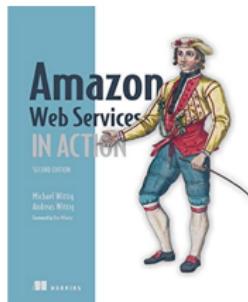




UMD DATA605 - Big Data Systems

11.2: AWS Overview

- **Instructor:** Dr. GP Saggese, gsaggese@umd.edu
- **References:**
 - Some basic info in the slides
 - Many tutorials on-line
 - Mastery
 - “Amazon Web Services in Action” 3rd Edition



Amazon Web Services (AWS)

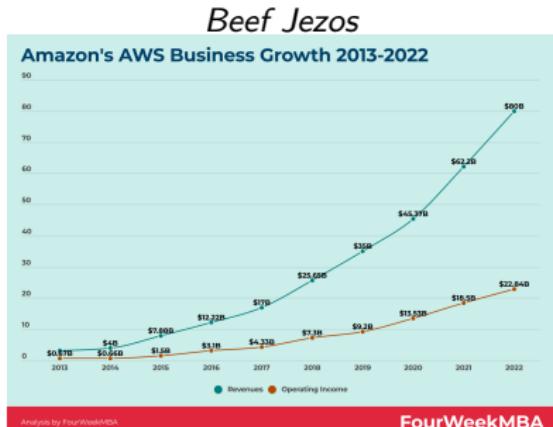
- AWS is a platform offering complete cloud solutions
 - Computing (e.g., EC2)
 - Storing (e.g., S3)
 - Networking
- Offers different levels of abstractions
 - IaaS, PaaS, SaaS
 - From virtual hardware to applications, e.g.,
 - Host websites
 - Run enterprise software
 - Run machine learning applications
- Control services in different ways
 - Web interface (console)
 - CLI: aws command
 - Language libraries, SDK (e.g., Python boto3)

AWS as Business

- Services charge pay-per-use pricing model
 - 500 new services and features annually
- Data centers distributed globally
 - US, Europe, Asia, South America
- Insanely profitable
 - \$91B/year in revenue (2023)
 - Grows 42% year-over-year
 - Controls 30% of cloud business



1998: "I sell books."
2017: "I sell whatever the f--- I want."



Types of Cloud Computing

- Cloud computing enables a **shared pool of configurable computing resources**
 - E.g., servers, storage, networks, applications, services
 - Accessible from anywhere
 - Convenient
 - On-demand
- Clouds can be:
 - *Public*: open to the general public (e.g., AWS)
 - *Private*: virtualize and share IT infrastructure within one organization (e.g., government)
 - *Hybrid*: mix of public and private clouds

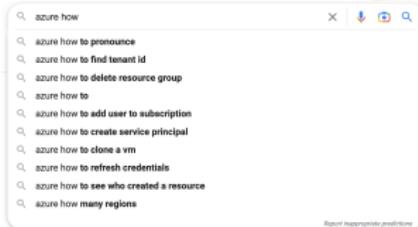
AWS vs Google Cloud vs Microsoft Azure

- **Similarities:**

- Worldwide infrastructure
- Provide IaaS (computing, networking, storage)
 - AWS EC2 / Google Compute Engine / Azure VMs
 - AWS S3 / Google Cloud Storage / Azure Blob Storage
- Pay-as-you-go pricing model

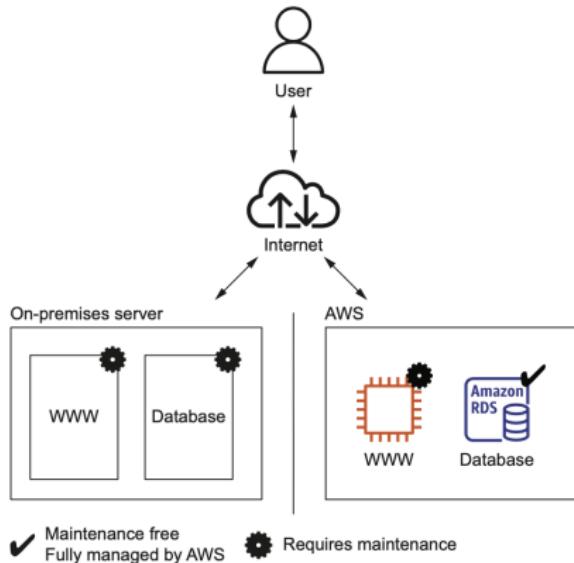
- **Differences:**

- AWS
 - Market leader, mature, powerful
 - Utilizes open source technologies
- Azure
 - Offers Microsoft stack in the cloud
- Google
 - Focuses on cloud-native applications



