

DevOps (Development + Operations)

Tejas Parikh t.parikh@northeastern.edu

CSYE 6225
Northeastern University



What is DevOps?

- DevOps is a new term that primarily focuses on improved collaboration, communication, and integration between software developers and IT operations.
- It's an umbrella term that some describe as a philosophy, cultural change and paradigm shift.



Climate impact of a data center should take into consideration resource utilization and energy efficiency, in addition to power mix. Carbon emissions are driven by three items: the number of servers running, the total energy required to power each server, and the carbon intensity of energy sources used to power these servers. Using fewer servers and powering them more efficiently are at least as important to reducing the carbon impact of a company's data center as its power mix.

AWS customers use
77% fewer servers
84% less power
utilize a 28% cleaner power mix

Why DevOps?

for a total reduction in carbon emissions of 88% from using the AWS Cloud instead of operating their own data centers.

A photograph of a two-story house engulfed in intense orange and yellow flames. Several firefighters are visible in the foreground, one holding a yellow hose. The scene is set at night or dusk, with a fire truck partially visible in the background.

**WORKED FINE IN
DEV**

OPS PROBLEM NOW

Why DevOps?

- A business must become increasingly agile to support accelerated innovation and rapidly evolving customer needs.
- Time to market is key.
- IT must become agile to facilitate business needs.
- IT operations must be able to deploy applications in a consistent, repeatable and reliable manner. This can only be achieved with the adoption of automation.
- On-premises vs cloud
- Self-hosted vs SaaS



SDLC & Challenges with Self Hosted Applications

- Cannot auto push updates to applications deployed in customer's environment. Must wait for their IT to update the application.
- Cannot deliver feature and bug fixes to application end users until IT deploys the updates.
- Vendor and end-users are at mercy of IT team's availability.
- Bad UX for end users and potential revenue risk for you, the vendor.
- ROI for the customers can be low due to infrastructure cost.



Benefits of SaaS

- Better ROI for your customers
- Company manages the infrastructure
- Multi-tenant SaaS environments cost significantly less. Savings are usually passed on to the customers.
- Multiple deployments a day. Bug fixes and new features are being constantly deployed without impacting end users.

SaaS stands for Software as a Service. It's a cloud-based software delivery model where applications are hosted by a third-party provider and made available to users over the internet. Users access the software through web browsers or APIs, eliminating the need for installation, maintenance, and infrastructure management on the user's end. SaaS typically operates on a subscription basis, with users paying a recurring fee for access to the software and services. Popular examples of SaaS include Google Workspace, Salesforce, and Dropbox.

A multi-tenant SaaS (Software as a Service) platform is a software application that serves multiple customers, or "tenants," from a single shared instance of the software.

Single-Tenant SaaS:

1. **Salesforce.com (Enterprise Edition)**: Salesforce.com offers a single-tenant option within its Enterprise Edition, providing dedicated resources and control to larger organizations for specific security and compliance needs.
2. **Adobe Experience Manager (AEM) Sites**: AEM Sites offers both single-tenant and multi-tenant deployment options. In single-tenant mode, organizations get dedicated resources and full customization.

Multi-Tenant SaaS:

1. **Google Workspace (formerly G Suite)**: Google Workspace provides a suite of productivity tools like Gmail and Google Drive as a multi-tenant SaaS platform, ensuring data isolation while multiple organizations share infrastructure.
2. **Microsoft Office 365**: Office 365 delivers productivity tools like Outlook and Teams as a multi-tenant SaaS platform, allowing organizations to share infrastructure while maintaining data separation.

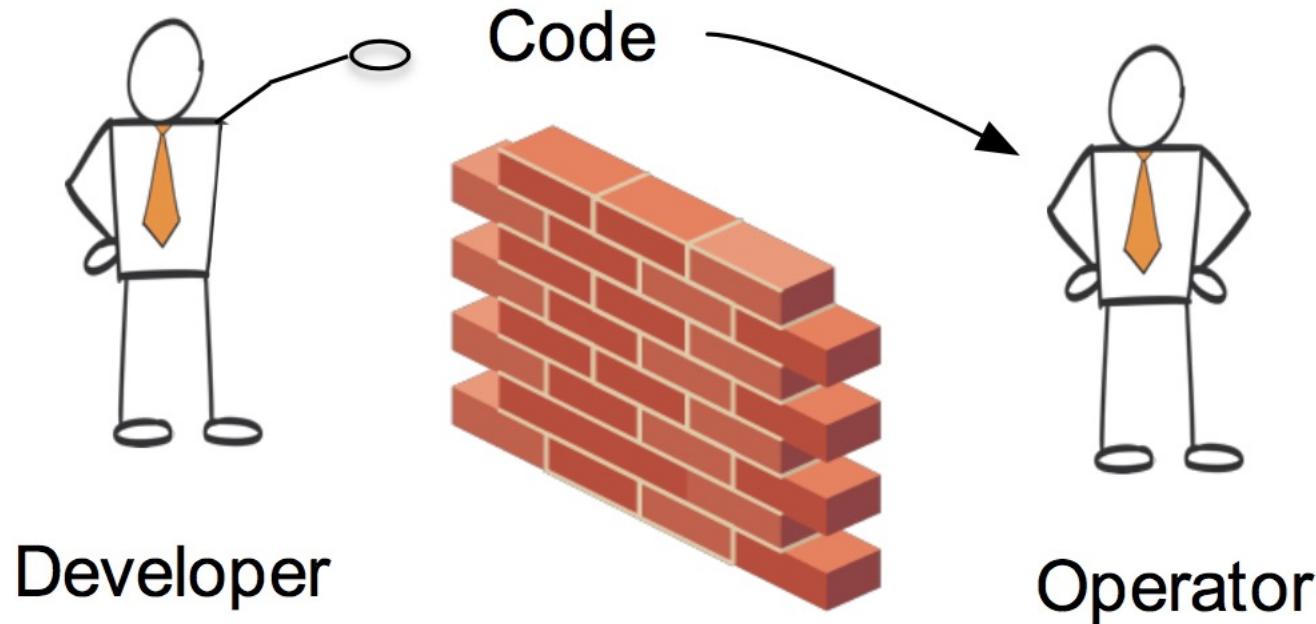


The Cloud

- Very low barrier to entry easier to adopt cloud
- Easy to start with minimal capital
- No hardware to buy; no upfront capital expenses
- Validate ideas quickly at relatively low cost
- Scale with users
- Control your infrastructure costs



Traditional Deployment Model



DevOps Workflow



<https://resources.github.com/devops/fundamentals/>



DevOps Practices

DevOps Practices

- Communication & Collaboration
 - Infrastructure as Code
 - Continuous Integration, Continuous Delivery, Continuous Deployment
 - Version Control
 - Automation
 - Logging and Monitoring
 - Security
- Continuous Integration: Developers merge code changes frequently, triggering automated builds and tests.
- Continuous Delivery: Continuous delivery is a software development practice where code changes are automatically built, tested, and prepared for a release to production but not deployed without some manual intervention
- Continuous Deployment: Every successful code change automatically goes through the pipeline (build & test & deployment) and is deployed to production without manual intervention.



Communication & Collaboration

- The key premise behind DevOps is collaboration.
- Development and operations teams coalesce into a functional team that communicates, shares feedback, and collaborates throughout the entire development and deployment cycle.



Infrastructure as Code

- Traditionally, infrastructure is provisioned using manual process.
- A fundamental principle of DevOps is to treat infrastructure the same way developers treat code.
- Practicing "Infrastructure as Code" means applying the same rigor of application code development to infrastructure provisioning and setup.
- All infrastructure provisioning "code" and environment configuration must be stored in version management system such as Git.
- Same programming best practices must apply to the infrastructure code as applied to application code.
- Infrastructure provisioning, orchestration, and deployment should support the use of "infrastructure code".

Infrastructure orchestration refers to the automated management and coordination of various infrastructure components, such as servers, networks, and storage, to support the deployment and operation of applications and services.



Automation

- Another core philosophy and practice of DevOps is automation.
- Automation focuses on the setup, configuration, deployment, and support of infrastructure and the applications that run on it.
- By using automation, you can set up environments more rapidly in a standardized and repeatable manner. The removal of manual process is a key to a successful DevOps strategy.
- Historically, server configuration and application deployment has been a predominantly a manual process. Environments become nonstandard, and reproducing an environment when issue arises is difficult.

most issues -> configuration errors

Ex: windows config deployed on linux
which will not run on linux



Benefits of Automation

- Rapid Changes
- Improved Productivity
- Repeatable Configurations
- Leveraged Elasticity
- Leveraged auto scaling
- Automated Testing



To make error is human. To propagate error to all server in automatic way is #devops.



A shirtless man with a beard and mustache is shown from the chest up, standing in a dense jungle. He has a weary or sad expression, looking slightly off-camera to his left. He is leaning his right arm against a large, dark tree trunk. His left hand is partially visible, holding a small, yellowish object. The background is filled with green foliage and trees.

WAITING FOR DEPLOYMENT

WITHOUT DEVOPS

Continuous Integration

- Continuous integration is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run.
- The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Continuous Delivery doesn't mean Continuous Deployment

Good practice to start with:

Continuous Delivery first, then Continuous Deployment & Continuous Integration



Continuous Delivery & Continuous Deployment

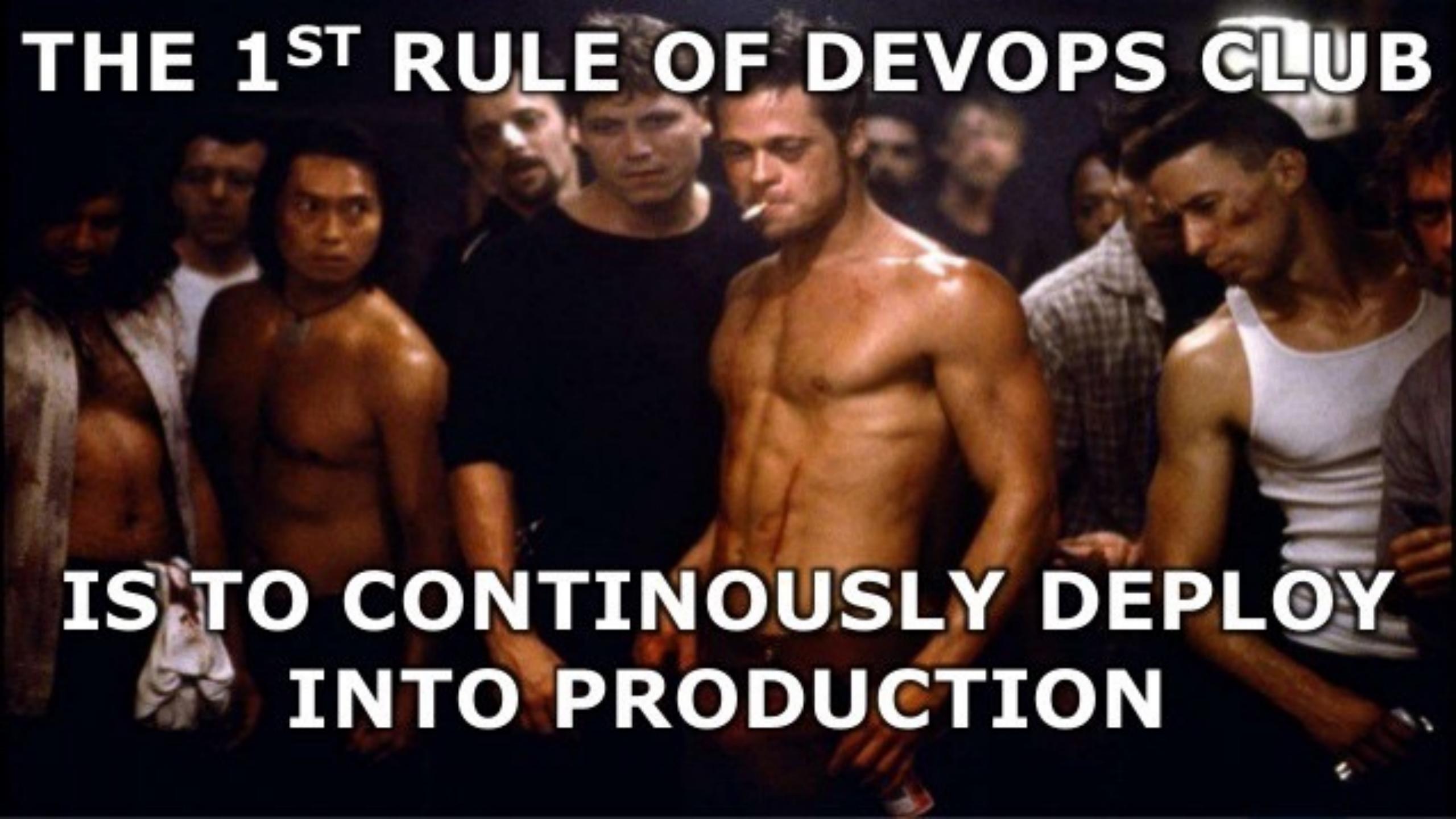
- Continuous delivery is a software development practice where code changes are automatically built, tested, and prepared for a release to production but not deployed without some manual intervention.
- Continuous deployment is a software development practice where code changes are automatically built, tested, and deployed to production.



In Continuous deployment: we can pinpoint exactly where an issue occurred, if there is frequent deployments.



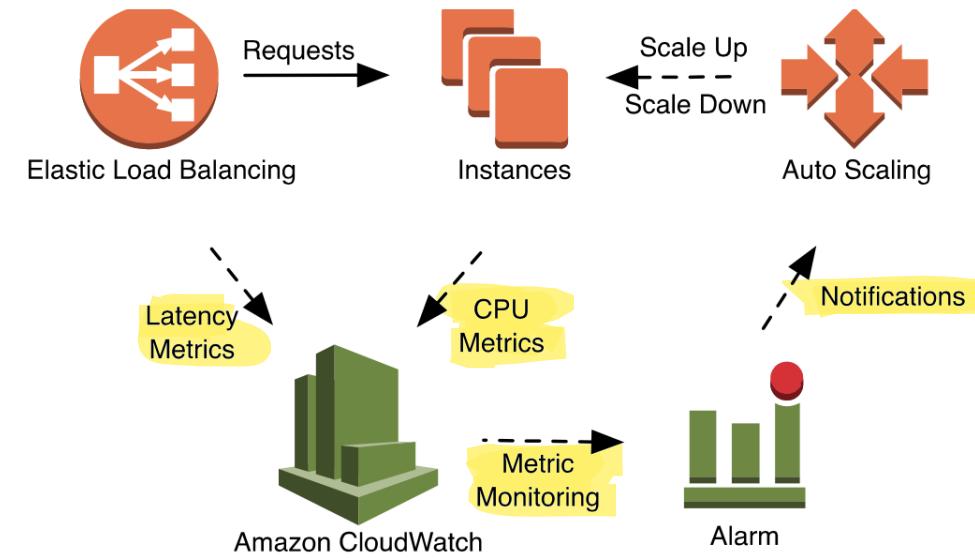
THE 1ST RULE OF DEVOPS CLUB

A black and white photograph of a group of shirtless men. In the center, a man with a cigarette in his mouth looks directly at the camera. He is surrounded by other men, some with their shirts off and others partially or fully clothed. The lighting is dramatic, creating strong shadows and highlights on their skin.

IS TO CONTINUOUSLY DEPLOY
INTO PRODUCTION

Monitoring

- Communication and collaboration is fundamental in a DevOps strategy.
- To facilitate this, feedback is critical.
- Feedback comes from logs, monitoring, alerting and auditing infrastructure so developers and operations teams can work together closely and transparently.



Security

what are honeypod servers

-> virtual server

-> we can see how hackers can modify code rather on a real server

- In a DevOps enabled environment, focus on security is still of paramount importance.
- Infrastructure and company assets needs to be protected, and when issue arise, they need to be rapidly and effectively addressed.



Summary

In order to make the journey to the cloud smooth, efficient and effective, technology companies should embrace DevOps principles and practices.



Additional Resources

See Lecture Page

NORTHEASTERN •
NORTHEASTERN •

Fundamentals of Cloud Computing

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University

I WAS HOPING FOR
A SLIGHTLY MORE DETAILED
EXPLANATION OF HOW
CLOUD COMPUTING WORKS
THAN - "IT'S MAGIC"!



On-Premise

Servers are acquired, operating systems are installed, other hardware may be involved, but all of that lives within your four walls, or the walls of your datacenter.

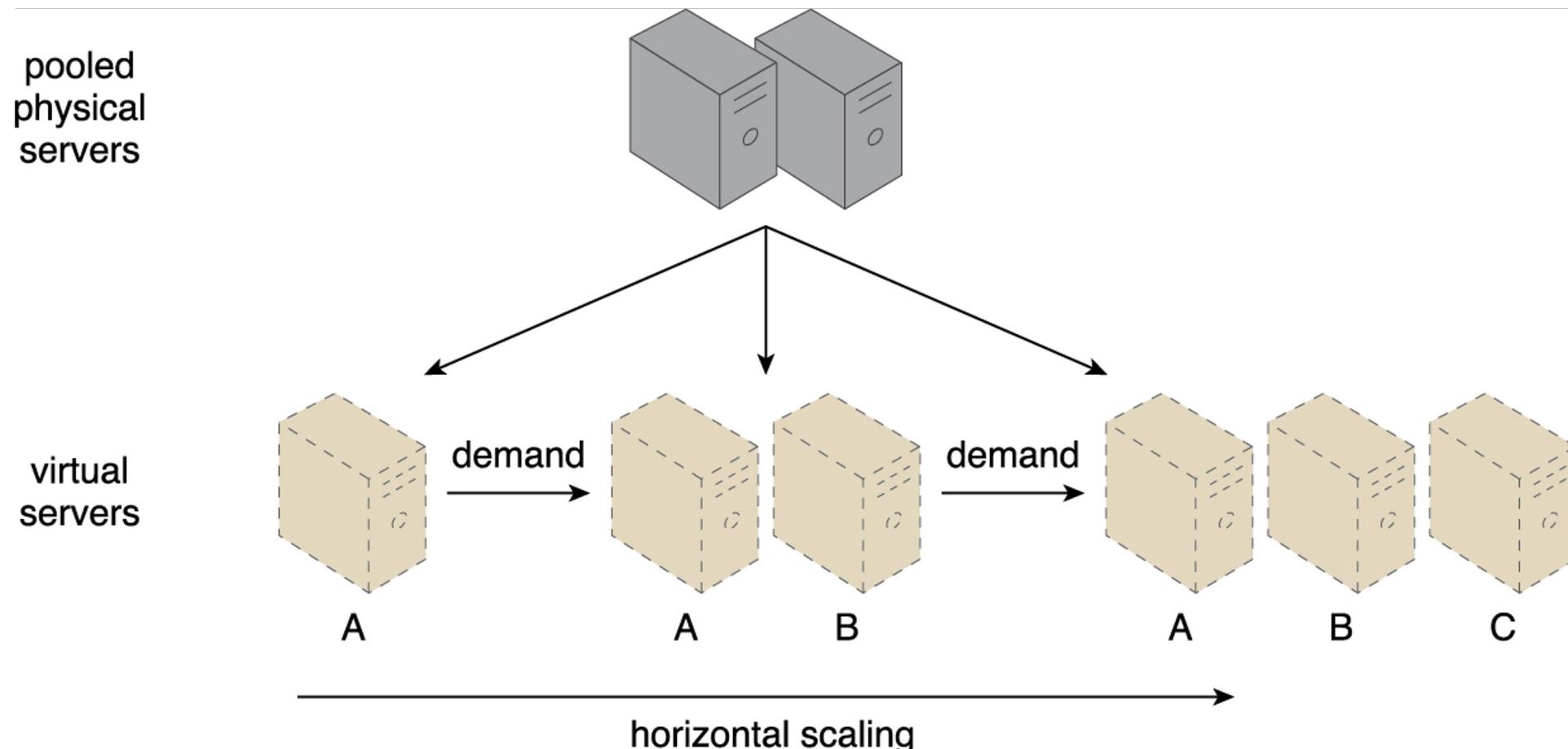
[upgrades on cloud is upon you](#)

