1.amazonaws.com",
      "RegionName": "eu-north-1",
      "OptInStatus": "opt-in-not-required"
    },
    [...]
    {
      "Endpoint": "ec2.us-west-2.amazonaws.com",
      "RegionName": "us-west-2",
      "OptInStatus": "opt-in-not-required"
    }
  ]
}
```

Software Development Kit (SDK)

- SDK is a library that calls into AWS API from your favorite programming language
  - E.g., Python, Go, Ruby, C++, JavaScript
- Pros: it handles
  - Authentication
  - Retry on error
  - HTTPs communication
  - XML / JSON de-/serialization
- Cons
  - Imperative approach
  - Need to deal with dependencies

```python
import boto3

# Get the service resource.
dynamodb = boto3.resource('dynamodb')

# Create the DynamoDB table.
table = dynamodb.create_table(
    TableName='users',
    KeySchema=[
        {
            'AttributeName': 'username',
            'KeyType': 'HASH'
        },
        {
            'AttributeName': 'last_name',
            'KeyType': 'RANGE'
        }
    ],
    AttributeDefinitions=[
        {
            'AttributeName': 'username',
            'AttributeType': 'S'
        },
        {
            'AttributeName': 'last_name',
            'AttributeType': 'S'
        },
    ],
    ProvisionedThroughput={
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    }
)

# Wait until the table exists.
table.wait_until_exists()

# Print out some data about the table.
print(table.item_count)
```

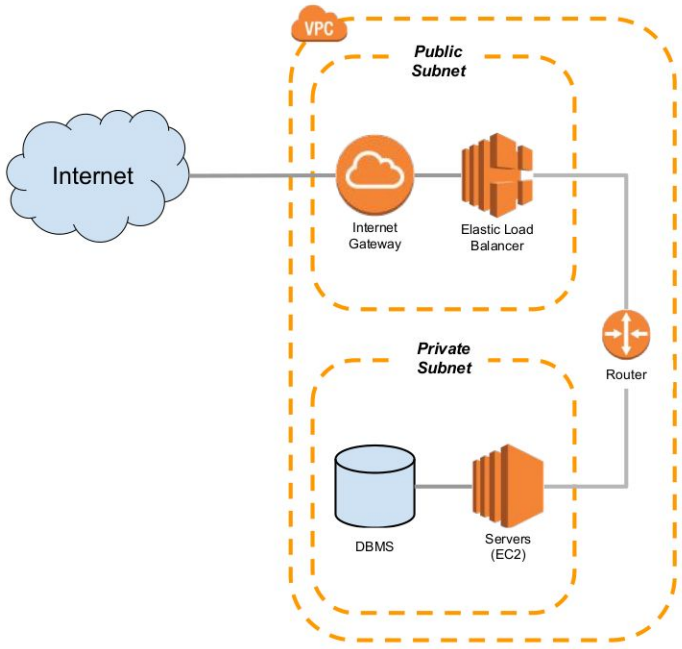
AWS CloudFormation

- **Use templates (e.g., JSON, YAML) to describe infrastructure**
- **Declarative vs imperative approach**
  - Declare the system, rather than listing the steps to build the system
- **AWS Stacks processes CloudFormation templates**
- **Pros**
  - Consistent way to describe the infrastructure
    - People implement same thing in a different way
  - Handle dependencies
  - Customizable
    - Inject custom parameters to customize templates
  - Testable
    - Create infra from a template, test, and shut down
  - Updatable
    - If you update the template, Stacks automatically applies changes to the infrastructure
  - It serves as documentation
    - Use it as code in a VCS

```
AWSTemplateFormatVersion: '2010-09-09'
Metadata:
  License: Apache-2.0
Description: 'AWS CloudFormation Sample Template DynamoDB_Table: This template demonstrates
  the creation of a DynamoDB table.  **WARNING** This template creates an Amazon DynamoDB
  table. You will be billed for the AWS resources used if you create a stack from
  this template.'
Parameters:
  HashKeyElementName:
    Description: HashType PrimaryKey Name
    Type: String
    AllowedPattern: '[a-zA-Z0-9]*'
    MinLength: '1'
    MaxLength: '2048'
    ConstraintDescription: must contain only alphanumberic characters
  HashKeyElementType:
    Description: HashType PrimaryKey Type
    Type: String
    Default: S
    AllowedPattern: '[S|N]'
    MinLength: '1'
    MaxLength: '1'
    ConstraintDescription: must be either S or N
  ReadCapacityUnits:
    Description: Provisioned read throughput
    Type: Number
    Default: '5'
    MinValue: '5'
    MaxValue: '10000'
    ConstraintDescription: must be between 5 and 10000
Resources:
  myDynamoDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
      - AttributeName: !Ref 'HashKeyElementName'
        AttributeType: !Ref 'HashKeyElementType'
      KeySchema:
      - AttributeName: !Ref 'HashKeyElementName'
        KeyType: HASH
      ProvisionedThroughput:
        ReadCapacityUnits: !Ref 'ReadCapacityUnits'
        WriteCapacityUnits: !Ref 'WriteCapacityUnits'
Outputs:
  TableName:
    Value: !Ref 'myDynamoDBTable'
    Description: Table name of the newly created DynamoDB table
```

Securing Your System

- **Always install software updates**
  - New security vulnerabilities are found and fixed every day
- **Restrict access to AWS account**
  - Multiple persons and scripts should use different AWS account
  - Principle of "least privilege": grant only permissions needed to perform a job
- **Restrict network traffic**
  - Leave open only the ports that you strictly need
    - E.g., 80 for HTTP traffic and 443 for HTTPS
  - Close everything else
  - Encrypt traffic (and data)
- **Create a private network**
  - Subnets that are not reachable from the Internet



AWS Shared-responsibility Principle

- AWS is responsible for:
  - Protect network through monitoring Internet access
    - E.g., prevent DDoS attacks
  - Ensuring physical security of data centers
  - Decommissioning storage devices after end of life
- You are responsible for:
  - Restrict access using IAM
  - Encrypting network traffic (e.g., HTTPS)
  - Configuring firewall for VPN
  - Encrypting data
  - Updating OS (kernel, libs) and software
- More info here