From On-premise to AWS: 1/3

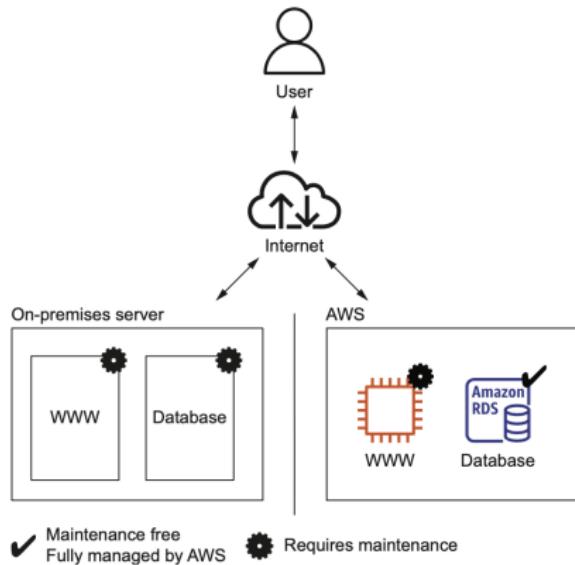
- Aka “Cloud-transformation”
- **Example:** move a medium-sized e-commerce site *from on-premise to the cloud*
- **No-cloud**
 - *Web-server*: handles customer requests
 - *DB*: stores product info and orders
 - *Static content* (e.g., JPEG images)
 - Delivered over CDN
 - Reduces load on services
 - *Dynamic content* (e.g., HTML pages)
 - E.g., products and prices
 - Delivered by web server



From On-premise to AWS: 2/3

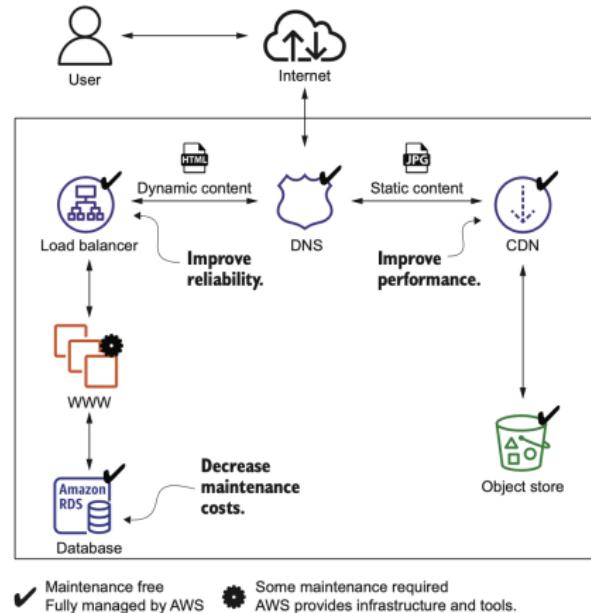
- **Move to cloud**

- Keep the same architecture
- Move components to cloud



From On-premise to AWS: 3/3

- Design for the cloud
 - DNS
 - Database
 - Object store (S3)
 - Managed solutions
 - Use multiple smaller virtual services with a load balancer to increase reliability



Low-Cost Processing

- Lots of batch jobs are not critical / run on a schedule
 - Analyze data daily
 - Generate reports from a database

1. **Allocate machines on demand**

- AWS bills VMs per minute
- Pay only when running jobs

2. **AWS Batch**

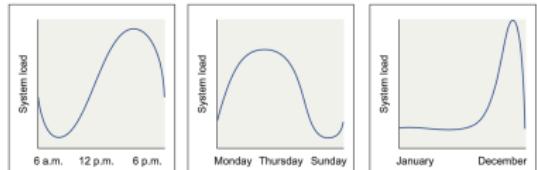
- Offers spare capacity at a discount
- Runs when capacity is available, saving 50%