Scaling

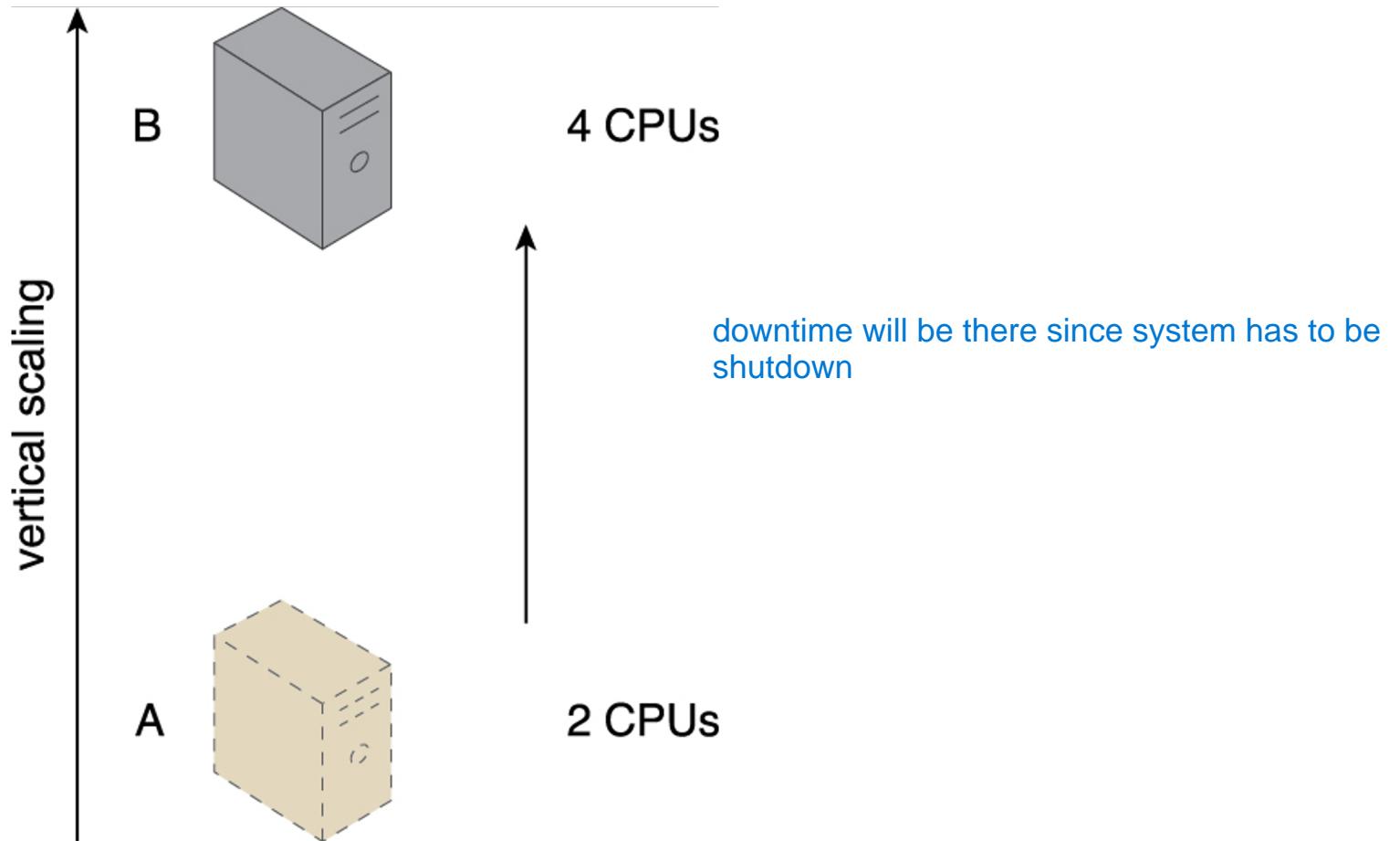
Scaling represents the ability of the resource to handle increased or decreased usage demands. The following are types of scaling:

- Horizontal Scaling – scaling out and scaling in
- Vertical Scaling – scaling up and scaling down

Horizontal Scaling



Vertical Scaling



Business Drivers

- Capacity Planning
- Cost Reduction
- Organizational Agility

Business Driver - Capacity Planning

- Capacity planning is the process of determining and fulfilling future demands of an organization's IT resources, products and services.
- A discrepancy between the capacity of an IT resource and its demand can result in a system becoming either inefficient (over-provisioning) or unable to fulfill user needs (under-provisioning).
- Different capacity planning strategies:
 - Lead Strategy – adding capacity to an IT resource in anticipation of demand
 - Lag Strategy – adding capacity when the IT resource reaches its full capacity lag, match & jit doesn't always work
 - Match Strategy – adding IT resources capacity in small increments, as demand increases JIT strategy
- Planning for capacity can be challenging because it requires estimating usage load fluctuations.
- Peak capacity is only rarely used and, consequently, average server utilization levels are often under 20%. Since few users deliberately provision for less than the expected peak, resources are idle at nonpeak times. The more pronounced the variation, the more the waste.

Business Driver – Cost Reduction

- A direct alignment between IT costs and business performance can be difficult to maintain.
- Two costs need to be accounted for
 - The cost of acquiring new infrastructure
 - Cost of its ongoing ownership
- Convert capital expenses to operating expenses (CapEx to OpEx).
- Operational overhead represents a considerable share of IT budgets, often exceeding up-front investment costs.
- Absence of up-front CapEx allows capital to be redirected to core business investment.
 - companies keep buying new laptops -> so that laptop as an asset inc the value of the company, an old laptop will keep depreciating the value of the company
 - ensure that high value assets > liability

Business Driver – Organizational Agility

- Businesses need the ability to adapt and evolve to successfully face change caused by internal and external factors.
- Organizational agility is the measure of an organization's responsiveness to change.
- Up-front investments and infrastructure ownership costs that are required to enable new or expanded business solutions may themselves be prohibitive.

What is Cloud Computing?



Definition of Cloud Computing

According to **Gartner** report Cloud Computing is:

“... a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service to external customers using Internet technologies.”

Definition of Cloud Computing

Forrester Research defines cloud computing as:

“... a standardized IT capability (services, software, or infrastructure) delivered via Internet technologies in a pay-per-use, self-service way.”

Definition of Cloud Computing

National Institute of Standards and Technology (NIST) defines Cloud Computing as:

.. a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.”

Benefits of Cloud Computing

- Reduce upfront cost of hardware, software and ownership
- Reduce or eliminate need for data center, cooling, backup datacenter & operations related cost
- On-demand access to pay-as-you-go computing resources
- Access to virtually unlimited computing resources
- Access to different kind of computing resources
- Flexibility in scaling resources almost instantaneously
- Increased Availability & Reliability

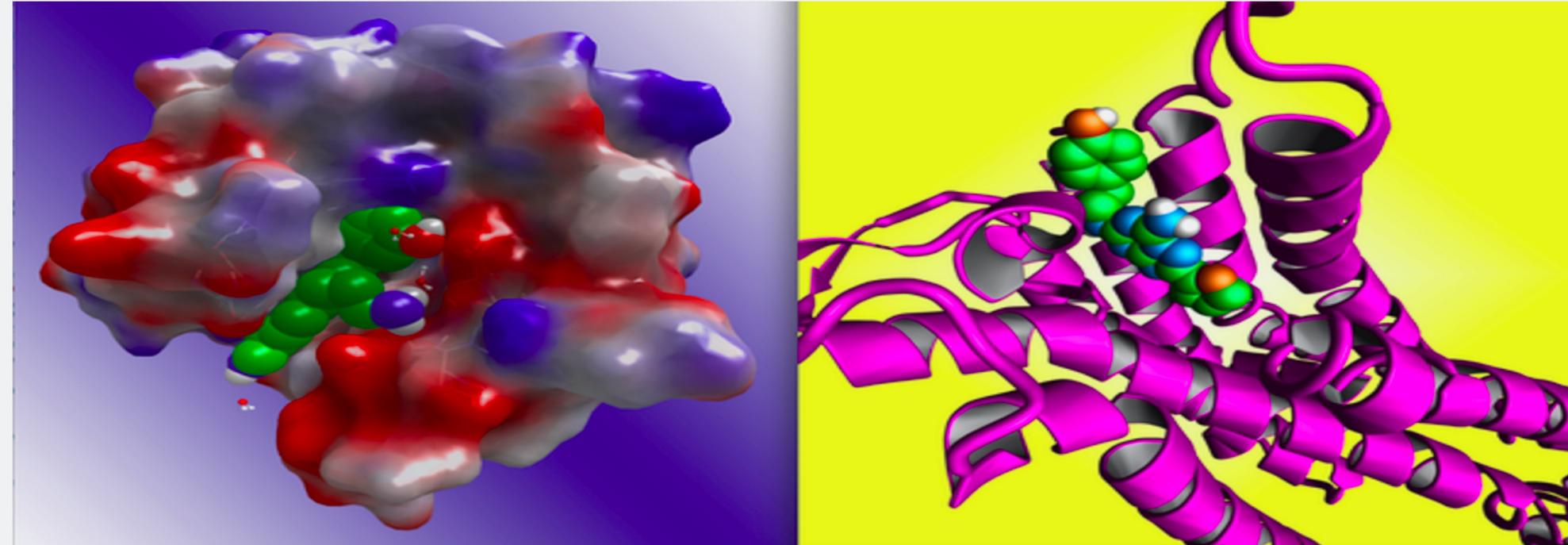


MINISTRY OF INNOVATION —

\$4,829-per-hour supercomputer built on Amazon cloud to fuel cancer research

A 50,000-core supercomputer deployed on Amazon shows the cloud's potential

JON BRODKIN - 4/19/2012, 9:00 AM



<https://arstechnica.com/business/2012/04/4829-per-hour-supercomputer-built-on-amazon-cloud-to-fuel-cancer-research/>

Obstacles for Cloud Computing

obstacles on paper, but not in real life

- Reduced Control over operations.
- Cloud Provider (Vendor) lock-in. Applications & Solutions may become vendor specific over time and make it difficult to migrate to different cloud provider or to on-prem.
- Reduce control over service configuration. Cloud services consumers can no longer fine tune parameters for services such as databases, etc.
- Data Confidentiality & Auditability. Regional compliance and legal issues can make it challenging to move application to cloud if the cloud provider does not meet the requirements. For e.g. HIPPA regulations.
- Cloud consumers usually have no recourse when cloud platform suffers outage.
binpacking?? - fit as much in a space and inc optimization
on prem - add more resources than actually required
cloud - we dont know which customer will get most resource -
did not understand this concept
- Performance Unpredictability
- Software Licensing
example migrating oracle db2 instances to cloud along with software lisence

Salesforce Outage: Can Customers Trust The Cloud?

Salesforce experienced an outage and service disruption to the NA14 instance, sending customers to Twitter to complain and organizations to evaluate the best way to work with cloud software providers.

AWS outage knocks Amazon, Netflix, Tinder and IMDb in MEGA data collapse

Cloudopocalypse stalks Sunday sofa surfers



BUSINESS
INSIDER

ENTERPRISE

Google apologizes for cloud outage that one person describes as a 'comedy of errors'

PayPal, 2009

In 2009, PayPal experienced a worldwide system outage for approximately five hours. The company handled \$2,000 in online commerce every second at the time, suggesting the event interfered with \$36 million worth of personal and business transactions.

Amazon is currently experiencing a degradation. They are [working on it](#).

reddit is down.



Follow

Pinterest is currently unavailable due to server outages. Our goal is to be back up by 10:30PM PST. Thanks for your patience!

Reply Retweet Favorite

10:07 PM - 29 Jun 12 via Mobile Web · Embed this Tweet

Azure status



We're having issues.

But we're working on it...



The sky is falling! Amazon's cloud seems to be down (raining?) so we're experiencing some issues too. Be back soon!

5 hours ago via web

Retweeted by [RealAmandaStone](#) and others



Amazon's massive AWS outage was caused by human error

One incorrect command and the whole internet suffers.

BY JASON DEL REY | @DELREY | MAR 2, 2017, 2:20PM EST



<https://www.recode.net/2017/3/2/14792636/amazon-aws-internet-outage-cause-human-error-incorrect-command>

Cloud Characteristics

For an IT environment to be considered cloud, it must meet following 5 essential characteristics:

1. On-demand self service
2. Broad Network Access
3. Resource Pooling (multitenancy)
4. Elasticity
5. Measured Service



Cloud Characteristics - On-demand self service

- A cloud consumer can unilaterally access cloud-based IT resources giving the cloud consumer the freedom to self-provision these IT resources.
- Once configured, usage of the self-provisioned IT resources can be automated, requiring no further human involvement by the cloud consumer or cloud provider. This results in an on-demand usage environment.

Cloud Characteristics - Broad Network Access

- Ability for a cloud service to be widely accessible.
- Establishing ubiquitous access for a cloud service can require support for a range of devices, transport protocols, interfaces, and security technologies.

Cloud Characteristics - Resource Pooling (multitenancy)

- The characteristic of a software program that enables an instance of the program to serve different consumers (tenants) whereby each is isolated from the other, is referred to as multitenancy.
- A cloud provider pools its IT resources to serve multiple consumers by using multitenancy models that frequently rely on the use of virtualization technologies.
- Through the use of multitenancy technology, IT resources can be dynamically assigned and reassigned, according to cloud service consumer demands.

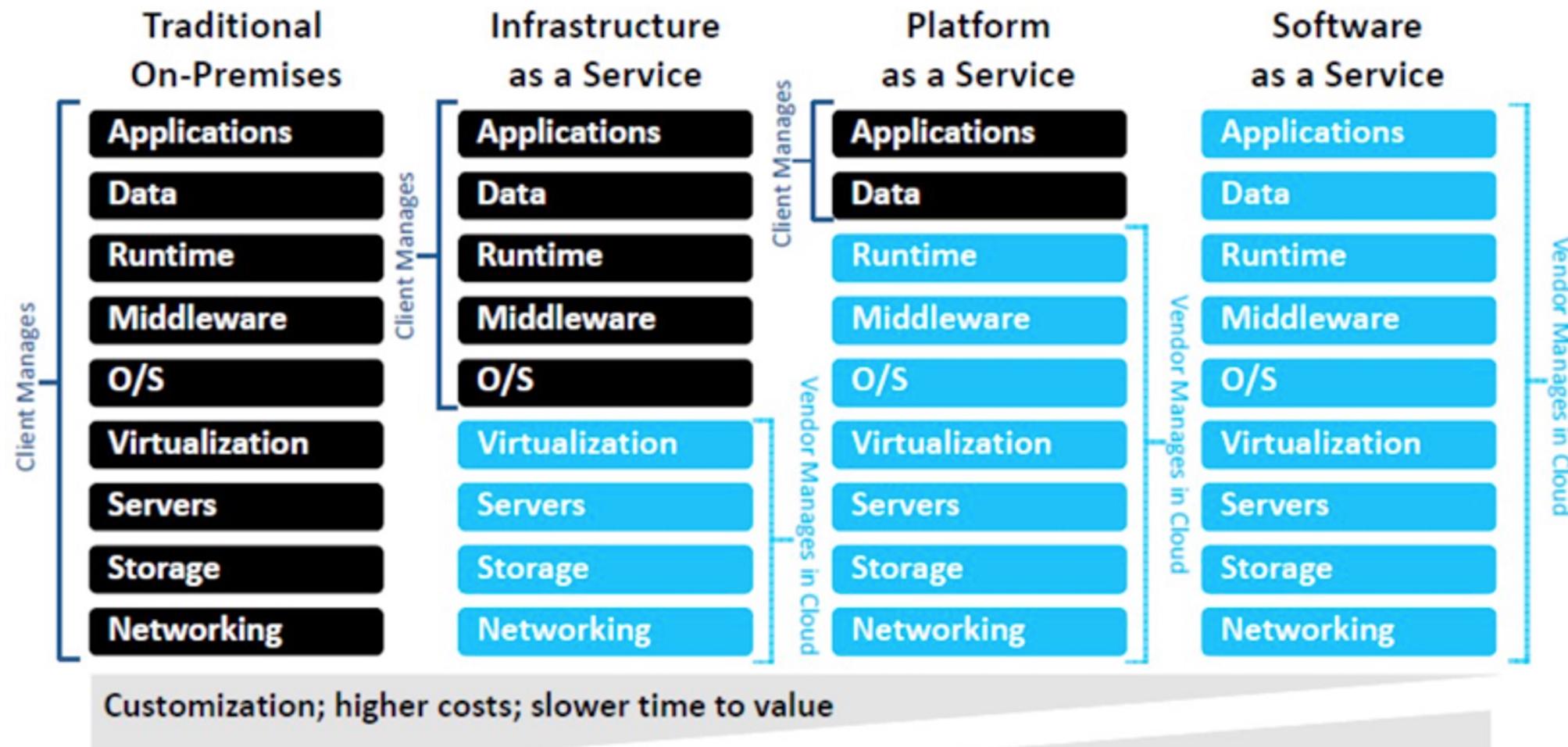
Cloud Characteristics - Elasticity

- Elasticity is the automated ability of a cloud to transparently scale IT resources, as required in response to runtime conditions or as pre-determined by the cloud consumer or cloud provider.
- Elasticity is often considered a core justification for the adoption of the cloud computing, primarily due to the fact that it is closely associated with reduced investment and proportional cost benefit.

Cloud Characteristics – Measured Usage

- The measured usage characteristic represents the ability of cloud platform to keep track of the usage of its IT resources, primarily by cloud consumers.
- Based on what is measured, the cloud provider can charge a cloud consumer only for the IT resources actually used and/or for the timeframe during which access to the IT resources was granted.

Cloud Delivery Models



Cloud Delivery Models



Infrastructure-as-a-Service (IaaS)

- IaaS provides self-contained IT environment consisting of hardware, networking, connectivity, Operating Systems and other “raw” IT resources.
- IaaS provides cloud consumers with a high level of control and responsibility over its configuration and utilization.
- IaaS providers may offer customized physical hardware for servers, load balancers, Gateways, firewalls, Intrusion Detection System (IDS) and Intrusion Prevention Systems.
- With IaaS, cloud consumer is responsible for securing the “raw” IT resources.
- Popular IaaS providers are Amazon Web Services, Google Cloud Platform, Microsoft Azure, IBM SoftLayer, Rackspace

Platform as a Service (PaaS)

- PaaS provides “ready-to-use” environment to deploy applications.
- Cloud consumers do not have control over or manage the underlying cloud infrastructure such as network, servers, operating systems, storage, etc.
- Popular PaaS providers are Heroku, IBM BlueMix and Google App Engine

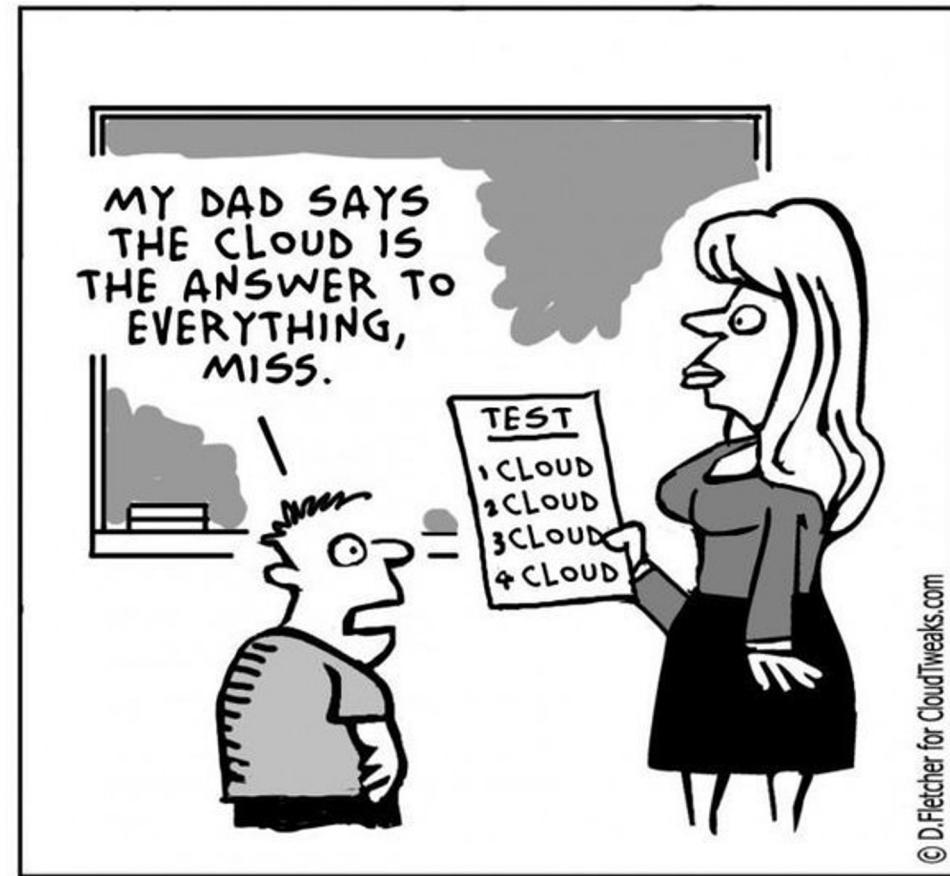
Software as a Service (SaaS)

- A software program positioned as a shared cloud service and made available as a “product” or generic utility represents the typical profile of a SaaS offering.
- Consumer of PaaS does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user specific application configuration settings.
- Popular SaaS offering examples are Gmail, GitHub, Google Drive, etc.