3. **AWS Lambda**

- Serverless

Capacity Scaling

- **No need to plan for capacity**
 - Do not predict capacity
 - Schedule capacity on-the-fly
 - No concern for rack space, switches, power supplies
 - Add more VMs (1 to 1000) and storage (GB to PB) as needed
- **Handle seasonal traffic by scaling up/down**
 - Day vs night
 - Weekday vs weekend
 - Holidays
 - No test system needed when the team is off
- **Worldwide presence**
 - AWS has many data centers
 - Deploy applications close to 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 billed based on usage:
 - Hours of virtual server (rounded up)
 - Used storage in GB (allocated or real)
 - Data traffic in GB or requests
 - ...

- **Free tier**

- Use some AWS services free for 12 months after signing up
- Experiment with services
 - E.g., EC2, S3
- Exceed limits, start paying (without notice)
 - Set an alarm!

Service	January usage	February usage	February charge	Increase
Visits to website	100,000	500,000		
CDN	25 M requests + 25 GB traffic	125 M requests + 125 GB traffic	\$115.00	\$100.00
Static files	50 GB used storage	50 GB used storage	\$1.15	\$0.00
Load balancer	748 hours + 50 GB traffic	748 hours + 250 GB traffic	\$19.07	\$1.83
Web servers	1 virtual machine = 748 hours	4 virtual machines = 2,992 hours	\$200.46	\$150.35
Database (748 hours)	Small virtual machine + 20 GB storage	Large virtual machine + 20 GB storage	\$133.20	\$105.47
DNS	2 M requests	10 M requests	\$4.00	\$3.20
Total cost			\$472.88	\$360.65

Pay-per-use

- **Advantages**

- No upfront investments or commitment
- Lower project startup cost
- Easier to divide the system into smaller parts
 - One big server or two smaller ones cost the same
- Affordable fault tolerance/high performance
 - For a scalable workload “buy 1 server for 1000 hours” = “buy 1000 servers for 1 hour”



Interacting with AWS

• GUI (Management Console)

- Start interacting with services easily
- Set up cloud infrastructure for development and testing

• Command-line tool (CLI)

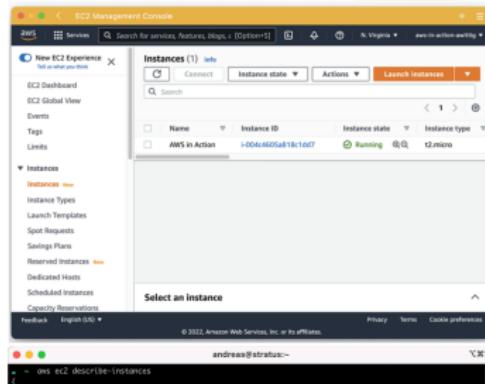
- Manage and access AWS services
- Automate recurring tasks

• SDKs

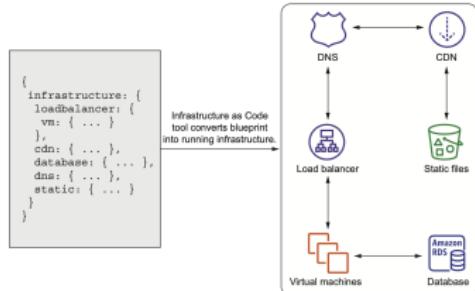
- Use libraries in any language to interact with AWS
- Integrate applications with AWS
- E.g., boto3 for Python

• Blueprints

- Describe your system with all services and dependencies
- Describe the system, not how to build it



```
andreas@stratus:~$ aws ec2 describe-instances
{
    "Reservations": [
        {
            "Group": "None",
            "Instances": [
                {
                    "AmiLaunchIndex": 0,
                    "ImageId": "ami-03291222c584d346",
                    "InstanceId": "i-004c6605a118c1a67",
                    "InstanceType": "t3.nano",
                    "LaunchTime": "2022-07-07T14:04:39+00:00",
                    "Monitoring": {
                        "State": "Disabled"
                    },
                    "Placement": {
                        "AvailabilityZone": "us-east-1a",
                        "Tenancy": "Default"
                    }
                }
            ]
        }
    ]
}
```