Cloud Deployment Models

There are four common cloud deployment models:

1. Public cloud
2. Community cloud
3. Private cloud
4. Hybrid cloud



Public Cloud

- Cloud infrastructure is provisioned for open use by the general public.
- Cloud provider is responsible for the creation of on-going maintenance of public cloud and its IT resources

Community Cloud

- A community cloud is similar to a public cloud except that its access is limited to a specific community of cloud consumers.
- Example of Community cloud is AWS GovCloud.
- AWS GovCloud (US) is an isolated AWS region designed to host sensitive data and regulated workloads in the cloud, helping customers support their U.S. government compliance requirements, including the International Traffic in Arms Regulations (ITAR) and Federal Risk and Authorization Management Program (FedRAMP). AWS GovCloud (US) is operated by employees who are vetted "U.S. Persons" and root account holders of AWS accounts must confirm they are U.S. Persons before being granted access credentials to the region.

Private Cloud

- The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units).
- Private clouds enable an organization to use cloud computing technology as a means of centralizing access to IT resources by different parts, locations, or departments of the organization.
- Private cloud mitigate some of the risks of public cloud as organization has much higher level of control.
- Even though private cloud resources might be hosted and controlled by the organization, it should not be confused with on-prem.

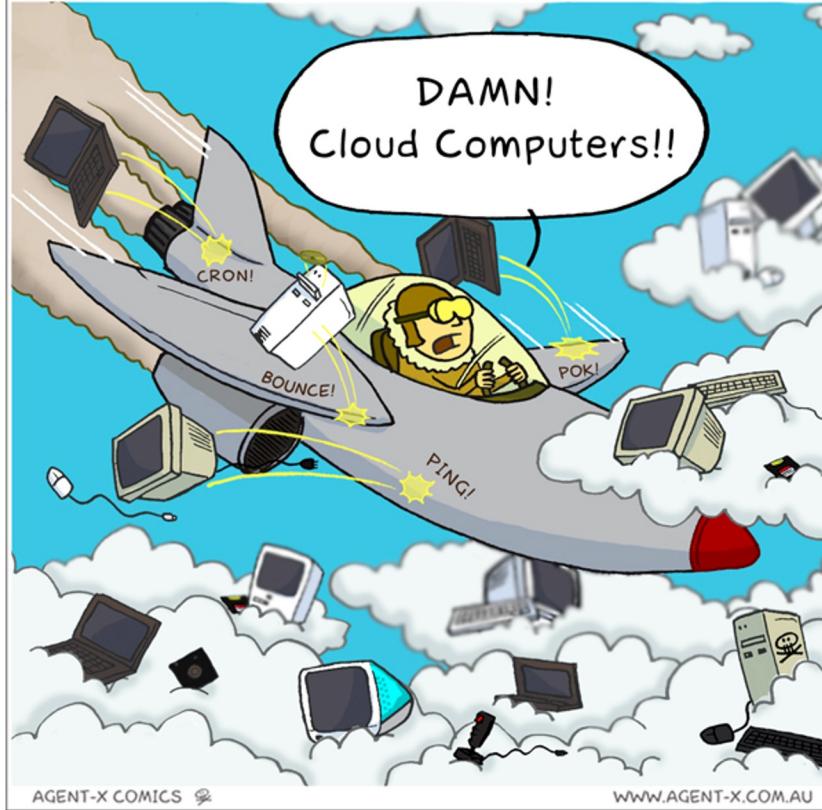
Hybrid Cloud

- A hybrid cloud is a cloud environment comprised of two or more different cloud deployment models.
- For example, a cloud consumer may choose to deploy cloud services processing sensitive data to a private cloud and other, less sensitive cloud services to a public cloud.

New Aspects in Cloud Computing from Hardware Provisioning & Pricing Point of View

- The appearance of infinite computing resources available on demand, quickly enough to follow load surges, thereby eliminating the need for cloud computing users to plan far ahead for provisioning.
- No commitments. Cloud users do not pay upfront fees allowing users to start small and increase resources only when there is an increase in their needs.
- The ability to pay for use of computing resources on a short-term basis as needed.

sla -> never 100% - reasonable 99.95 or greater, but sla always takes upper hand



Additional Resources

See Lecture Page

Identity & Access Management (IAM)

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University

Identity and Access Management (IAM)

According to Gartner

Identity and Access Management (IAM) is the security discipline that enables the right individuals to access the right resources at the right times for the right reasons.

Identity and Access Management (IAM)

- AWS Identity and Access Management (IAM) enables you to securely control access to AWS services and resources for your users.
- Using IAM, you can create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources.

IAM Features

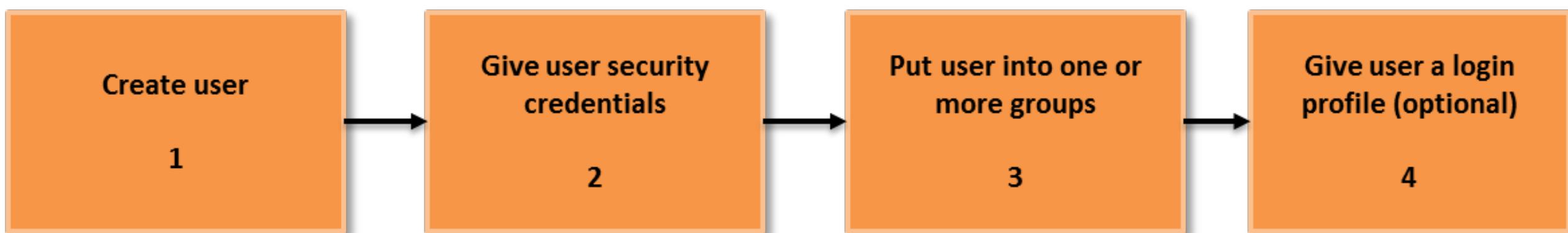
- **Shared access to your AWS account** - You can grant other people permission to administer and use resources in your AWS account without having to share your password or access key.
- **Granular permissions** - You can grant different permissions to different people for different resources.
- **Secure access to AWS resources for applications that run on Amazon EC2** - You can use IAM features to securely give applications that run on EC2 instances the credentials that they need in order to access other AWS resources.

Benefits of IAM

- Enhanced Security
- Granular control
- Temporary Credentials
- Flexible security credential management
- Leverage external identity systems
- Seamlessly integrated into AWS services

Enhanced Security

- IAM enables security best practices by allowing you to grant unique security credentials to users and groups to specify which AWS service APIs and resources they can access.
- IAM is secure by default; users have no access to AWS resources until permissions are explicitly granted.



Granular control

- IAM provides the granularity to control a user's access to specific AWS services and resources using permissions. For example, terminating EC2 instances or reading the contents of an Amazon S3 bucket.

Temporary Credentials

- In addition to defining access permissions directly to users and groups, IAM lets you create roles.
- Roles allow you to define a set of permissions and then let authenticated users or EC2 instances assume them, getting temporary access to the resources you define.

Flexible security credential management

- IAM allows you to authenticate users in several ways, depending on how they want to use AWS services. You can assign a range of security credentials including passwords, key pairs, and X.509 certificates
- You can also enforce [multi-factor authentication \(MFA\)](#) on users who access the AWS Management Console or use APIs

Leverage external identity systems

- You can use IAM to grant your employees and applications access to the AWS Management Console and to AWS service APIs, using your existing identity systems.
- AWS supports federation from corporate systems like Microsoft Active Directory as well as external Web Identity Providers like Google and Facebook.

Seamlessly integrated into AWS services

- IAM is integrated into most AWS services.
- This provides the ability to define access controls from one place in the AWS Management Console that will take effect throughout your AWS environment.

IAM Best Practices

- Lock Away Your AWS Account Root User Access Keys
- Create Individual IAM Users
- Use AWS Defined Policies to Assign Permissions Whenever Possible
- Use Groups to Assign Permissions to IAM Users
- Grant Least Privilege
- Use Access Levels to Review IAM Permissions
- Configure a Strong Password Policy for Your Users
- Enable MFA for Privileged Users
- Use Roles for Applications That Run on Amazon EC2 Instances
- Delegate by Using Roles Instead of by Sharing Credentials
- Rotate Credentials Regularly
- Remove Unnecessary Credentials

Additional Resources

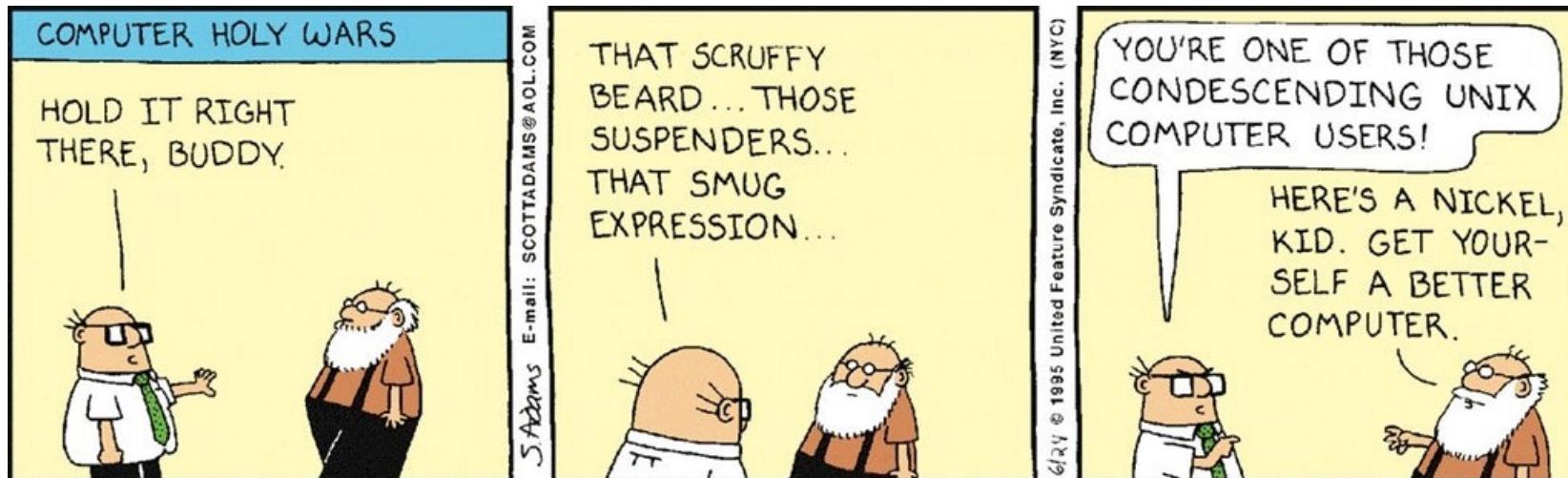
See Lecture Page

Introduction to Linux Command Line

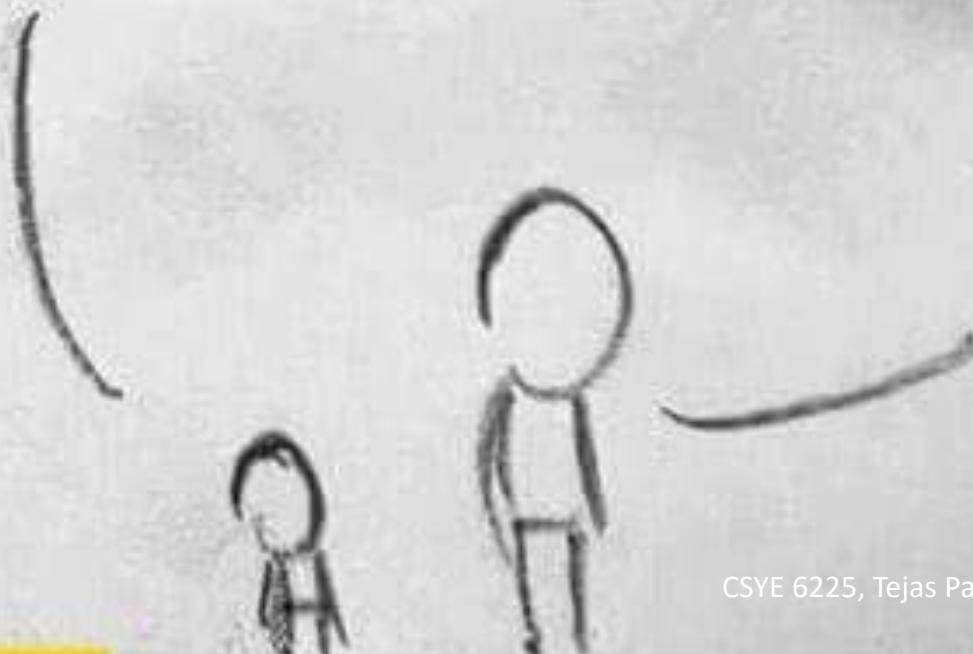
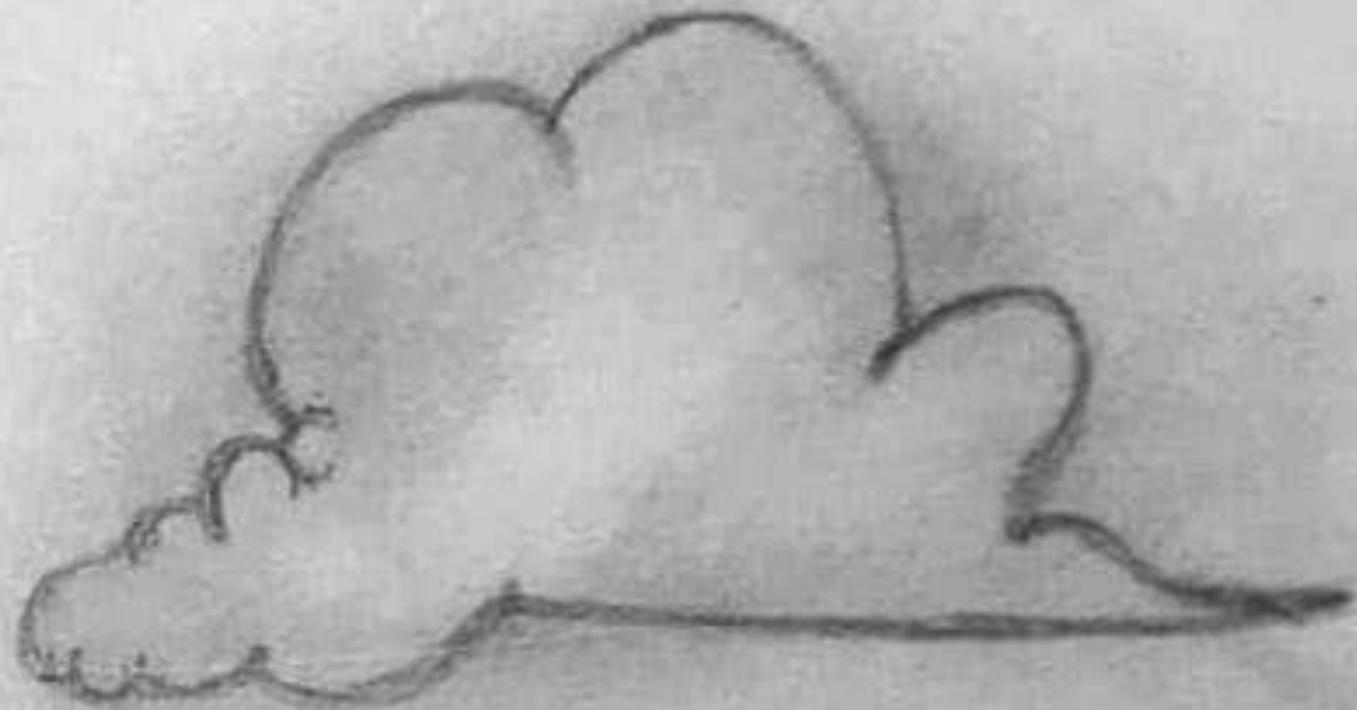
Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225

Northeastern University



DADDY, WHAT ARE
CLOUDS MADE OF?



LINUX SERVERS,
MOSTLY

What is Linux?

Linux is part of the OS, it is a **KERNEL** of the OS and Linux is at the heart of the OS

Amazon & Oracle Linux -> CentOS (fixes made upstream then viewable to Linux)

RedHat -> CentOS , before Fedora (Latest every 6mo)

Ubuntu -> Debian (stable OS)

Enterprise customers love stability -> Choose a stable Distribution (ex: Debian, CentOS)

- The term “Linux” is often used to refer to the entire operating system, but in reality, Linux is the operating system kernel, which is started by the boot loader, which is itself started by the BIOS/UEFI.
- The kernel assumes a role like that of a conductor in an orchestra—it ensures coordination between hardware and software.
- This role includes managing hardware, processes, users, permissions, and the file system.
- The kernel provides a common base to all other programs on the system and typically runs in *ring zero*, also known as **kernel space**.

Components of Operating Systems

No new applications can be run on Kernel Space
All applications go to User Space
(Context ? switching between spaces)
Netflix uses kernel - Mint (Not linux)
its faster

- **The Bootloader:** The software that manages the boot process of your computer.
- **The kernel:** This is the one piece of the whole that is actually called “Linux”. The kernel is the core of the system and manages the CPU, memory, and peripheral devices. The kernel is the “lowest” level of the OS.
- **Daemons:** These are background services (printing, sound, scheduling, etc) that either start up during boot, or after you log into the desktop.
- **Desktop Environment:** This is the piece of the puzzle that the users interact with.
- **Applications:** Desktop environments do not offer the full array of apps.
- **The Shell:** A command process that allows you to control the computer via commands typed into a text interface.

The Command Line (Linux Shell)

- The **shell** acts as an interface between the user and the kernel.
- When a user logs in, the login program checks the username and password, and then starts another program called the shell.
- The shell is a command line interpreter (CLI). It interprets the commands the user types in and arranges for them to be carried out.
- The commands are themselves programs: when they terminate, the shell gives the user another prompt.

Popular Linux Shells

- sh - Bourne Shell
- ksh - Korn Shell
- csh / tcsh - C Shell
- bash - Bourne-Again Shell
- zsh - Z shell

Popular Linux Shells

Bourne again shell: bash -> Easy to use with only keyboard
Ksh, csh / tcsh -> Linux Administration

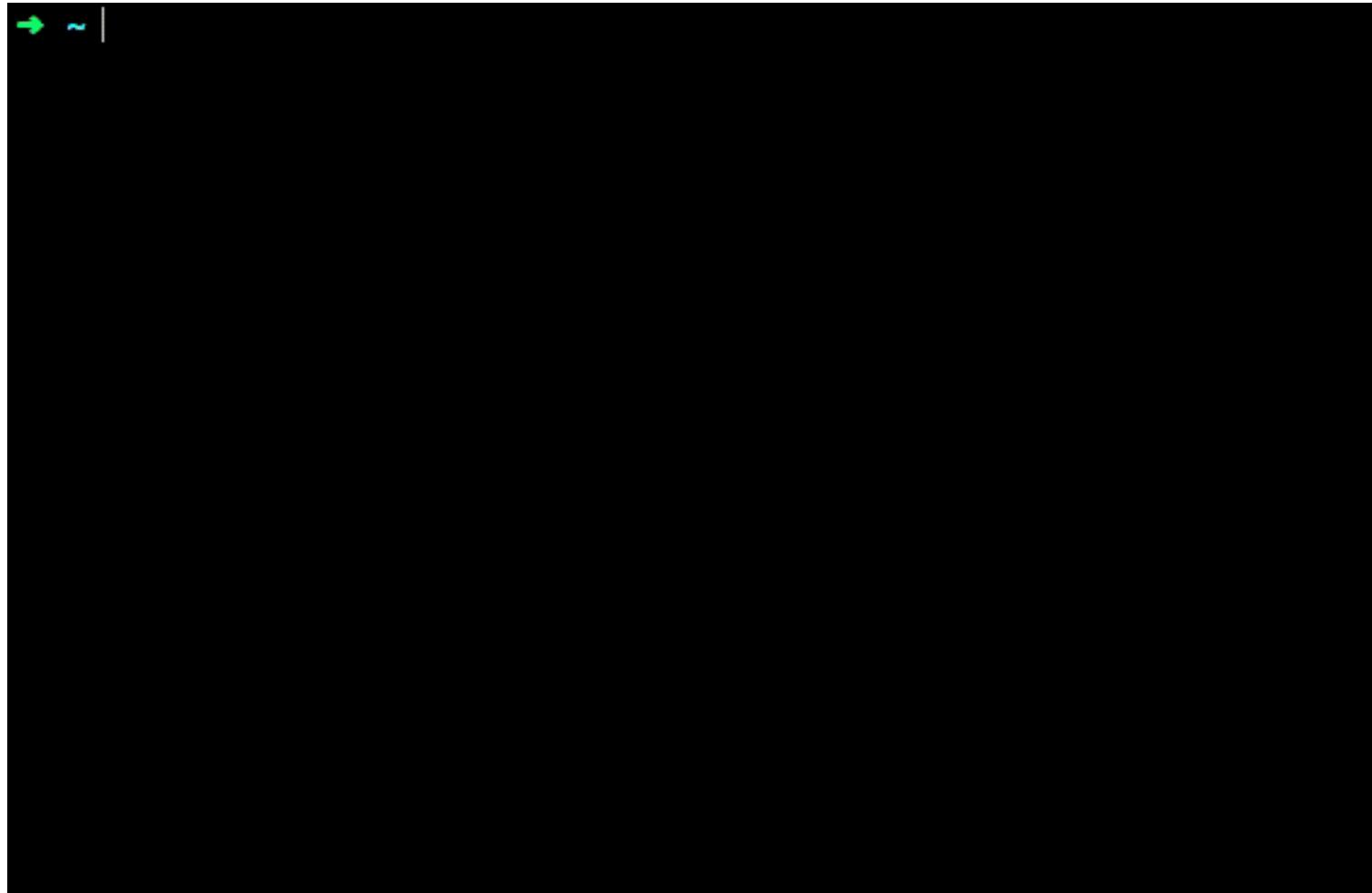
Zsh -> not recommended for servers: “oh my zsh” -
<https://ohmyz.sh/>

Shells may not be compatible on diff distributions

Bash Shell Special Characters

Even shells have reserved words (similar to programming languages, ex: int)

Switching Between Linux Shells



Bash Shell Special Characters

Special Character	Meaning
~	current user's home directory, same as \$HOME (e.g. /home/user)
\$	used to access a variable (e.g. \$HOME)
&	used to put a command in the background
*	wildcard, matching zero or more character

Do not touch /boot & /root on

Virtual directories: may not persist on reboot, changeable any time, ex: /dev, /proc, /run, /sys, /tmp (cleared on each reboot)

/etc -> (global) configuration is present here

System User (admins, have /home) Vs Human Users (may not have /home)

On a server nothing should be run on /home (/home upload OS, /opt transfer to run OS)

Application core lives on partitions of /mnt

/root (requires su (super user) to run it) master privileges

Applications should be run on user not super user

Any user can be written to /tmp, since its read write by anyone, softwares can't be executed from here

Linux File System

<https://www.linuxfoundation.org/blog/blog/classic-sysadmin-the-linux-filesystem-explained>

Linux Command Line

Linux is case sensitive

Mv is used for rename in Linux??