Accounts and Users

- **Users**

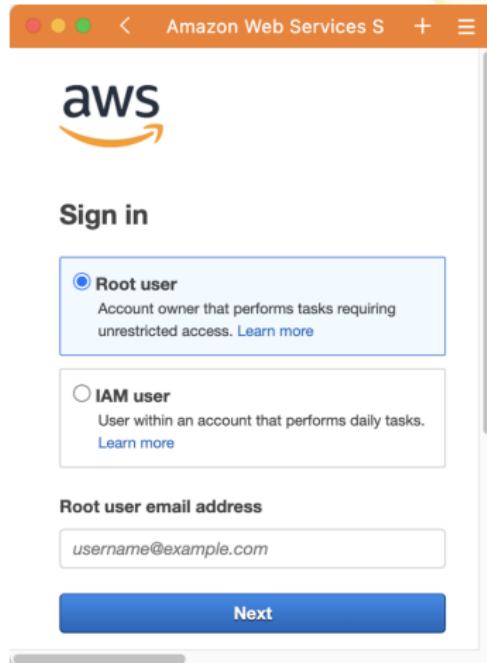
- One root AWS account
- Attach multiple users to an account
 - Different privileges
 - Isolate workloads

- **Be safe**

- Never use root account to develop
- Always use 2FA
- Avoid costly mistakes

- **Key pair**

- Create a key pair to access a virtual server
- Public key (in AWS and on virtual servers)
- Private key is your secret
 - Don't lose it; can't retrieve it



Create User Account

- **Create a new user**

- Avoid using AWS root account for development
- IAM: Identity and Access Management
- Access key ID + secret access key

- **Access control**

- Enable programmatic access
- Enable console access
- Limit user actions through policies

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

mycli

Add another user

User name of the new user is mycli.

Select AWS access type

Select how these users will primarily access AWS. If you choose only programmatic access, it does NOT prevent users from accessing the console using an assumed role. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Select AWS credential type*

Access key - Programmatic access

Enables an access key ID and secret access key for the AWS API, CLI, SDK, and other development tools.

Password - AWS Management Console access

Enables a password that allows users to sign-in to the AWS Management Console.

Check Programmatic access to generate an access key.

Click to create a new user.

You haven't created a user at the moment.

Add user

User name	Groups	Last activity	MFA

Set permissions

Add user to group

Copy permissions from existing user

Attach existing policies directly

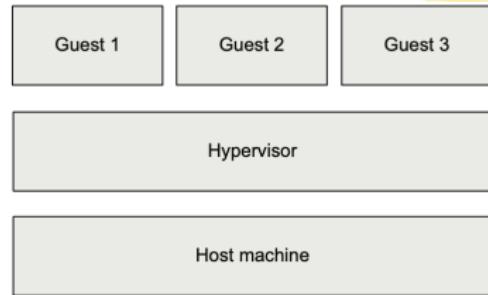
Create policy

Filter policies

Policy name	Search	Type	Used as
Select AdministratorAccess policy to grant full permissions.			
<input checked="" type="checkbox"/>	AdministratorAccess	Job function	Permissions policy (4)
<input type="checkbox"/>	AdministratorAccess-Amplify	AWS managed	None
<input type="checkbox"/>	AdministratorAccess-AWSElasticBeanstalk	AWS managed	None

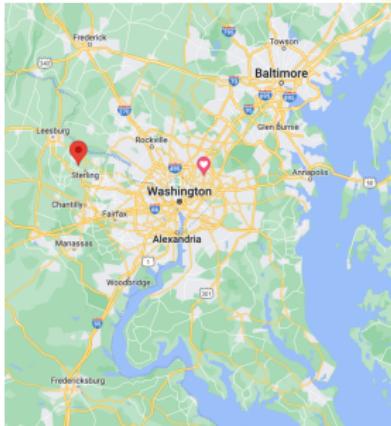
AWS VMs

- With virtualization
 - Multiple VMs run on the same hardware
 - Start and stop VMs on-demand
- **Physical server**
 - Aka “host machine”, “bare metal”
 - Consists of CPUs, memory, networking interfaces, storage
- **Hypervisor**
 - Software + CPU hardware
 - Isolates guests
 - Schedules hardware requests
- AWS
 - Before Xen hypervisor (open-source)
 - Switched to AWS Nitro
 - Hardware-assisted virtualization
 - Performance close to bare metal
- **Virtual servers**
 - Isolated on the same hardware



Starting an EC2 Instance

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



Starting an EC2 Instance

- **Select OS**

- Amazon Machine Image (AMI)
- Contains OS + pre-installed software
- Saves time installing OS, packages, software

- **Choose instance parameters**

- E.g., t2.micro
- More info later

- **Configure instance**

- Network, shutdown behavior, termination protection, monitoring

Select the dashboard.