System Admin want to see logs in /log (convention)

Don't give all chmod 777 (Give proper permissions)

Create mysql group (give permissions for mysql user only)

No other user should be able to use mysql

Never use common ssh keys for everything (id_rsa) create it in new one

```
dr-xr-xr-x. 17 root root 224 Sep 13 2022 .
dr-xr-xr-x. 17 root root 224 Sep 13 2022 ..
lrwxrwxrwx. 1 root root 7 Jun 22 2021 bin -> usr/bin
dr-xr-xr-x. 6 root root 4096 Jan 25 18:41 boot
drwxr-xr-x. 16 root root 2620 Jan 25 18:39 dev
drwxr-xr-x. 93 root root 8192 Jan 25 18:40 etc
drwxr-xr-x. 3 root root 24 Jan 25 18:40 home
lrwxrwxrwx. 1 root root 7 Jun 22 2021 lib -> usr/lib
lrwxrwxrwx. 1 root root 9 Jun 22 2021 lib64 -> usr/lib64
drwxr-xr-x. 2 root root 6 Jun 22 2021 media
drwxr-xr-x. 2 root root 6 Jun 22 2021 mnt
drwxr-xr-x. 2 root root 6 Jun 22 2021 opt
dr-xr-xr-x. 107 root root 0 Jan 25 18:39 proc
dr-xr-x---. 3 root root 103 Jan 25 18:40 root
drwxr-xr-x. 28 root root 840 Jan 25 18:40 run
lrwxrwxrwx. 1 root root 8 Jun 22 2021 sbin -> usr/sbin
drwxr-xr-x. 2 root root 6 Jun 22 2021 srv
dr-xr-xr-x. 13 root root 0 Jan 25 18:39 sys
drwxrwxrwt. 8 root root 172 Jan 25 18:56 tmp
drwxr-xr-x. 13 root root 158 Sep 13 2022 usr
drwxr-xr-x. 20 root root 4096 Jan 25 18:39 var
```

/bin

/bin is the directory that contains *binaries*, that is, some of the applications and programs you can run.

/boot

- The */boot* directory contains files required for starting your system.
- **DO NOT TOUCH!**. If you mess up one of the files in here, you may not be able to run your Linux and it is a pain to repair.

/dev

/dev contains *device* files. Many of these are generated at boot time or even on the fly.

/dev is a virtual directory, the data here may not be persisted across BOOTUP, some other examples of such directories are */proc*, */run*, */sbin*

hard disk & network devices

`/etc`

- `/etc` is the directory where names start to get confusing. `/etc` gets its name from the earliest Unixes and it was literally “et cetera” because it was the dumping ground for system files administrators were not sure where else to put.
- Nowadays, it would be more appropriate to say that *etc* stands for “Everything to configure,” as it contains most, if not all system-wide configuration files.

Any path that starts with a `/home/etc` is an absolute path

/home

/home is where you will find your users' personal directories.

/lib

/lib is where *libraries* live. Libraries are files containing code that your applications can use. They contain snippets of code that applications use to draw windows on your desktop, control peripherals, or send files to your hard disk.

/media

The */media* directory is where external storage will be automatically mounted when you plug it in and try to access it.

[hdmi cable, mouse](#)

/mnt

The */mnt* directory, however, is a bit of remnant from days gone by. This is where you would manually mount storage devices or partitions. It is not used very often nowadays.

/opt

The */opt* directory is often where software you compile (that is, you build yourself from source code and do not install from your distribution repositories) sometimes lands.

It is an abbreviation for "optional." Historically, */opt* has been used to install non-traditional or non-standard software that isn't considered part of the main underlying system (OS)

/proc

- */proc*, like */dev* is a virtual directory.
- It contains information about your computer, such as information about your CPU and the kernel your Linux system is running.
- As with */dev*, the files and directories are generated when your computer starts, or on the fly, as your system is running and things change.

/root

- */root* is the home directory of the superuser (also known as the “Administrator”) of the system.
- It is separate from the rest of the users’ home directories
BECAUSE YOU ARE NOT MEANT TO TOUCH IT.
- Keep your own stuff in your own directories.

/run

- */run* is another new directory. System processes use it to store temporary data for their own nefarious reasons.
- This is another one of those DO NOT TOUCH folders.

/sbin

- */sbin* is similar to */bin*, but it contains applications that only the superuser (hence the initial s) will need.
- You can use these applications with the sudo command that temporarily concedes you superuser powers on many distributions. */sbin* typically contains tools that can install stuff, delete stuff and format stuff.
- As you can imagine, some of these instructions are lethal if you use them improperly, so handle with care.

/usr

- */usr* contains a mish-mash of directories which in turn contain applications, libraries, documentation, wallpapers, icons and a long list of other stuff that need to be shared by applications and services.

/srv

- The `/srv` directory contains data for servers.
- If you are running a web server from your Linux box, your HTML files for your sites would go into `/srv/http` (or `/srv/www`).
- If you were running an FTP server, your files would go into `/srv/ftp`.

/sys

sys is another virtual directory like */proc* and */dev* and also contains information from devices connected to your computer.

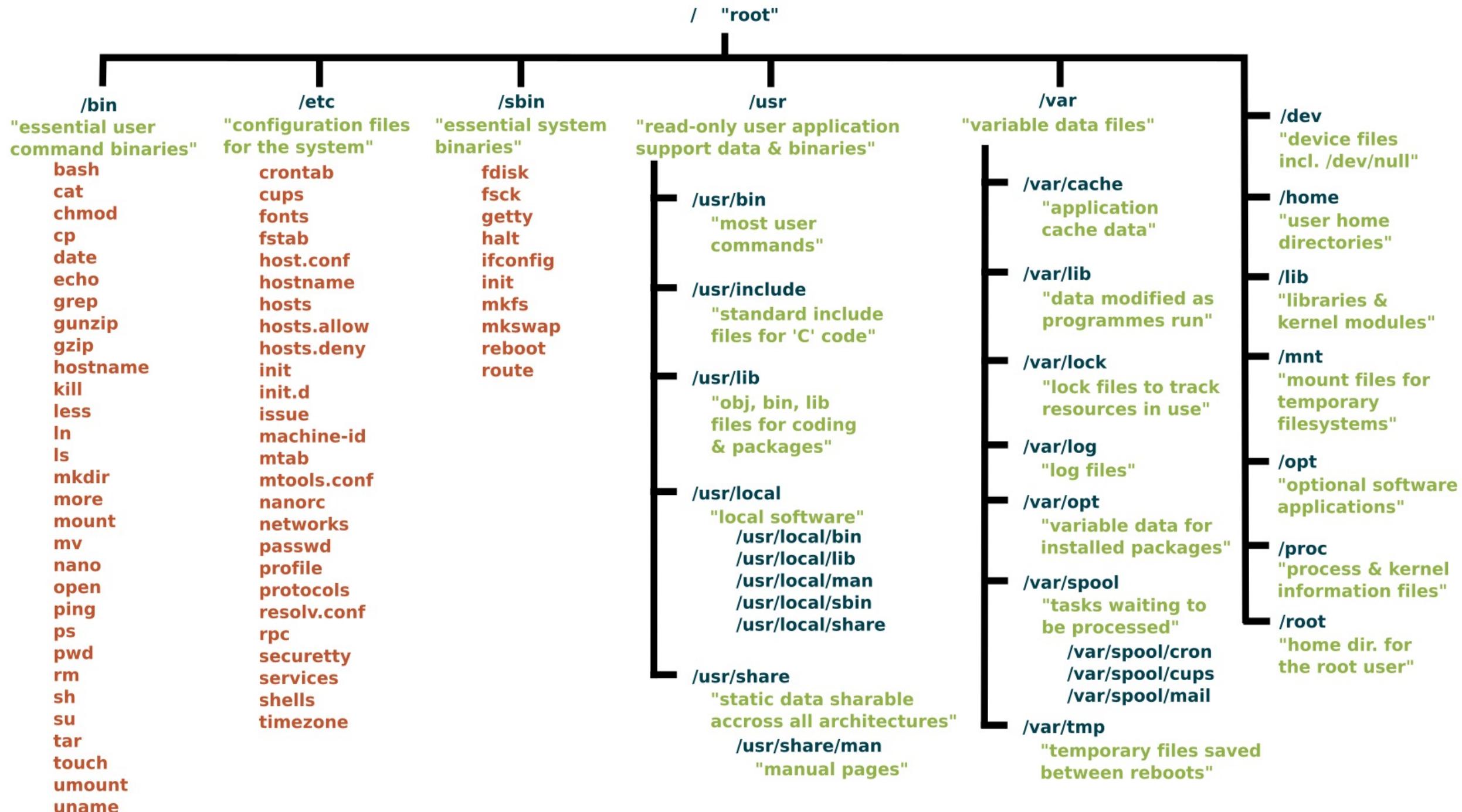
/tmp

- */tmp* contains temporary files, usually placed there by applications that you are running.
- The files and directories often (not always) contain data that an application doesn't need right now, but may need later on.
- You can also use */tmp* to store your own temporary files — */tmp* is one of the few directories hanging off / that you can actually interact with without becoming superuser.

/tmp is cleared at each boot up

/var

/var contains things like logs in the */var/log* subdirectories.



Navigation

pwd, cd, ls

pwd – Print Working Directory

- At any given time, we are inside a single directory, and we can see the files contained in the directory and the pathway to the directory above us (called the parent directory) and any subdirectories below us.
- The directory we are standing in is called the current working directory.
- To display the current working directory, we use the **pwd** (print working directory) command.

```
root@localhost:~# pwd  
/root  
root@localhost:~#
```

The 'which' command in Linux is a simple and effective tool for locating the paths of executable files. 'which' operates by searching through the directories listed in the PATH environment variable. It is important to remember that 'which' can only locate executables, not shell built-in commands.

Changing the Current Working Directory

- To change your working directory, we use the **cd** command.
- To do this, type **cd** followed by the pathname of the desired working directory.
- Pathnames can be specified in one of two different ways; as ***absolute pathnames*** or as ***relative pathnames***.

```
root@localhost:~# pwd  
/root  
root@localhost:~# cd /etc/ssh ←  
root@localhost:/etc/ssh# pwd  
/etc/ssh  
root@localhost:/etc/ssh# cd ~/csye6225 ←  
root@localhost:~/csye6225# pwd  
/root/csye6225  
root@localhost:~/csye6225#
```

ls – Listing the Contents Of A Directory

To list the files and directories in the current working directory, we use the ls command.

```
root@localhost:~# pwd
/root
root@localhost:~# ls ←
csye6225 file.txt
root@localhost:~# ls -l ←
total 4
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
-rw-r--r-- 1 root root 0 Aug 13 03:33 file.txt
root@localhost:~# ls -al ←
total 40
drwx----- 5 root root 4096 Aug 13 03:33 .
drwxr-xr-x 22 root root 4096 Mar 7 20:37 ..
-rw----- 1 root root 5 Aug 13 03:23 .bash_history
-rw-r--r-- 1 root root 3106 Oct 22 2015 .bashrc
drwx----- 2 root root 4096 Jul 22 2016 .cache
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
-rw-r--r-- 1 root root 0 Aug 13 03:33 file.txt
-rw----- 1 root root 35 Sep 15 2016 .lesshist
-rw-r--r-- 1 root root 148 Aug 17 2015 .profile
drwx----- 2 root root 4096 Aug 13 03:23 .ssh
-rw----- 1 root root 663 Aug 13 03:25 .viminfo
root@localhost:~# |
```

Important Facts About Filenames

- Filenames that begin with period character are hidden.
- Filenames and commands in Linux are case sensitive.
- Linux has no concept of a “file extension”.

Manipulating Files & Directories

cp, mv, mkdir, rm, ln

cp – Copy files and directories

The **cp** command copies files or directories.

```
root@localhost:~# ls  
csye6225 file.txt  
root@localhost:~# cp file.txt file-copy.txt ←  
root@localhost:~# ls  
csye6225 file-copy.txt file.txt  
root@localhost:~# cp csye6225 csye6225_copy ←  
cp: omitting directory 'csye6225' WHAT!!  
root@localhost:~# cp -R csye6225 csye6225_copy ←  
root@localhost:~# |
```

mv – Move/rename files and directories

- The **mv** command performs both file *moving* and file *renaming*, depending on how it is used.
- In either case, the original filename no longer exists after the operation.
- **mv** is used in much the same way as **cp**.

```
root@localhost:~# ls
csye6225 csye6225_copy dir1 dir2 dir3 dir4 file-copy.txt file.txt
root@localhost:~# mv file.txt mv_file.txt ←
root@localhost:~# ls
csye6225 csye6225_copy dir1 dir2 dir3 dir4 file-copy.txt mv_file.txt
root@localhost:~#
```

mkdir – Create directories

- The **mkdir** command is used to create directories.

Mkdir -p log1/log2/log3 -> -p is used to create directories if they don't exist.

```
root@localhost:~# mkdir dir1 ←
root@localhost:~# mkdir dir2 dir3 ←
root@localhost:~# mkdir dir4/dir5/dir6 ←
mkdir: cannot create directory 'dir4/dir5/dir6': No such file or directory
root@localhost:~# mkdir -p dir4/dir5/dir6 ←
root@localhost:~#
```

```
root@localhost:~# tree
```

```
.
├── csye6225
├── csye6225_copy
├── dir1
├── dir2
├── dir3
├── dir4
│   └── dir5
│       └── dir6
└── file-copy.txt
    └── file.txt
```

```
8 directories, 2 files
root@localhost:~# |
```

rm – remove files and directories

- The **rm** command is used to remove (delete) files and directories

```
root@localhost:~# ls
csye6225 csye6225_copy dir1 dir2 dir3 dir4 file-copy.txt mv_file.txt
root@localhost:~# rm file-copy.txt ←
root@localhost:~# rm dir4 ←
rm: cannot remove 'dir4': Is a directory
root@localhost:~# rm -r dir4 ←
root@localhost:~# ls
csye6225 csye6225_copy dir1 dir2 dir3 mv_file.txt
root@localhost:~# |
```

In – Create hard or soft symbolic links

- The **In** command is used to create either hard or symbolic links.
- **Symbolic links** work by creating a special type of file that contains a text pointer to the referenced file or directory.
 - When you delete a symbolic link, only the link is deleted, not the file itself.
- A **hard link** is indistinguishable from the file itself.
 - Unlike a symbolic link, when you list a directory containing a hard link you will see no special indication of the link.

```
root@localhost:~# ls
csye6225 csye6225_copy dir1 dir2 dir3 mv_file.txt
root@localhost:~# ln -s mv_file.txt softlink.txt
root@localhost:~# ls
csye6225 csye6225_copy dir1 dir2 dir3 mv_file.txt softlink.txt
root@localhost:~# ls -l
total 20
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root    0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root   11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~# |
```

I/O Redirection

Commands: cat, grep, head, tail

& Pipes

Output Streams

- Standard Output
 - **System.out.println("this message goes to standard output");**
- Standard Error
 - **System.err.println("this message goes to standard error");**

Redirecting | Standard Output

- I/O redirection allows us to redefine where standard output goes.
- To redirect standard output to another file instead of the screen, we use the “>” redirection operator followed by the name of the file.

```
root@localhost:~# ls -l
total 20
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root    0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root   11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~# ls -l > ls-output.txt ↗
root@localhost:~# cat ls-output.txt
total 20
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root    0 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root    0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root   11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~# |
```

Redirecting | Standard Error

- Redirecting standard error lacks the ease of a dedicated redirection operator.
- To redirect standard error, we must refer to its file descriptor.

```
root@localhost:~# ls -l 2> ls-err-output.txt ↗
total 24
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root    0 Aug 13 20:27 ls-err-output.txt
-rw-r--r-- 1 root root  429 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root    0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root   11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~#
```

Redirecting Standard Output And Standard Error To One File

- There are cases in which we may wish to capture all the output of a command to a single file.

```
root@localhost:~# ls -l > ls-std-out-err-output.txt 2>&1
root@localhost:~# |
```

/dev/null

- Used for disposing of unwanted output.
- It is sometimes referred to as “black hole”.

```
root@localhost:~# ls -l > /dev/null 2>&1
root@localhost:~#
```

cat - Concatenate files

- The **cat** command reads one or more files and copies them to standard output.

```
root@localhost:~# cat ls-output.txt ls-err-output.txt ↗
total 20
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root 0 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root 0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root 11 Aug 13 04:38 softlink.txt -> mv_file.txt
total 24
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root 0 Aug 13 20:28 ls-err-output.txt
-rw-r--r-- 1 root root 429 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root 0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root 11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~#
```

head – Output the first part of a file

- Sometimes you don't want all the output from a command. You may only want the first few lines.
- The head command prints the first ten lines of a file .

```
root@localhost:~# head ls-output.txt
total 20
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root 0 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root 0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root 11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~# head -n3 ls-output.txt
total 20
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
root@localhost:~# |
```

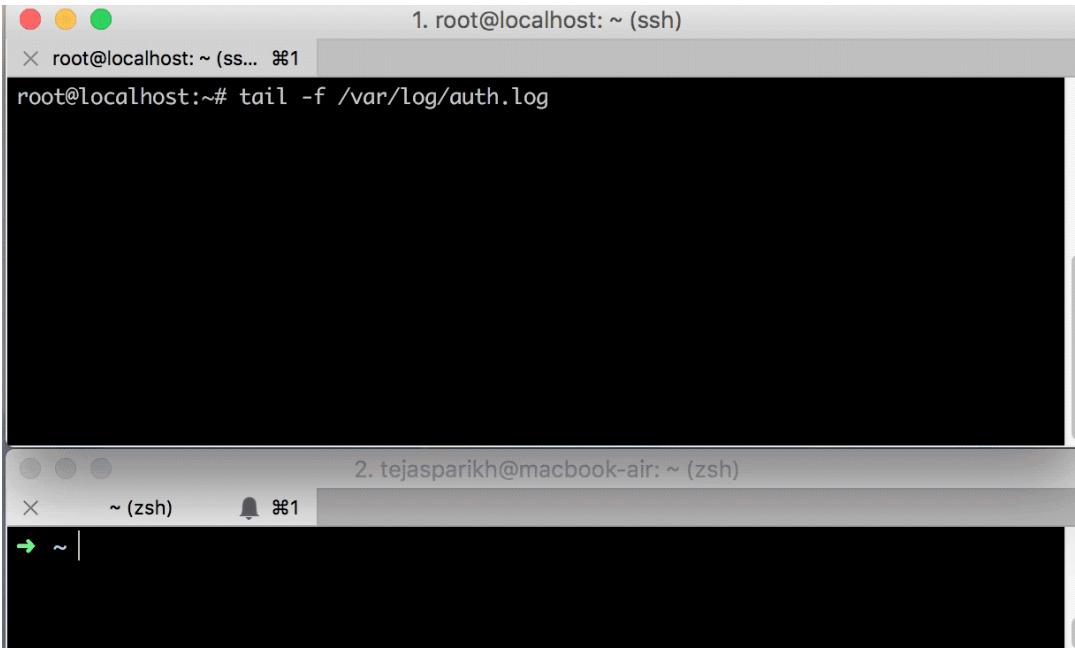
tail – Output the last part of a file

- Sometimes you don't want all the output from a command. You may only want the last few lines.
- The **tail** command prints the last ten lines of a file.

```
root@localhost:~# tail ls-output.txt
total 20
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root 0 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root 0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root 11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~# tail -n3 ls-output.txt
-rw-r--r-- 1 root root 0 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root 0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root 11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~#
```

tail – Follow files in real-time

- **tail** has an option which allows you to view files in real-time



The image shows two terminal windows side-by-side. The top window, titled '1. root@localhost: ~ (ssh)', displays the command 'root@localhost:~# tail -f /var/log/auth.log' followed by a large black redacted area. The bottom window, titled '2. tejasparikh@macbook-air: ~ (zsh)', shows a command prompt with a green arrow and a cursor, indicating it is ready for input.

```
1. root@localhost: ~ (ssh)
root@localhost:~# tail -f /var/log/auth.log

2. tejasparikh@macbook-air: ~ (zsh)
~ (zsh) ~ |
```

Pipelines

- The ability of commands to read data from standard input and send to standard output is utilized by a shell feature called **pipelines**.
- Using the pipe operator “|” (vertical bar), the standard output of one command can be **piped** into the standard input of another.

```
root@localhost:~# cat ls-output.txt | tail -n3
-rw-r--r-- 1 root root    0 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root    0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root 11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~#
```

grep – Print lines matching a pattern

- **grep** is a powerful program used to find text patterns.
- When grep encounters a “pattern”, it prints out the lines containing it.

```
root@localhost:~# cat ls-output.txt
total 20
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root    0 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root    0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root   11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~# cat ls-output.txt | grep mv_file
-rw-r--r-- 1 root root    0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root   11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~# |
```

The Difference Between > and |

- It may be hard to understand the redirection performed by the pipeline operator “|” versus the redirection operator “>”.
- The redirection operator (“>”) connects a command with a file while the pipeline operator (“|”) connects the output of one command with the input of a second command.