Select N. Virginia region.

Instance type

t2.micro

Family t2 1 vCPU 1 GB Memory

On-Demand Linux pricing: 0.0176 USD per Hour

On-Demand Windows pricing: 0.0182 USD per Hour

Free tier eligible

Compare instance types

Application and OS Images (Amazon Machine Image)

Search our full catalog including 1000s of application and OS images

Quick Start

Select Amazon Linux.

Description

Amazon Linux 2 Kernel 5.10 AMI 2.0.20220316.0.x86_64 HVM gp2

Architecture

x86_64 (x86) ami-0c02fb55956c7d516

Virtualization

HVM

ENI enabled: true

Root device type: ebs

Free tier eligible

Choose Amazon Linux 2.

Use 64-bit (x86) architecture.

AWS Instance Type

- **Instance type** describes the computing power
- **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
- [AWS EC2 list](#)

M7g	M4	M6g	M1	M1n	M1a	M5	M5n	M5m	M5a	M4	A1	T4g	T5
T3a	T2												
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								

- 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: size (2 vCPUs, 8GB memory)
 - 0.14 USD/hr

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 EC2 machines (as of 2023)
 - Cheapest: 37 USD/yr
 - Most expensive: 1.91M USD/yr (500 CPUs and 24TB of memory)
- Alternative sites:
 - E.g., <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

The screenshot shows two configuration steps in the AWS EC2 instance creation wizard:

Configure storage (Info):
1x 8 GiB gp2 Root volume
Configure 8 GB of storage for the root volume.
Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage
Add new volume
The selected AMI contains more instance store volumes than the instance allows. Only the first 0 instance store volumes from the AMI will be accessible from the instance

Network settings (Info):
vpc-a17392c7 Keep the default network.
Subnet Info
No preference (Default subnet in any availability zone)
Auto-assign public IP Info Ensure assigning a public IP is enabled.
Firewall (security groups) Info A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.
Create security group (radio button selected) Select existing security group
We'll create a new security group called 'launch-wizard-1' with the following rules:
 Allow SSH traffic from Deselect inbound SSH traffic, because we will use Helps you connect to your Instance a more advanced approach to connect to the VM.
 Allow HTTPS traffic from the internet To set up an endpoint, for example when creating a web server
 Allow HTTP traffic from the internet To set up an endpoint, for example when creating a web server