Permissions

Commands: chmod, su, sudo, chown, passwd

Permissions

Under the traditional UNIX and Linux filesystem model, every file has a set of nine permission bits that control who can read, write, and execute the contents of the file.

The Permission Bits

- Three sets of permissions define access for the owner of the file, the group owners of the file, and everyone else (in that order).
- Each set has three bits: a **read** bit, a **write** bit, and an **execute** bit (in that order)

```
root@localhost:~# ls -l
total 32
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root 488 Aug 13 20:28 ls-err-output.txt
-rw-r--r-- 1 root root 429 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root 555 Aug 13 20:29 ls-std-out-err-output.txt
-rw-r--r-- 1 root root 0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root 11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~# |
```

Owner	Group	World
rwx	rwx	rwx

Permission Attributes

Attribute	Files	Directories
r	Allows a file to be opened and read.	Allows a directory's contents to be listed if the execute attribute is also set.
w	Allows a file to be written to or truncated, however this attribute does not allow files to be renamed or deleted. The ability to delete or rename files is determined by directory attributes.	Allows files within a directory to be created, deleted, and renamed if the execute attribute is also set.
x	Allows a file to be treated as a program and executed. Program files written in scripting languages must also be set as readable to be executed.	Allows a directory to be entered, e.g., <code>cd directory</code> .

Permission Attribute Examples

File Attributes	Meaning
- rwx-----	A regular file that is readable, writable, and executable by the file's owner. No one else has any access.
- rw-----	A regular file that is readable and writable by the file's owner. No one else has any access.
- rwx-r--r--	A regular file that is readable and writable by the file's owner. Members of the file's owner group may read the file. The file is world-readable.
- rwxr-xr-x	A regular file that is readable, writable, and executable by the file's owner. The file may be read and executed by everybody else.
- rw-rw----	A regular file that is readable and writable by the file's owner and members of the file's group owner only.
1rwxrwxrwx	A symbolic link. All symbolic links have "dummy" permissions. The real permissions are kept with the actual file pointed to by the symbolic link.
drwxrwx---	A directory. The owner and the members of the owner group may enter the directory and, create, rename and remove files within the directory.
drwxr-x---	A directory. The owner may enter the directory and create, rename and delete files within the directory. Members of the owner group may enter the directory but cannot create, delete or rename files.

File Modes In Binary And Octal

Octal	Binary	File Mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	r w-
7	111	rwx

chmod – Change File Mode

- The **chmod** command changes the permissions on a file.
- Only the owner of the file and the superuser can change its permissions.

To run script, we have to set permission on it, we can use chmod

```
root@localhost:~# ls -l
total 32
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root 488 Aug 13 20:28 ls-err-output.txt
-rw-r--r-- 1 root root 429 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root 555 Aug 13 20:29 ls-std-out-err-output.txt
-rw-r--r-- 1 root root 0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root 11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~# chmod 660 mv_file.txt ↗
root@localhost:~# ls -l
total 32
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root 488 Aug 13 20:28 ls-err-output.txt
-rw-r--r-- 1 root root 429 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root 555 Aug 13 20:29 ls-std-out-err-output.txt
-rw-rw--- 1 root root 0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root 11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~# |
```

chown – Change File Owner And Group

- The **chown** command is used to change the owner and group owner of a file or directory.
- Superuser privileges are required to use this command.
- **chown** can change the file owner and/or the file group owner depending on the first argument of the command.

```
root@localhost:~# ls -l
total 32
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root 488 Aug 13 20:28 ls-err-output.txt
-rw-r--r-- 1 root root 429 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root 555 Aug 13 20:29 ls-std-out-err-output.txt
-rw-rw---- 1 root root 0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root 11 Aug 13 04:38 softlink.txt -> mv_file.txt
```

```
root@localhost:~# chown root:newgroup mv_file.txt
root@localhost:~# ls -l
total 32
drwxr-xr-x 2 root root 4096 Aug 13 03:33 csye6225
drwxr-xr-x 2 root root 4096 Aug 13 04:26 csye6225_copy
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir1
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir2
drwxr-xr-x 2 root root 4096 Aug 13 04:29 dir3
-rw-r--r-- 1 root root 488 Aug 13 20:28 ls-err-output.txt
-rw-r--r-- 1 root root 429 Aug 13 20:24 ls-output.txt
-rw-r--r-- 1 root root 555 Aug 13 20:29 ls-std-out-err-output.txt
-rw-rw---- 1 root newgroup 0 Aug 13 03:33 mv_file.txt
lrwxrwxrwx 1 root root 11 Aug 13 04:38 softlink.txt -> mv_file.txt
root@localhost:~#
```

chown Argument Examples

Argument	Results
bob	Changes the ownership of the file from its current owner to user bob .
bob:users	Changes the ownership of the file from its current owner to user bob and changes the file group owner to group users .
:admins	Changes the group owner to the group admins . The file owner is unchanged.
bob:	Change the file owner from the current owner to user bob and changes the group owner to the login group of user bob .

su – Run A Shell With Substitute User And Group IDs

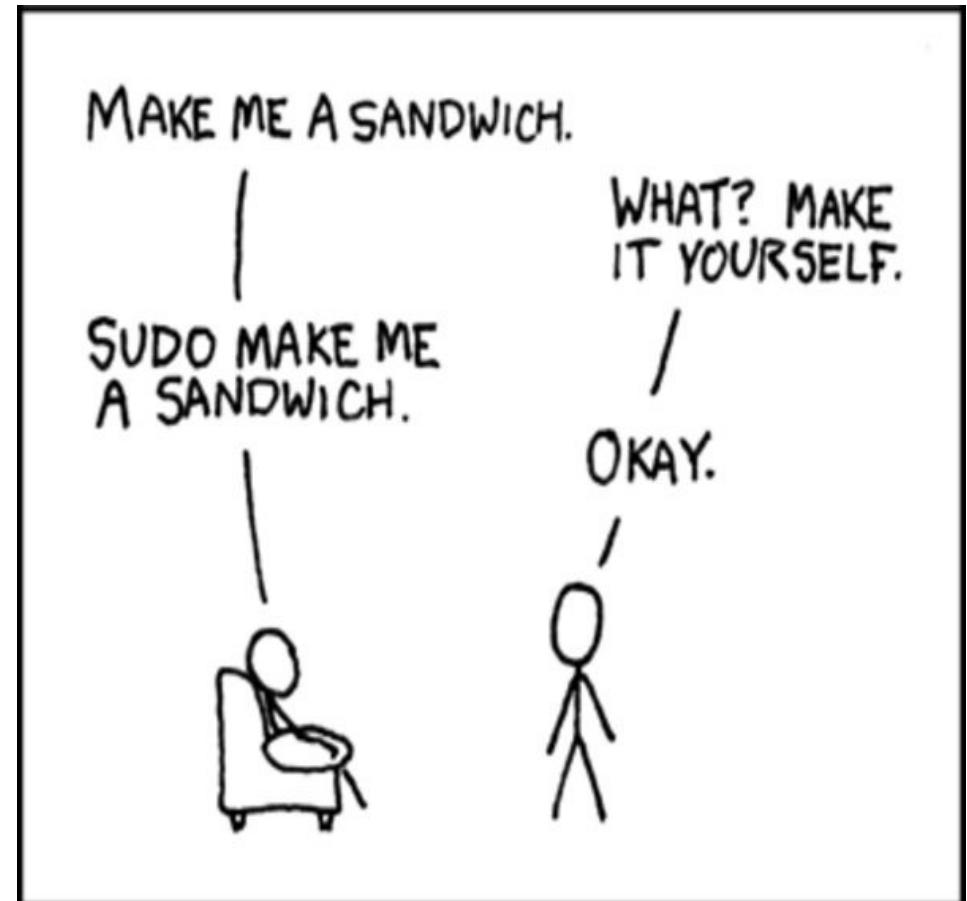
The **su** command is used to start a shell as another user. The command syntax looks like this:

```
su [-[l]] [user]
```

If the “-l” option is included, the resulting shell session is a login shell for the specified user. This means that the user's environment is loaded, and the working directory is changed to the user's home directory.

sudo – Execute A Command As Another User

- The **sudo** command is like su in many ways but has some important additional capabilities.
- The administrator can configure sudo to allow an ordinary user to execute commands as a different user (usually the superuser) in a very controlled way.
- A user may be restricted to one or more specific commands and no others.
- Another important difference is that the use of sudo does not require access to the superuser's password. To authenticate using sudo, the user uses his/her own password.



Processes

Commands: ps, top, fg, kill, shutdown, nohup

ps – Viewing Processes

- ps is the most used command to view processes.

```
root@localhost:~# ps
  PID TTY      TIME CMD
 8961 pts/0    00:00:00 bash
 9475 pts/0    00:00:00 ps
root@localhost:~# |
```

```
root@localhost:~# [ps -ef | grep bash]
root  8961 8910  0 20:24 pts/0    00:00:00 -bash
root  9479 8961  0 21:31 pts/0    00:00:00 grep --color=auto bash
root@localhost:~# ps
  PID TTY      TIME CMD
 8961 pts/0    00:00:00 bash
 9480 pts/0    00:00:00 ps
root@localhost:~#
```

```
root@localhost:~# ps -ef
UID      PID  PPID   C STIME TTY          TIME CMD
root      1    0  0 03:18 ?
root      2    0  0 03:18 ?
root      3    2  0 03:18 ?
root      5    2  0 03:18 ?
root      6    2  0 03:18 ?
root      7    2  0 03:18 ?
root      8    2  0 03:18 ?
root      9    2  0 03:18 ?
root     10    2  0 03:18 ?
root     11    2  0 03:18 ?
root     12    2  0 03:18 ?
root     13    2  0 03:18 ?
root     15    2  0 03:18 ?
root    313    2  0 03:18 ?
root    314    2  0 03:18 ?
root    316    2  0 03:18 ?
root    317    2  0 03:18 ?
root    318    2  0 03:18 ?
root    320    2  0 03:18 ?

00:00:04 /lib/systemd/systemd --sys
00:00:00 [kthreadd]
00:00:00 [ksoftirqd/0]
00:00:00 [kworker/0:0H]
00:00:00 [kworker/u2:0]
00:00:01 [rcu_sched]
00:00:00 [rcu_bh]
00:00:00 [migration/0]
00:00:00 [lru-add-drain]
00:00:00 [cpuhp/0]
00:00:00 [kdevtmpfs]
00:00:00 [netns]
00:00:00 [oom_reaper]
00:00:00 [writeback]
00:00:00 [kcompactd0]
00:00:00 [crypto]
00:00:00 [kintegrityd]
00:00:00 [bioset]
00:00:00 [kblockd]
```

top – Viewing Processes Dynamically

- While the **ps** command can reveal a lot about what the machine is doing, it provides only a snapshot of the machine's state at the moment the **ps** command is executed.
- To see a more dynamic view of the machine's activity, we use the **top** command

```
root@localhost:~# |
```



Putting A Process In The Background

- To launch a program so that it is immediately placed in the background, we follow the command with an “&” character

```
root@localhost:~# sleep 3  
|
```

nohup – run a command without hangups

- **nohup** command is used to continue run programs in background after logging off.
- Ignores all **hangup** signals.
- Writes all terminal output to nohup.out file. nohup.out file is created in the directory the command is executed.

Example Usage: nohup *somecommand* &

jobs – List background or suspended processes

When a process is running, backgrounded or suspended, it will be entered onto a list along with a job number. To examine this list, use the command **jobs**.

fg – Returning A Process To The Foreground

To return a process to the foreground, use the **fg** command .

Example Usage: **fg <JOB_ID>**

kill – Terminate Process

The **kill** command is used to “kill” processes.

This allows us to terminate programs that need killing.

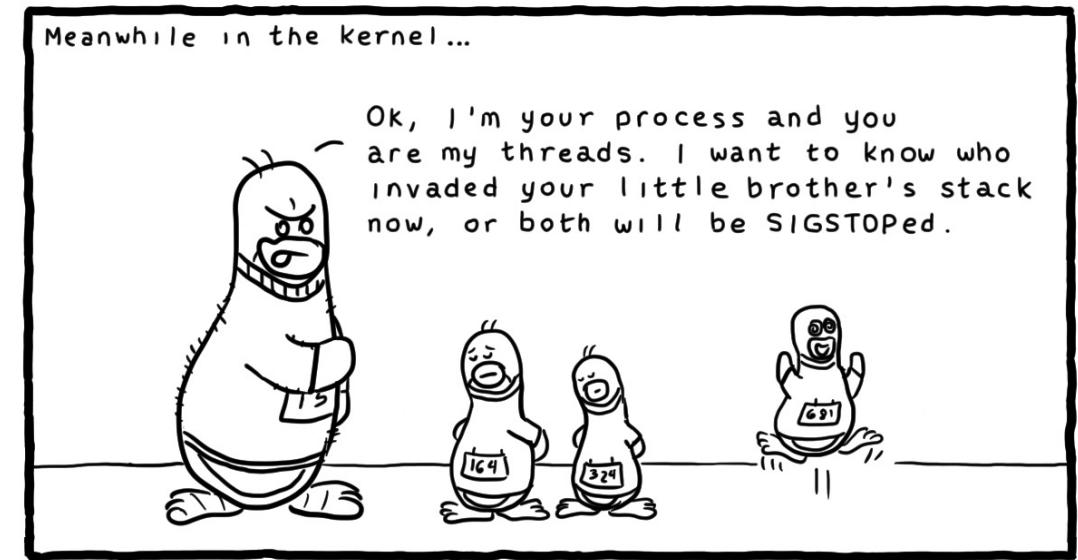
Example Usage: `kill <pid>`

If a process refuses to be killed, uses the **-9** option.

Example Usage: `kill -9 <pid>`

Notes:

- `kill -9 -1` will kill all process that you have permission to kill.
- It will also end your current login session.
- Never execute this command as root as it will KILL ALL processes CRASHING your server.



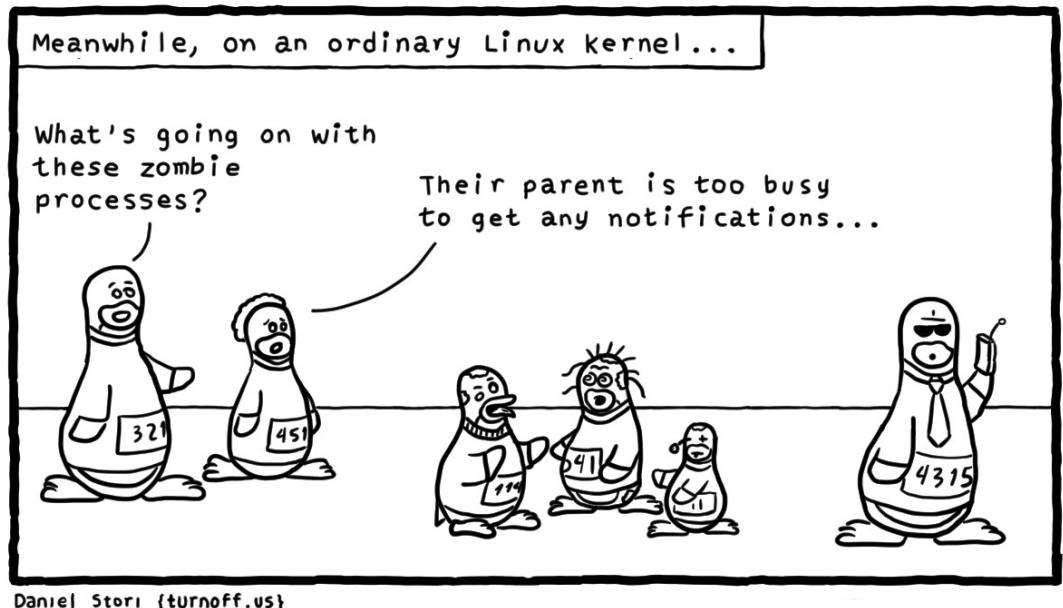
Shutdown – Poweroff or Reboot

shutdown command will terminate processes in orderly function before powering off the system.

sudo shutdown -r now – reboot the system

Zombie Process

- On Unix and Unix-like computer operating systems, a zombie process or defunct process is a process that has completed execution (via the exit system call) but still has an entry in the process table: it is a process in the "Terminated state".
- This occurs for child processes, where the entry is still needed to allow the parent process to read its child's exit status: once the exit status is read via the wait system call, the zombie's entry is removed from the process table, and it is said to be "reaped".
- Zombie process can be identified with **ps aux | grep 'Z'** command.



Disk Management

Command: du, df

du – estimate file space usage

Usage: du [OPTION]... [FILE]...

Example Usage:

- du –sm (display output in Megabytes)
- du –sg (display output in Gigabytes)

df – report file system disk space usage

Usage: df [OPTION]... [FILE]...

Example Usage:

- df –m

(display output in Megabytes)

Filesystem	1M-blocks	Used	Available	Use%	Mounted on
/dev/root	19907	817	18063	5%	/
devtmpfs	493	0	493	0%	/dev
tmpfs	495	0	495	0%	/dev/shm
tmpfs	495	27	469	6%	/run
tmpfs	5	0	5	0%	/run/lock
tmpfs	495	0	495	0%	/sys/fs/cgroup
tmpfs	99	0	99	0%	/run/user/0

Misc

Commands: tar, zip, unzip, ssh, scp

Archives

- **tar** – to archive a file

Usage: tar [OPTION] DEST SOURCE

Example Usage: tar -cvf /home/archive.tar /home/original
tar -xvf /home/archive.tar

- **zip** – package and compress (archive) files

Usage: zip [OPTION] DEST SOURCE

Example Usage: zip original.zip original

- **unzip** – list, test and extract compressed files in a ZIP archive

Usage: unzip filename

Example Usage: unzip original.zip

Network

- **ssh – SSH client (remote login program)**

SSH is a program for logging into a remote machine and for executing commands on a remote machine.

Usage: ssh [options] [user]@hostname

Example Usage: ssh guest@10.105.11.20

- **scp – secure copy (remote file copy program)**

scp copies files between hosts on a network

Usage: scp [options] [[user]@host1:file1] [[user]@host2:file2]

Example Usage: scp file1.txt guest@10.105.11.20:~/Desktop/



Getting Help

Commands: man

man – Display manual page

The manual pages tell you which options a particular command can take, and how each option modifies the behavior of the command.



Additional Resources

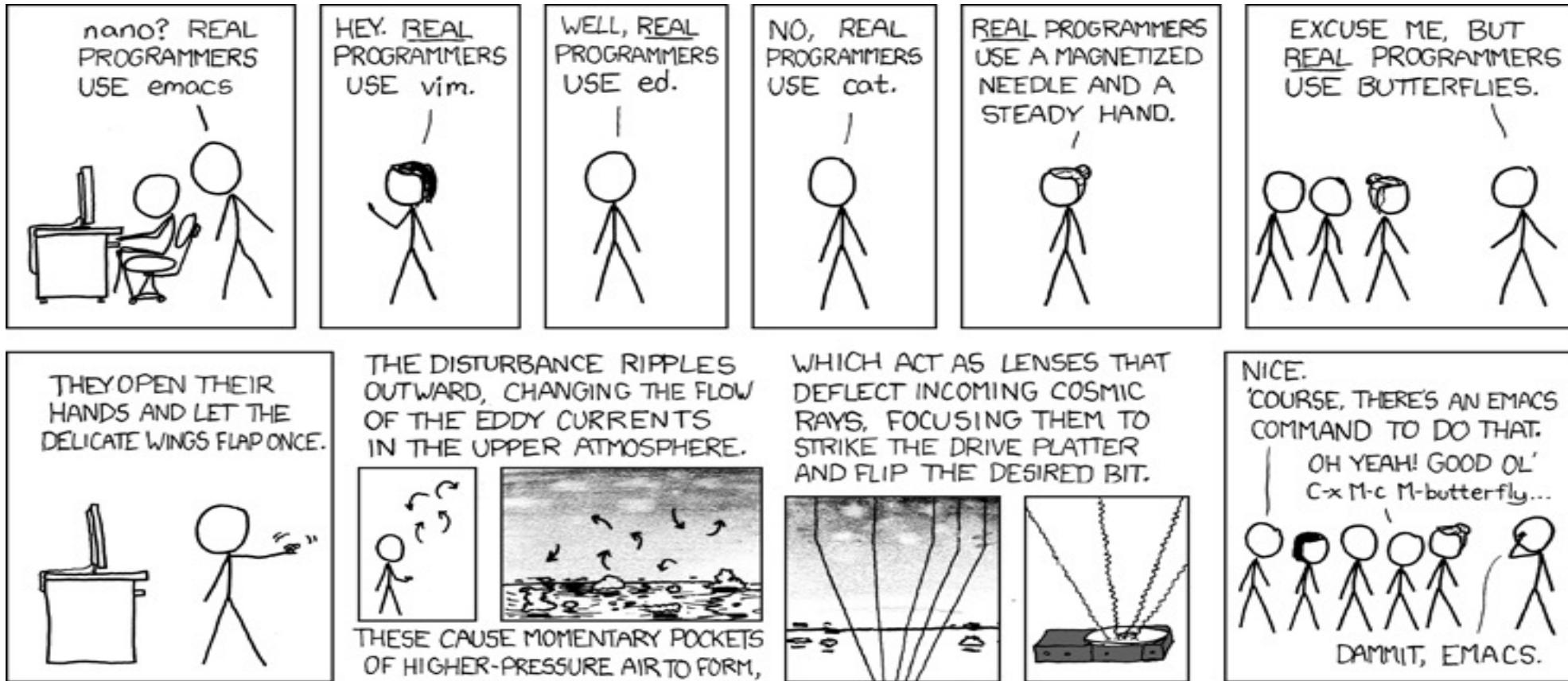
See Lecture Page

Editing with *vi*

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University

The Great Text Editor War



Why We Should Learn vi

- vi is always available.
- vi is lightweight and fast.

Vi can open files of large size

Vi is supported on most distributions of linux & mac os

A Little Background

- First version written by Bill Joy in 1976, a University of California at Berkley student who later went on to co-found Sun Microsystems.
- Most Linux distributions don't include real vi; rather, they ship with an enhanced replacement called vim (which is short for “vi improved”) written by Bram Moolenaar.
- vim is a substantial improvement over traditional Unix vi and is usually symbolically linked (or aliased) to the name “vi” on Linux systems.
- When we refer to “vi”, we will be actually referring to vim.”

Starting and Stopping vi

```
tejas@csye6225:~$
```

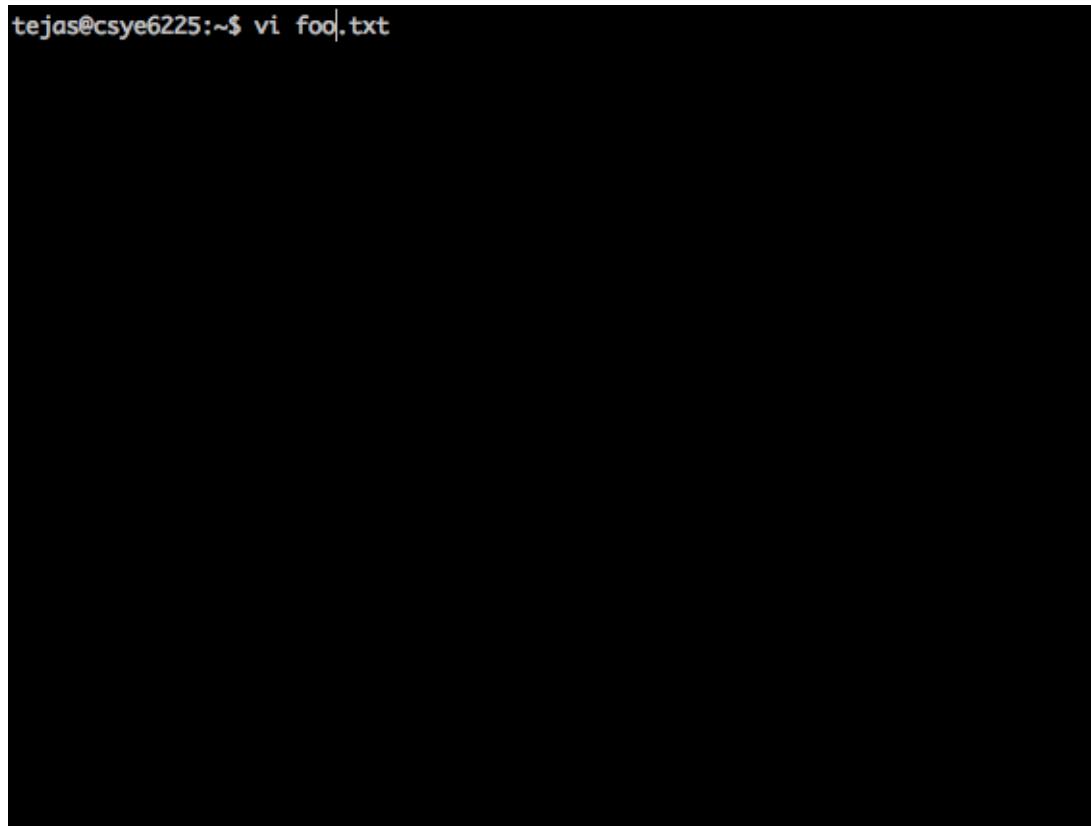


Running in Vi compatible mode

- “Running in Vi compatible mode” means that vim will run in a mode that is closer to the normal behavior of vi rather than the enhanced behavior of vim.
- To run vim with its enhanced behavior, you can do one of the following:
 - Try running *vim* instead of *vi*.
 - Add **alias vi='vim'** to your *.bashrc* file.
 - Add **set nocp** to your *.vimrc* configuration file.
- Different Linux distributions package vim in different ways. Some distributions install a minimal version of vim by default that only supports a limited set of vim features.

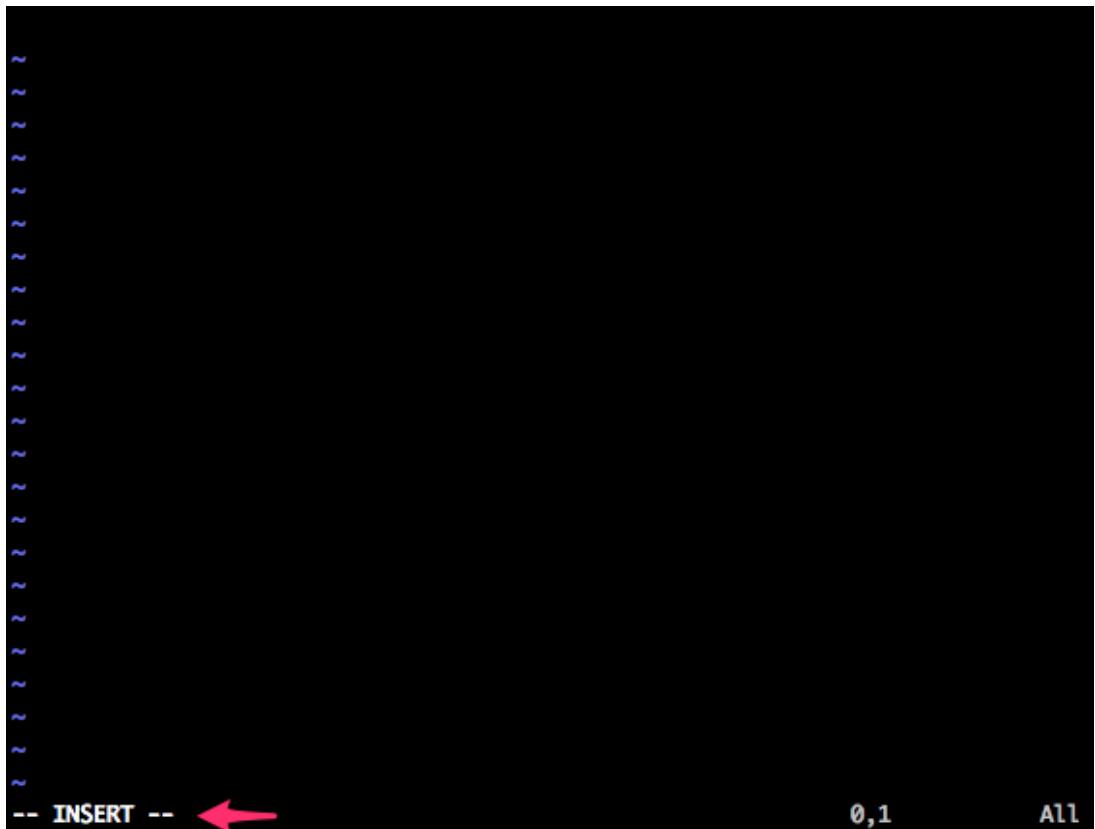
Editing Modes

```
tejas@csye6225:~$ vi foo.txt
```



Entering & Exiting Insert Mode

- In order to add some text to our file, we must first enter *insert mode*. To do this, we press the “i” key.
- To exit insert mode and return to command mode, press the **Esc** key.



Saving File Changes

- To save the change made to our file, we must enter command mode. This is easily done by pressing the ":" key. After doing this, a colon character should appear at the bottom of the screen.
- To write our modified file, we follow the colon with a "w" then Enter.

```
tejas@csye6225:~$
```

Moving The Cursor Around

Key	Moves The Cursor
l or Right Arrow	Right one character.
h or Left Arrow	Left one character.
j or Down Arrow	Down one line.
k or Up Arrow	Up one line.
0 (zero)	To the beginning of the current line.
^	To the first non-whitespace character on the current line.
\$	To the end of the current line.
w	To the beginning of the next word or punctuation character.
W	To the beginning of the next word, ignoring punctuation characters.
b	To the beginning of the previous word or punctuation character.
B	To the beginning of the previous word, ignoring punctuation characters.
Ctrl-f or Page Down	Down one page.
Ctrl-b or Page Up	Up one page.
numberG	To line <i>number</i> . For example, 1G moves to the first line of the file.
G	To the last line of the file.

Hit zero to go to the head/start of the line

“set number” or “set nonumber” -> to set and unset line numbers, useful to figure out which line you are talking about when you are collaborating with someone > so its easy to explain it to them

Appending Text

- Since we will almost always want to append text to the end of a line, vi offers a shortcut to move to the end of the current line and start appending. It's the “A” command. Let's try it and add some more lines to our file.
- First, we'll move the cursor to the beginning of the line using the “0” (zero) command.
- Now we type “A” and add the lines of text.
- Press the Esc key to exit insert mode. As we can see, the “A” command is more useful as it moves the cursor to the end of the line before starting insert mode.

Deleting Text

- vi offers a variety of ways to delete text, all of which contain one of two keystrokes.

Command	Deletes
x	The current character.
3x	The current character and the next two characters.
dd	The current line.
5dd	The current line and the next four lines.
dw	From the current cursor position to the beginning of the next word.
d\$	From the current cursor location to the end of the current line.
d0	From the current cursor location to the beginning of the line.
d^	From the current cursor location to the first non-whitespace character in the line.
dG	From the current line to the end of the file.
d20G	From the current line to the twentieth line of the file.

Cut, Copy & Paste

- The d command not only deletes text, it also “cuts” text. Each time we use the d command the deletion is copied into a paste buffer (think clipboard) that we can later recall with the p command to paste the contents of the buffer after the cursor or the P command to paste the contents before the cursor.
- The y command is used to “yank” (copy) text in much the same way the d command is used to cut text.

Command	Copies
yy	The current line.
5yy	The current line and the next four lines.
yw	From the current cursor position to the beginning of the next word.
y\$	From the current cursor location to the end of the current line.
y0	From the current cursor location to the beginning of the line.
y^	From the current cursor location to the first non-whitespace character in the line.
yG	From the current line to the end of the file.
y20G	From the current line to the twentieth line of the file.

Search & Replace (Searching The Entire File)

- To move the cursor to the next occurrence of a word or phrase, the / command is used.
 - When you type the / command a "/" will appear at the bottom of the screen. Next, type the word or phrase to be searched for, followed by the Enter key. The cursor will move to the next location containing the search string.
 - A search may be repeated using the previous search string with the n command.

Global Search-And-Replace

:%s/Line/line/g

Item	Meaning
:	The colon character to enter command mode
%	Specifies the range of lines for the operation. % is a shortcut meaning from the first line to the last line.
s	Specifies the operation. In this case, substitution (search-and-replace)
/Line/line/	The search pattern and the replacement text.
g	This means “global” in the sense that the search-and-replace is performed on every instance of the search string in the line. If omitted, only the first instance of the search string on each line is replaced.

Save a file (that requires root permissions)
from within vim editor

- <https://www.cyberciti.biz/faq/vim-vi-text-editor-save-file-without-root-permission/>

Additional Resources

See Lecture Page

Linux Shell Programming

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225

Northeastern University

What Are Shell Scripts?

- In the simplest terms, a shell script is a file containing a series of commands.
- The shell reads this file and carries out the commands as though they have been entered directly on the command line.
- Shell scripts and functions are both interpreted. This means they are not compiled.
- Shell scripts and functions are ASCII text that is read by shell command interpreter.

Why Write Scripts?

- Good system administrators write scripts.
- Scripts standardize and automate the performance of administrative chores and free up admins' time for more important and more interesting tasks.
- In a sense, scripts are also a kind of low-rent documentation in that they act as an authoritative outline of the steps needed to complete a particular task.
- The shell is always available, so shell scripts are relatively portable and have few dependencies other than the commands they invoke.

Case Sensitivity

- Linux is case sensitive.

Special Characters

- All of the following characters have a special meaning or function. If they are used in a way that their special meaning is not needed, they must be **escaped**.
- To escape, or remove its special function, the character must be immediately preceded with a backslash.
- `\ / ; , . ~ # $? & * () [] ' " + - ! ^ = | < >`

In shell script, don't use ‘ over “”, because when a string is inside single quote, then that string is not evaluated and it won't read the variables, \${FIRST_NAME} for single quote and “Vishnu” for double quotes

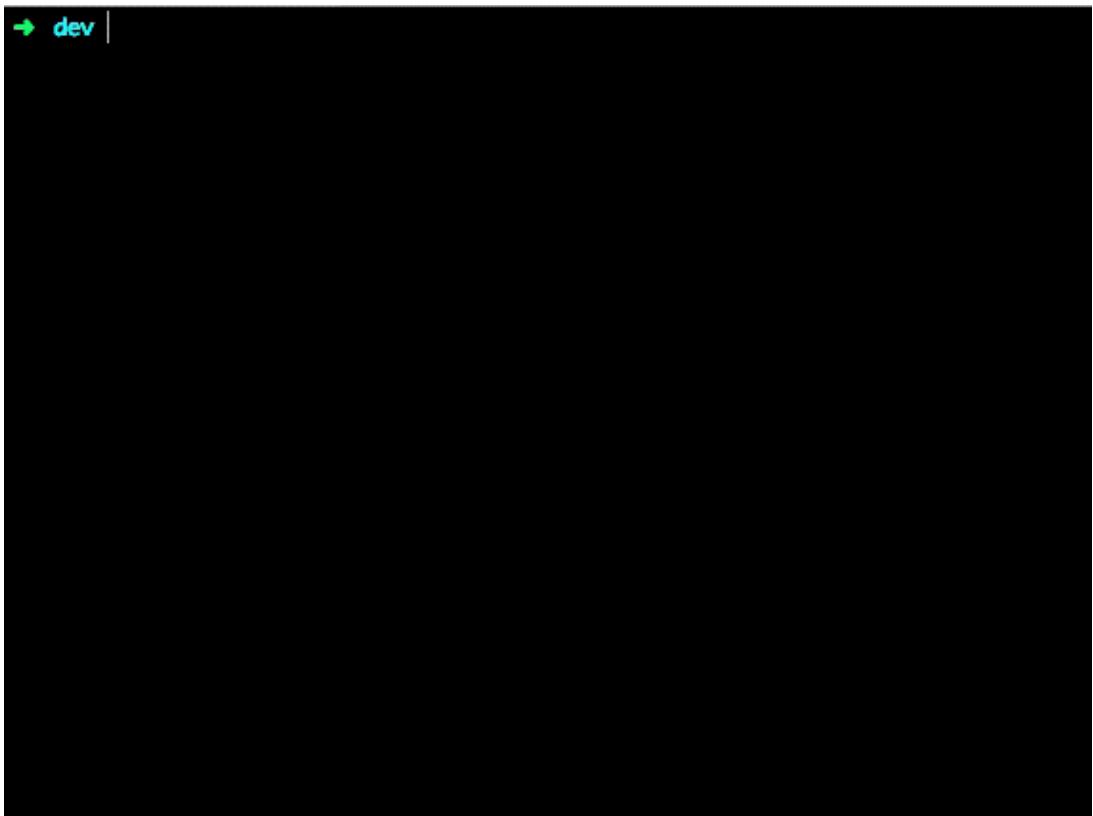
Difference between “” and “-> if you write a string in any language within “”, that means that the programming language evaluates the string to check if there are any variables in the string. v/s, “-> where the string inside “ is not evaluated!

Writing a Shell Script

To create a shell script:

1. Use a text editor such as vi.
Write required Linux commands
and logic in the file.
2. Save and close the file (exit from
vi).
3. Make the script executable.
4. Run the script.

`#!/bin/bash` - first line indicates, that it should be run with
bash script - or else it will take user set shell, its a
comment



Declare the Shell in the Shell Script

- The `#!` syntax used in scripts to indicate an interpreter for execution under UNIX / Linux operating systems.
- If no shell is declared, the script will execute in the *default* shell, defined by the system for the user executing the shell script.
- Shell declaration statement must appear on the *first line* of the shell script.
- Linux shell script that uses bash shell will starts with the following line:

```
#!/bin/bash
```

Comments in Shell Scripts

- Command readability and step-by-step comments are just the very basics of a well-written script.
- Using a lot of comments will make our life much easier when we have to come back to the code after not looking at it for six months, and believe me; we will look at the code again.
- Comment everything! This includes, but is not limited to, describing what our variables and files are used for, describing what loops are doing, describing each test, maybe including expected results and how we are manipulating the data and the many data fields.
- A hash mark, #, precedes each line of a comment.

Variables

Variables are all uppercase, separated with underscore_
ex: FIRST_NAME

- A *variable* is a character string to which we assign a value. The value assigned could be a number, text, filename, device, or any other type of data. A variable is nothing more than a pointer to the actual data.
- Creating and setting variables within a script is fairly simple. Use the following syntax:

varName=someValue

Do not add spaces in your expressions

someValue is assigned to given **varName** and **someValue** must be on right side of = (equal) sign. If **someValue** is not given, the variable is assigned the null string.