Starting an EC2 Instance

- Start instance
- Find public IP

```
Get system log | EC2 Management Console
AWS Services Search for services, features, blogs, docs, and more [Consent] N. Virginia • AdministrationActions[admin@ec2-100-100-100-100.us-east-1.compute.amazonaws.com]
EC2 > Instances > i-0298230214b135e0 > Get system log
Download logs to archive or analyze in detail.
Copy log Download

Get system log info
Review system log for instance i-0298230214b135e0 as of Mon Apr 04 2022 11:07:33 GMT+0200 (Central European Summer Time)

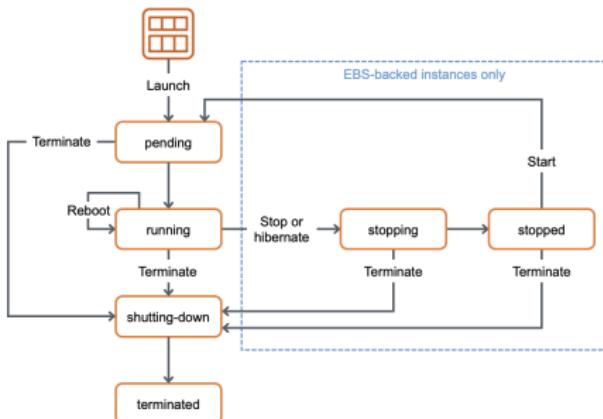
[18.04.2022/17:52:48+02:00] cloud-init[1342]: Start: / running, pid: 3303
[21.04.2022/17:52:48+02:00] cloud-init[1342]: No packages needed for security, 0 packages available
[22.04.2022/17:52:48+02:00] cloud-init[1342]: No packages needed for update
[22.04.2022/17:52:48+02:00] cloud-init[1342]: Cloud-init v. 19.3-45~amzn2 running 'modules-final' at Mon, 04 Apr 2022 08:52:07 +0000, ip 22.49.100.100
[22.04.2022/17:52:48+02:00] cloud-init[1342]: ci-info: no authorized ssh keys fingerprints found for user ec2-user.
[23.04.2022/17:52:48+02:00] cloud-init[1342]: 1 authorized ssh keys fingerprints found for user ec2-user.
[24.04.2022/17:52:48+02:00] cloud-init[1342]: BEGIN SSH HOST KEY FINGERPRINTS -----
[24.04.2022/17:52:48+02:00] cloud-init[1342]: 4 48:32:00+02:00 256 SHA256:VCh4t0WnB0DccyRj3eb0d9b40BD7d1Nr-TDh4 no comment (RSA)
[24.04.2022/17:52:48+02:00] cloud-init[1342]: 4 48:32:00+02:00 256 SHA256:3V1zNP3kcr0fIy46P7zxh4ZkNvAaUk1UvJwAMwQ0 no comment (RSA)
[24.04.2022/17:52:48+02:00] cloud-init[1342]: END SSH HOST KEY FINGERPRINTS -----
[24.04.2022/17:52:48+02:00] cloud-init[1342]: BEGIN SSH HOST KEY EXCS -----
[24.04.2022/17:52:48+02:00] cloud-init[1342]: 4 48:32:00+02:00 256 AAAE2V3Y3D4KLWnT1T1m1zduMjY7MA23t1wduNjNTYAAA889WeCz1Gf3cGP3f093Q4bRuxt75oWrxvewuR21cz7E/1p8d7f4Lcv2kxPS
[24.04.2022/17:52:48+02:00] cloud-init[1342]: END SSH HOST KEY EXCS -----
[24.04.2022/17:52:48+02:00] cloud-init[1342]: 22.11697663 cloud-init[1342]: Cloud-init v. 19.3-45~amzn2 finished at Mon, 04 Apr 2022 08:52:08 +0000. Datasource DataSourceOfEc2.
```

- 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**
 - Start stopped VM
- **Stop**
 - Stopped VM not billed
 - Network HDD persist and incur charges
 - Local disk does not persist
 - VM can restart on a different host (different IP)
- **Reboot**
 - Network HDD persist
 - Local disk does not persist
 - Software remains installed
 - VM restarts on a different host
- **Terminate**
 - Delete: cannot restart
 - Network HDD persist
 - Local disk wiped out



Moving / Upgrading EC2 Instances

- **Scale up / down**

- Increase VM size for more computing power
- Stop VM
- Change instance type (e.g., m3.large)
- Start VM
- IP addresses change

- **AWS regions**

- Regions are collections of data centers
- Regions are independent
- No data transfer across regions
- Some services (e.g., IAM, CDN, DNS) are global

- **Why move across AWS regions**

- Proximity to users
- Compliance
 - Data storage and processing permissions
- Service availability
 - Some services unavailable in certain regions
- Redundancy
- Costs
 - Vary by region

Optimizing Costs

- **On-demand instances**
 - Maximum flexibility, no restrictions
 - Start and stop VMs anytime
 - 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
 - Useful for dev servers
- **Capacity reservation**
 - Access machines even in peak hours
- **Spot instances**
 - Bid for unused capacity
 - Price based on supply/demand
 - Discount: up to 10x cheaper than on-demand
 - Useful for asynchronous tasks

Programming the Infrastructure

- On AWS everything can be **controlled via an API**
 - Start a VM
 - Create 1TB of storage
 - Start Hadoop cluster
 - ...
 - You can use:
 - AWS GUI 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 describe infrastructure state, translated into API
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**
 - An email worth \$100B/yr



Infrastructure-as-Code

- Use high-level **programming language to control IT systems**
- Apply **software development principles to infrastructure**
 - Code repository
 - Automated tests
 - Continuous integration
- **DevOps (SRE in Google parlance)**
 - Mix developers and operations in the same team
 - Use software to bring development and operations closer
 - Switch roles to experience each other's pain
 - Devs → responsible for operational tasks (e.g., on-call)
 - Ops → involved in software development, making system easier to operate
 - Foster communication and collaboration

Infrastructure-as-Code: Advantages

- **Save time**

- Reuse scripts or blueprints
- Automate tasks
- Copy-paste vs click-click-click

- **Fewer mistakes**

- Push-button flow

- **Consistency of actions**

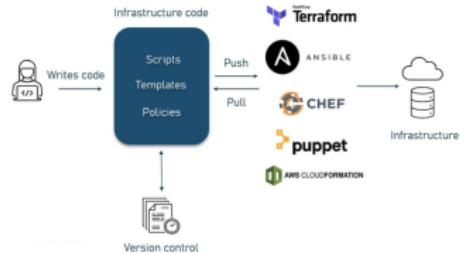
- Multiple deployments per day

- **Deployment pipeline**

- Commit changes to source code
- Build application from source
- Automatic tests (e.g., integration tests)
- Build testing environment
- Run acceptance tests in isolation
- Propagate changes to production

- **The script is detailed documentation**

- Explains what and how, not why



Name	Languages	Function	Approach	Infrastructure type
Terraform	HCL + TypeScript, Python, Java, C#, Go with CDK	Provisioning	Declarative	Immutable
AWS CloudFormation	JSON, YAML + TypeScript, Python, Java, .NET, and Go with CDK	Provisioning	Declarative	Both
Ansible	Python, Ruby, YAML	Configuration management	Imperative	Mutable
Puppet	PuppetDSL, YAML	Configuration management	Declarative	Mutable
CHEF	Ruby	Configuration management	Declarative	Mutable



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., AKIAIR...D7ZCA
- AWS secret access key
 - E.g., SSKIng7jkA...enqBj7
- Default region name
 - E.g., us-east-1

- Execute command

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

Software Development Kit (SDK)

- SDK is a library calling AWS API **from your programming language**
 - E.g., Python, Go, C++, JavaScript
- **Pros:** handles
 - Authentication
 - Retry on error
 - HTTPS communication
 - XML / JSON de-/serialization
- **Cons**
 - Imperative approach
 - Deal with dependencies

```
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()

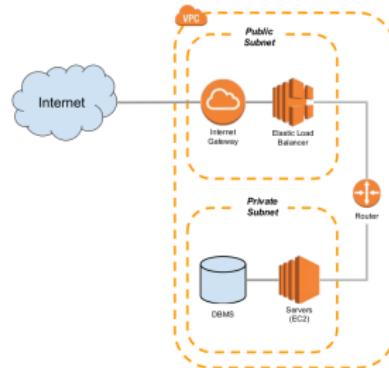
# Print out some data about the table.
print(table.item_count)
```

AWS CloudFormation

- **Use templates to describe infrastructure**
 - E.g., JSON, YAML
- **Declarative vs imperative approach**
 - Declare the system, not the steps to build it
- **AWS Stacks** processes CloudFormation templates
- **Pros**
 - Consistent infrastructure description
 - Handle dependencies
 - Customizable
 - Inject parameters to customize templates
 - Testable
 - Create infra from a template, test, shut down
 - Updatable
 - Update template, Stacks applies changes automatically
 - Serves as documentation
 - Use as code in source control

Securing Your System

- **Always install software updates**
 - Fix security vulnerabilities daily
- **Restrict access to AWS account**
 - Use different AWS accounts for individuals and scripts
 - Apply “least privilege”: grant necessary permissions only
- **Restrict network traffic**
 - Open essential ports
 - E.g., 80 for HTTP, 443 for HTTPS
 - Close others
 - Encrypt traffic and data
- **Create a private network**
 - Use subnets not reachable from the Internet



AWS Shared-responsibility Principle

- **AWS is responsible for:**
 - Protecting network by monitoring Internet access
 - Preventing DDoS attacks
 - Ensuring physical security of data centers
 - Decommissioning storage devices after end of life
- **You are responsible for:**
 - Restricting access using IAM
 - Encrypting network traffic (e.g., HTTPS)
 - Configuring firewall for VPN
 - Encrypting data
 - Updating OS and software