- You can display the value of a variable with **echo \$varName** or **echo \${varName}**

Quoting

- Your bash shell understands special characters with special meanings. For example, \$var is used to expand the variable value. Bash expands variables and wildcards, for example:

```
echo "$PATH"  
echo "$PS1"  
echo /etc/*.conf
```

- Sometime you do not wish to use variables or wildcards. For example, do not print value of \$PATH, but just print \$PATH on screen as a word. You can enable or disable the meaning of a special character by enclosing them in single quotes. This is also useful to suppress warnings and error messages while writing the shell scripts.

echo "Path is \$PATH" ## \$PATH will be expanded

OR

echo 'I want to print \$PATH' ## PATH will not be expanded

Perform arithmetic operations

- Arithmetic expansion and evaluation is done by placing an integer expression using the following format:

```
$((expression))  
$(( n1+n2 ))  
$(( n1/n2 ))  
$(( n1-n2 ))
```

- Add two numbers using x and y variable. Create a shell program called add.sh using a text editor:

```
#!/bin/bash  
x=5  
y=10  
ans=$(( x + y ))  
echo "$x + $y = $ans"
```

Bash variable existence check

In modern bash (version 4.2 and above):

```
if [[ -v name_of_var ]]  
then  
    echo "set"  
else  
    echo "not set"
```

-v VAR, True if the shell variable VAR is set

If..else..fi

if..then..else Syntax

```
if command
then
    command executed successfully
    execute all commands up to else statement
    or to fi if there is no else statement

else
    command failed so
    execute all commands up to fi
fi
```

Nested ifs

You can put if command within if command and create the nested ifs as follows:

```
if condition
then
    if condition
    then
        ....
        ..
        do this
    else
        ....
        ..
        do this
    fi
else
    ...
    ....
    do this
fi
```

for . . . in statement

```
for loop_variable in argument_list
do
```

```
    commands
```

```
done
```

while statement

```
while test_condition_is_true  
do  
    commands  
done
```

case statement

We use case statements a lot in the shell scripting.
Case is like switch from python.

```
case $variable in
    match_1)
        commands_to_execute_for_1
    ;;
    match_2)
        commands_to_execute_for_2
    ;;
    match_3)
        commands_to_execute_for_3
    ;;
    .
    .
    .
    *)
        (Optional - any other value)
        commands_to_execute_for_no_match
    ;;
esac
```

Command-Line Arguments

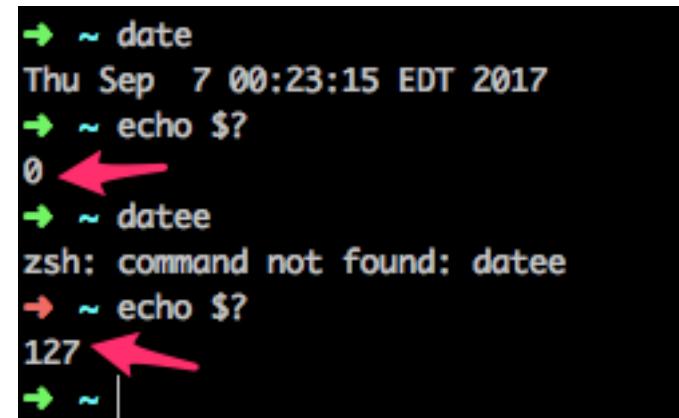
- The command-line arguments $\$1, \$2, \$3, \dots, \9 are positional parameters, with $\$0$ pointing to the actual command, program, shell script, or function and $\$1, \$2, \$3, \dots, \9 as the arguments to the command.
- The positional parameters, $\$0, \2 , and so on in a function are for the function's use and may not be in the environment of the shell script that is calling the function.
- Where a variable is known in a function or shell script is called the scope of the variable.

Special Parameters \$* and \$@

- There are special parameters that allow accessing all the command-line arguments at once.
- \$* and \$@ both will act the same unless they are enclosed in double quotes,"".
- The \$* special parameter specifies all command-line arguments.
- The \$@ special parameter also specifies all command-line arguments.
- The "\$*" special parameter takes the entire list as one argument with spaces between.
- The "\$@" special parameter takes the entire list and separates it into separate arguments.

The exit status of a command

- Each Linux command returns a status when it terminates normally or abnormally. You can use command exit status in the shell script to display an error message or take some sort of action.
- Exit Status
 - Every Linux command executed by the shell script or user, has an exit status.
 - The exit status is an integer number.
 - The Linux man pages stats the exit statuses of each command.
 - 0 exit status means the command was successful without any errors.
 - A non-zero (1-255 values) exit status means command was failure.
 - You can use special shell variable called ? to get the exit status of the previously executed command.



A screenshot of a terminal window showing command execution and exit status. The session starts with 'date' (exit status 0), followed by 'echo \$?' (exit status 0). Then, 'datee' is run, resulting in an error message 'zsh: command not found: datee' and an exit status of 127. Finally, 'echo \$?' is run again, showing the exit status 127. Two pink arrows point to the exit status values '0' and '127'.

```
→ ~ date
Thu Sep  7 00:23:15 EDT 2017
→ ~ echo $?
0
→ ~ datee
zsh: command not found: datee
→ ~ echo $?
127
→ ~ |
```

Check the Return Code (Script Example)

```
test -d /usr/local/bin
if [ "$?" -eq 0 ] # Check the return code
then          # The return code is zero
    echo '/usr/local/bin does exist'
else          # The return code is NOT zero
    echo '/usr/local/bin does NOT exist'
fi
```

Conditional Execution

- You can link two commands under bash shell using conditional execution based on the exit status of the last command. This is useful to control the sequence of command execution. Also, you can do conditional execution using the if statement.
- The bash shell support the following two conditional executions:
 1. Logical AND && - Run second command only if first is successful.
 2. Logical OR || - Run second command only if first is not successful.

For loop and While loop

- Bash shell can repeat particular instruction again and again, until particular condition satisfies.
- A group of instruction that is executed repeatedly is called a loop.
- Bash supports:
 - The for loop
 - The while loop

Linking Commands

Under bash you can create a sequence of one or more commands separated by one of the following operators

Operator	Syntax	Description	Example
;	command1; command2	Separates commands that are executed in sequence.	In this example, pwd is executed only after date command completes. date ; pwd
&	command arg &	The shell executes the command in the background in a subshell. The shell does not wait for the command to finish, and the return status is 0. The & operator runs the command in background while freeing up your terminal for other work.	In this example, find command is executed in background while freeing up your shell prompt. find / -iname "*.pdf" >/tmp/output.txt &
&&	command1 && command2	command2 is executed if, and only if, command1 returns an exit status of zero i.e. command2 only runs if first command run successfully.	[! -d /backup] && mkdir -p /backup See Logical AND section for examples.
	command1 command2	command2 is executed if and only if command1 returns a non-zero exit status i.e. command2 only runs if first command fails.	tar cvf /dev/st0 /home mail -s 'Backup failed'you@example.com </dev/null See Logical OR section for examples.
	command1 command2	Linux shell pipes join the standard output of command1 to the standard input of command2.	In this example, output of the ps command is provided as the standard input to the grep command ps aux grep httpd

Shell Functions

- Shell functions
- Sometime shell scripts get complicated.
- To avoid large and complicated scripts use functions.
- You divide large scripts into a small chunks/entities called **functions**.
- Functions makes shell script modular and easy to use.
- Function avoids repetitive code. For example, `is_root_user()` function can be reused by various shell scripts to determine whether logged on user is root or not.
- Function performs a specific task. For example, add or delete a user account.
- Function used like normal command.
- In other high level programming languages function is also known as procedure, method, subroutine, or routine.

```
hello() { echo 'Hello world!' ; }
```

Trap Statement

- While running a script user may press Break or CTRL+C to terminate the process.
- User can also stop the process by pressing CTRL+Z.
- Error can occur due to bug in a shell script such as arithmetic overflow.
- This may result into errors or unpredictable output.
- Whenever user interrupts a signal is sent to the command or the script.
- Signals force the script to exit.
- However, the trap command captures an interrupt.
- The trap command provides the script to capture an interrupt (signal) and then clean it up within the script.

```
#!/usr/bin/env bash
CWD=`pwd`
TMP=${TMP:-/tmp/tmpdir}

trap \
  '{ /usr/bin/rm -r "${TMP}" ; exit 255; }' \
  SIGINT SIGTERM ERR EXIT

## create tmp dir
mkdir "${TMP}"
tar xf "${1}" --directory "${TMP}"

## move to tmp and run commands
pushd "${TMP}"
for IMG in *.jpg; do
  mogrify -verbose -flip -flop "${IMG}"
done
tar --create --file "${1%.*}".tar *.jpg

## move back to origin
popd

## zip tar
bzip2 --compress ${TMP}/${1%.*}.tar \
--stdout > "${1%.*}.tbz"
```

Additional Resources

See Lecture Page

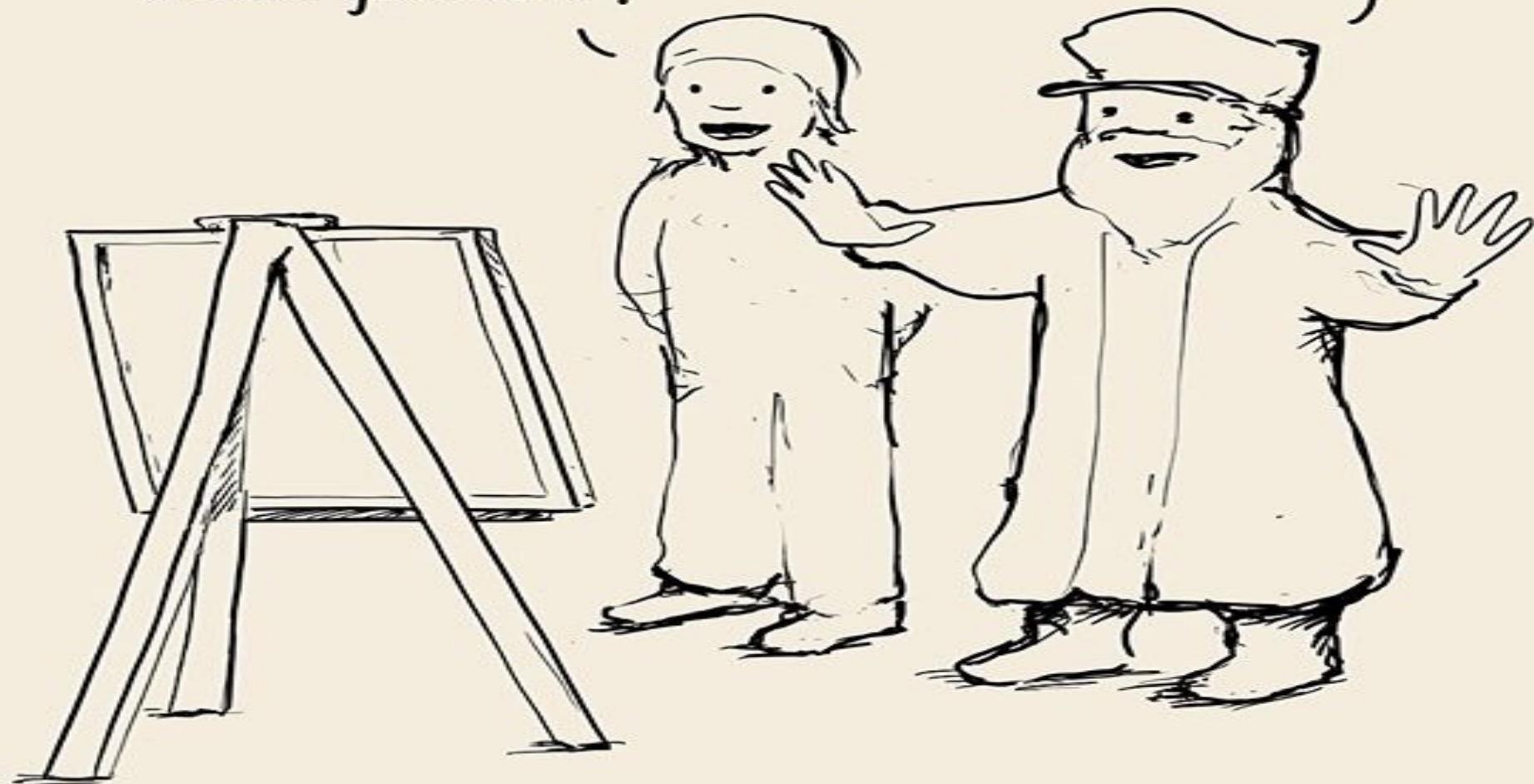
Version Control with Git

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University

beautiful!
what do you call it?

mona_lisa_finalrealupdateFINALL6



What is Version Control & Why should you care?

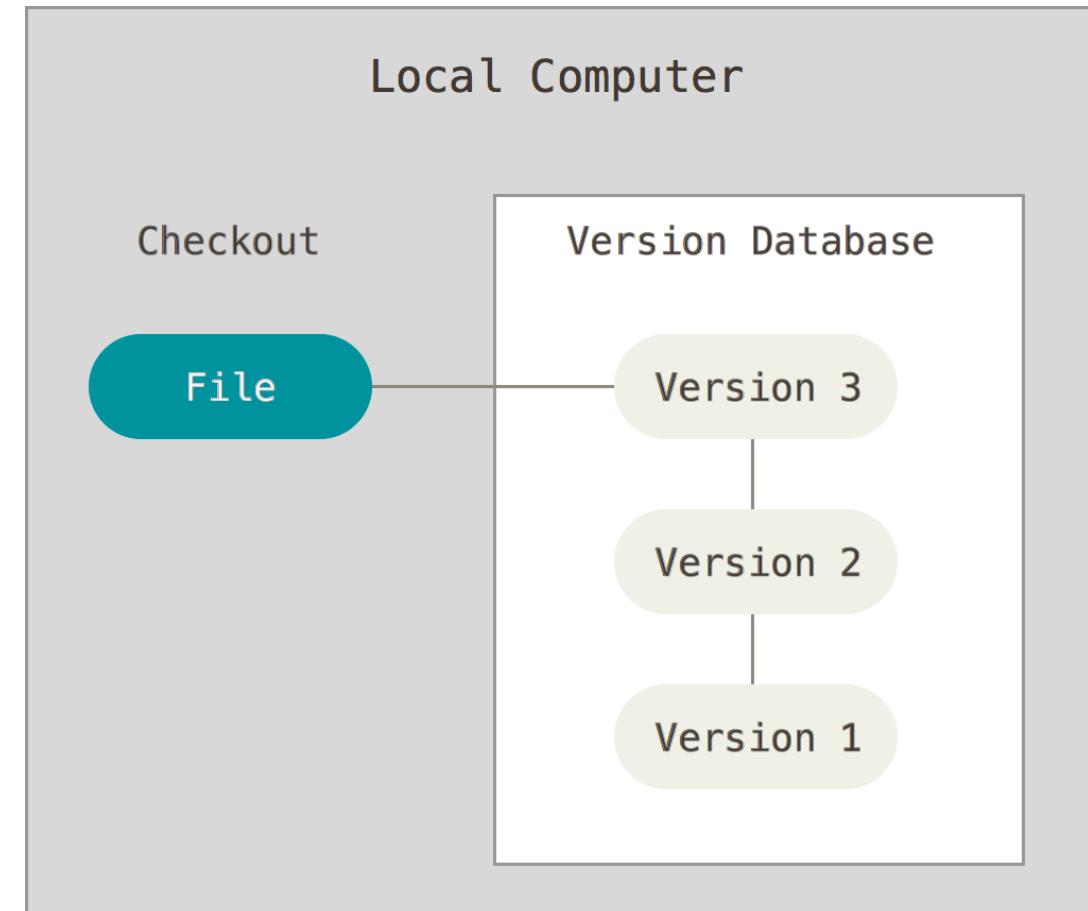
- Version Control System track changes to your files and folders over time so that you can recall specific versions later.
- With a Version Control System you can compare various revisions of a file or revert back to an older version if there is a problem with latest version of a file.
- Version Control System also lets you track WHO made the changes for a particular version which is useful if you are working in teams.



Local Version Control

Example:

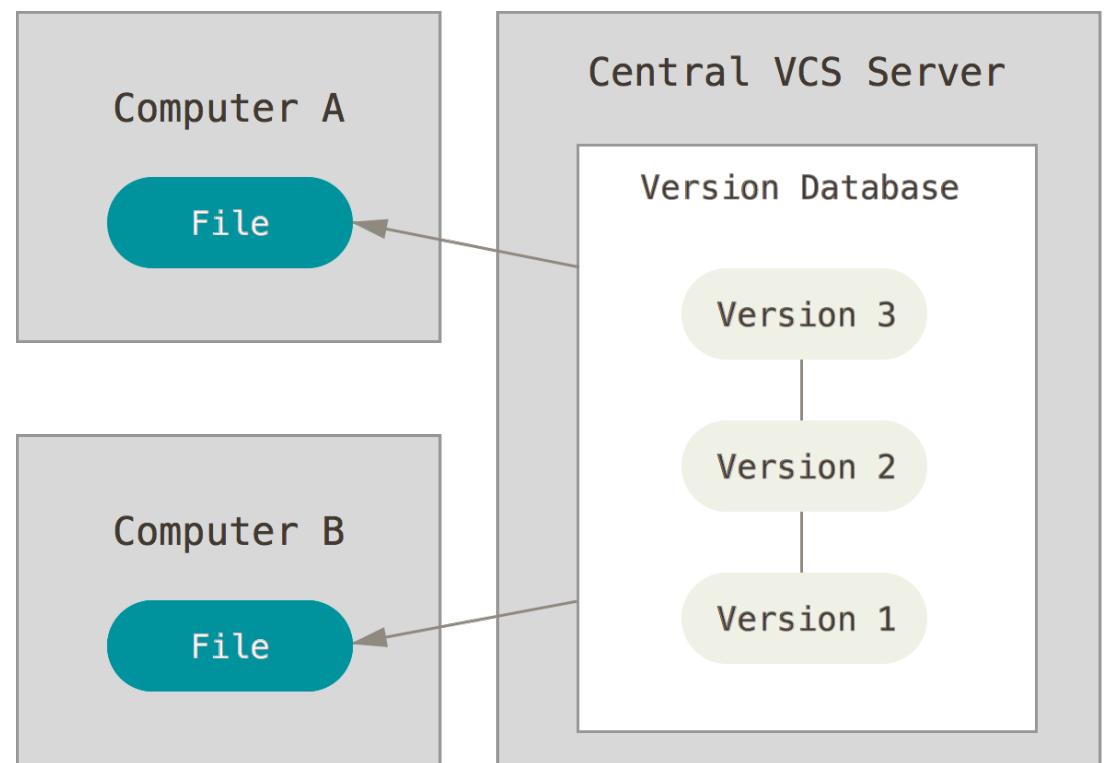
- Revision Control System (RCS)
- Source Code Control System (SCCS)



Centralized Version Control

Example:

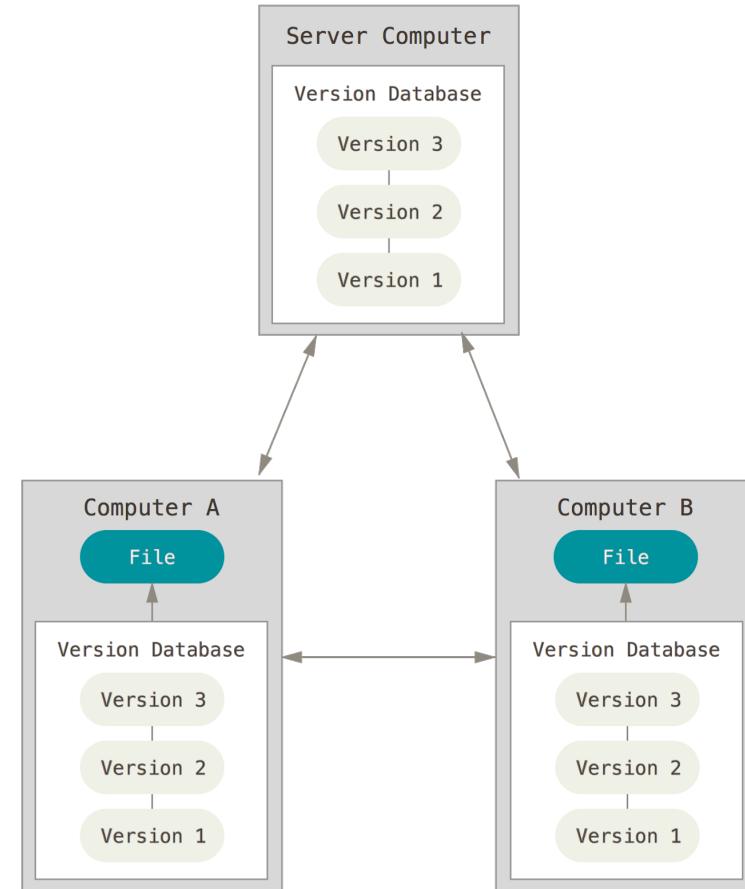
- CVS
- SVN
- Perforce



Distributed Version Control Systems

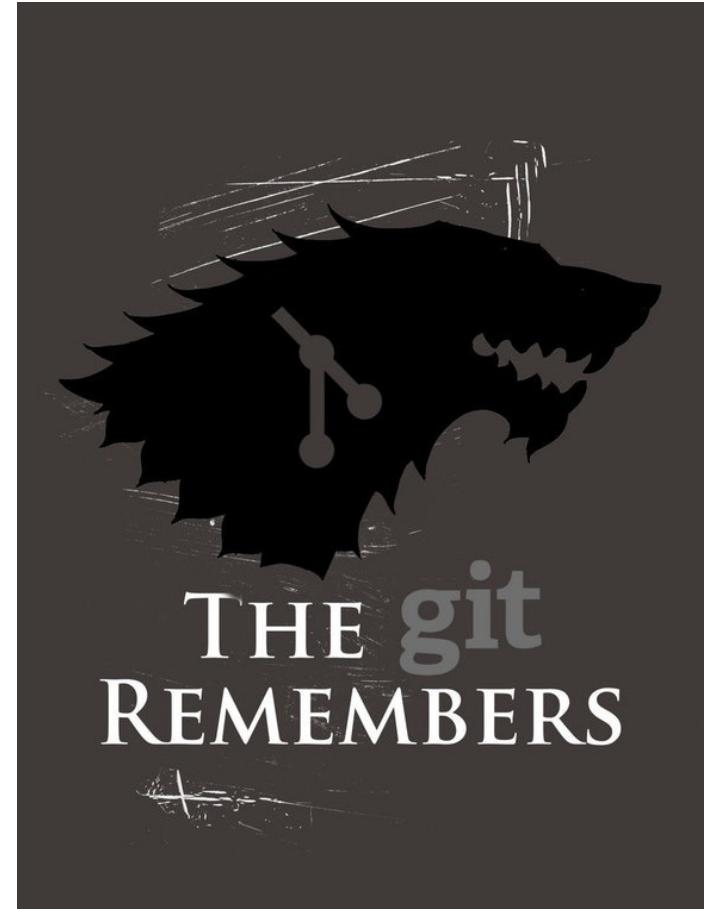
Example

- Git
- Mercurial
- Bazaar
- Darcs

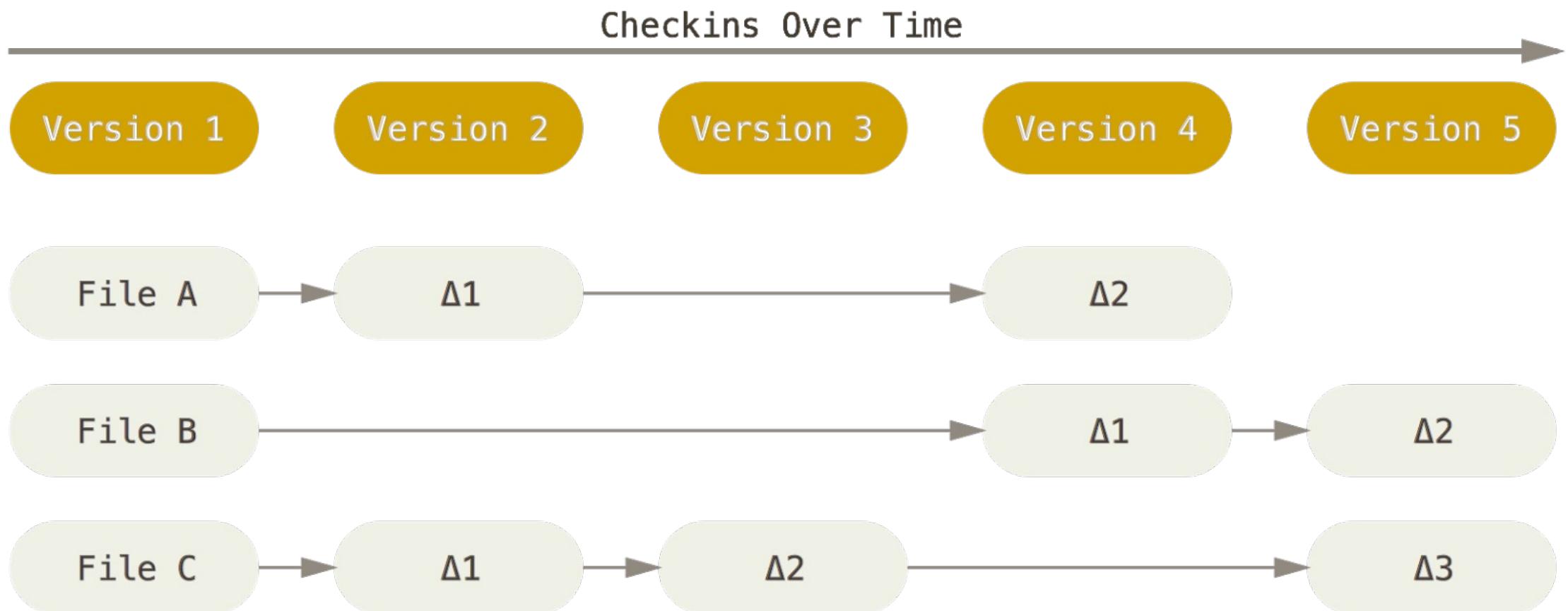


A Short History of Git

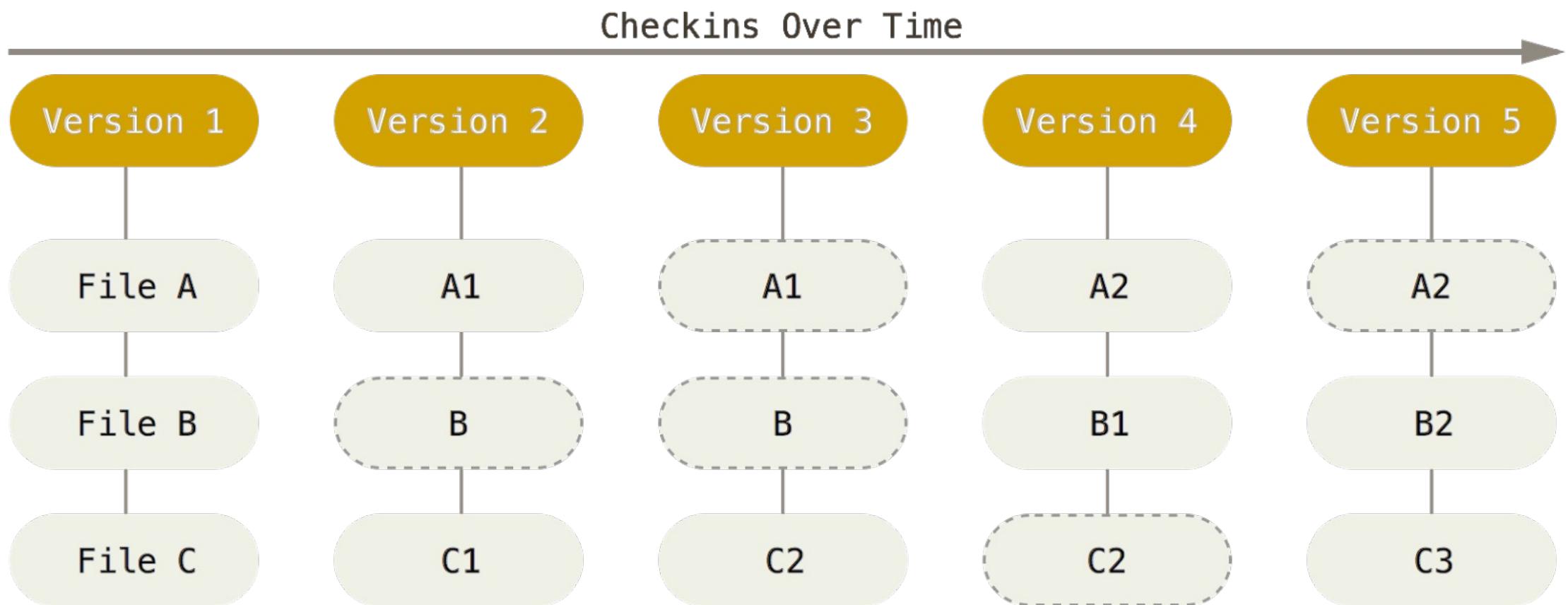
- Linux kernel project began using a proprietary DVCS called BitKeeper in 2002 but in 2005 the relationship between community & commercial company broke down and BitKeeper was no longer free for the community.
- Linus Torvalds created Git in 2005 to manage Linux Kernel. Since then Git has become the goto Distributed Version Control System for developers.
- In this course we will use Git and Github for hosting assignment and term project.



Tracking Changes w/Differences (CVS, SVN)



Tracking Changes w/Snapshots (Git)



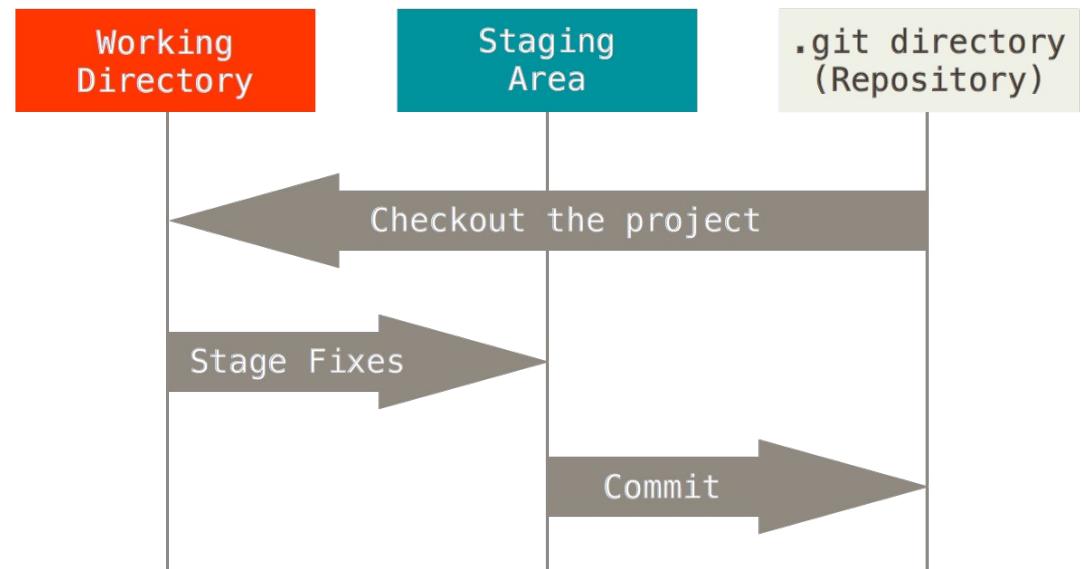
Git Has Integrity

- Everything in Git is check-summed before it is stored and is then referred to by that checksum.
- Once a file has been committed, it is impossible to change the contents of file or directory without Git finding out.
- Git uses SHA-1 hash which is a 40-character string composed of hexadecimal characters (0–9 and a–f) and calculated based on the contents of a file or directory structure in Git. Example commit hash
fb1d8e0e2c50f374cfcc244564decfc3f0a336cb4
- Git stores everything in its database not by file name but by the hash value of its contents which means you will see these hash values all over the place.

Local Git Workflow

The basic Git workflow is something like this:

- You modify files in your working tree.
- You stage the files, adding snapshots of them to your staging area.
- You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.



Installing Git

You can download the Git binaries from <https://git-scm.com/downloads>

THIS IS GIT. IT TRACKS COLLABORATIVE WORK
ON PROJECTS THROUGH A BEAUTIFUL
DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZIZE THESE SHELL
COMMANDS AND TYPE THEM TO SYNC UP.
IF YOU GET ERRORS, SAVE YOUR WORK
ELSEWHERE, DELETE THE PROJECT,
AND DOWNLOAD A FRESH COPY.



First-Time Git Setup (your identity & editor)

Set your name and address using following commands

- **\$ git config --global user.name "John Doe"**
- **\$ git config --global user.email johndoe@example.com**

Set your default editor on Linux using following command.

- **\$ git config --global core.editor vim**
- The default text editor that will be used when Git needs you to type in a message.

You can modify the settings later in **~/.gitconfig**

GitHub SSH key Setup

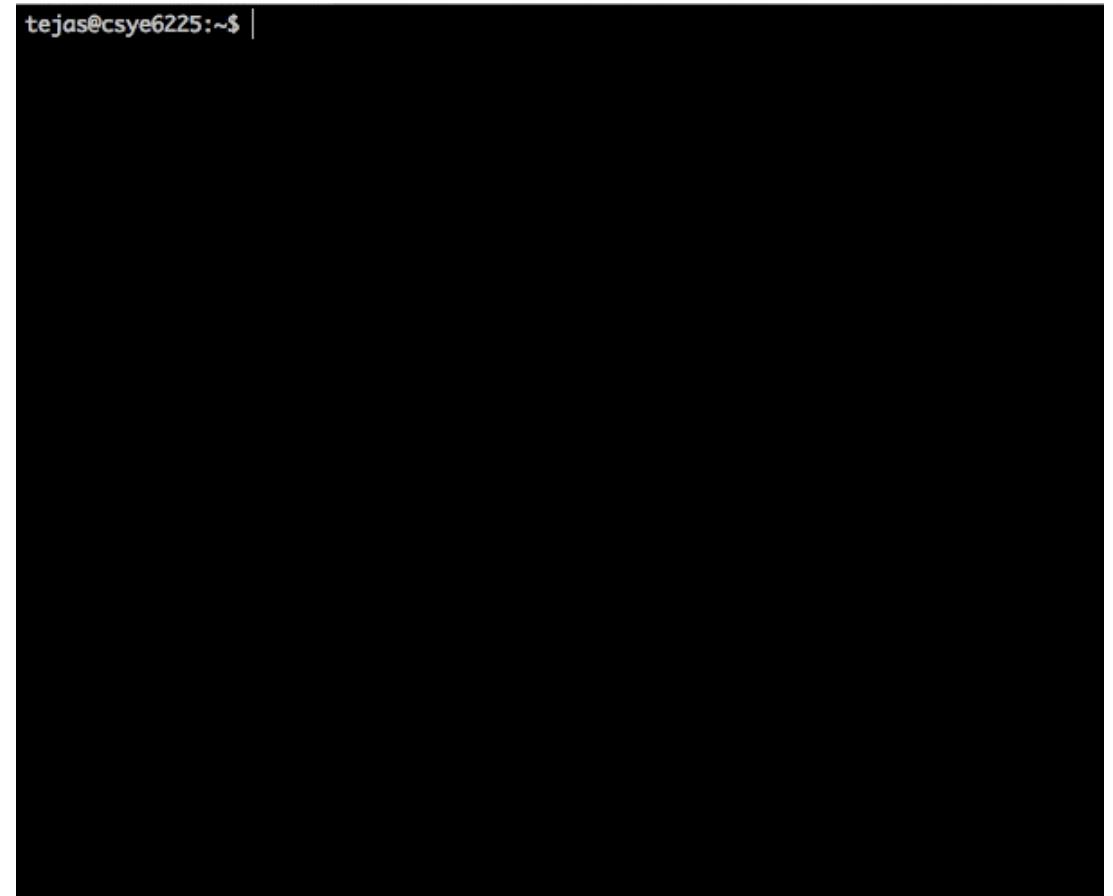
Follow steps documented in the articles below:

1. <https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/>
2. <https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/>

Initializing a Repository in Existing Directory

```
$ cd /home/user/your_repo  
$ git init
```

This creates a new subdirectory named **.git** that contains all of your necessary repository files – a Git repository skeleton. At this point, nothing in your project is tracked yet.



Cloning an Existing Repository

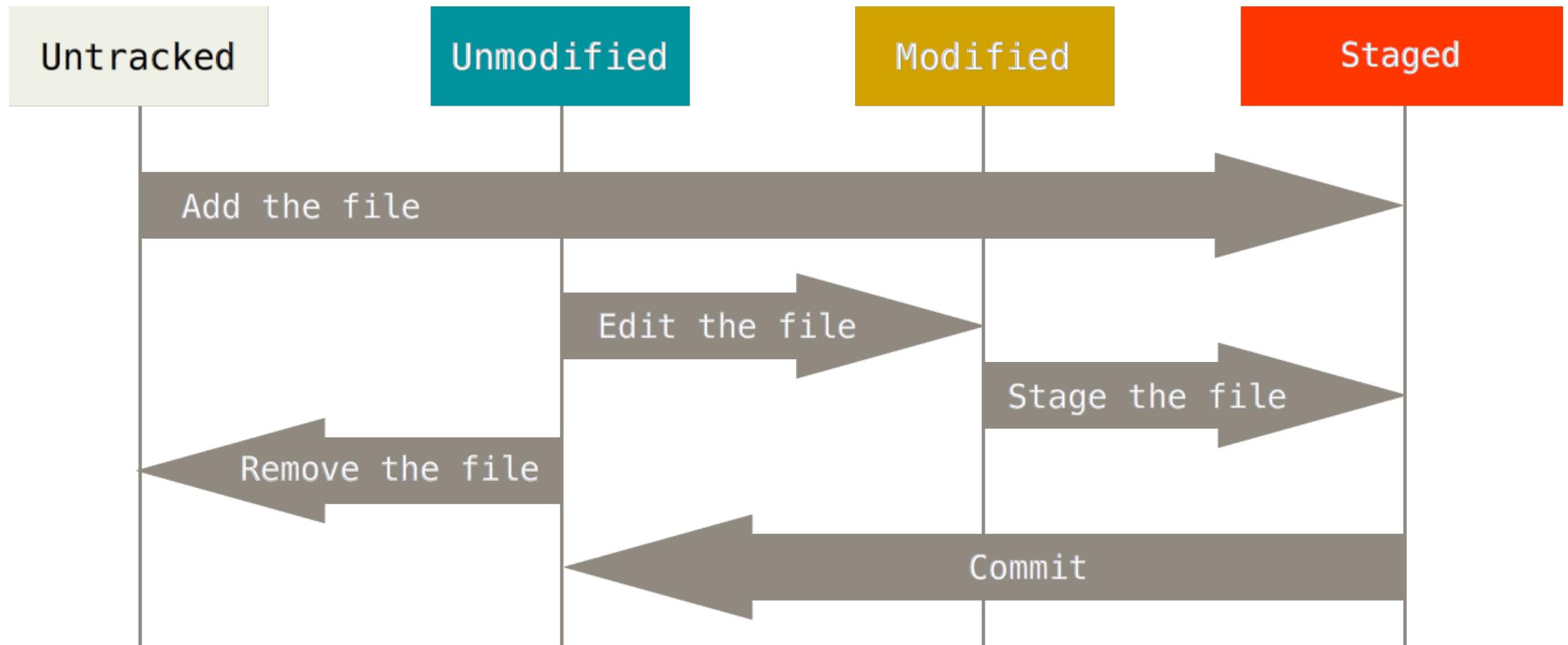
- If you want to get a copy of existing Git repository you need to use **git clone**

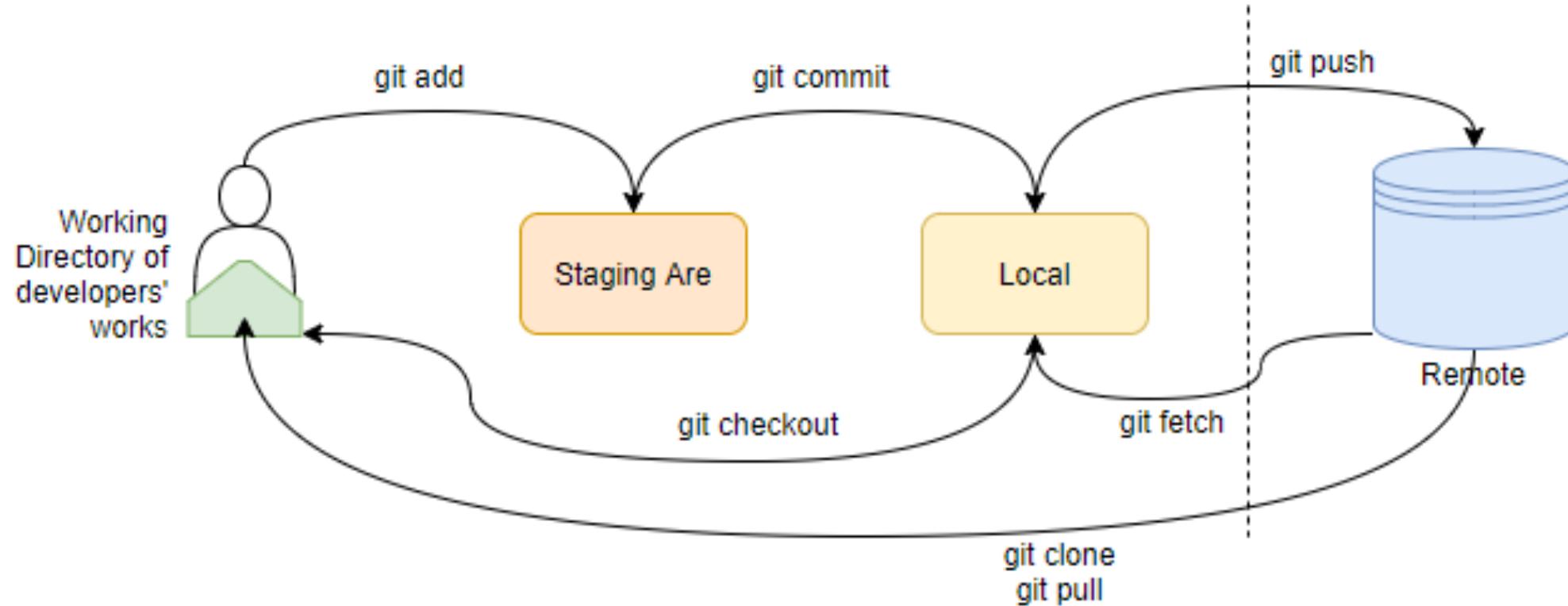
```
$ git clone git@github.com:torvalds/linux.git
```

- Note that you can clone git repository using either **git** (SSH) or **https** transfer protocol. I recommend you use **git** protocol wherever possible.



Lifecycle of the Status of your files





I DONT KNOW ABOUT

THEM FANCY GIT COMMANDS

Git Status

Displays paths that have differences between the index file and the current HEAD commit, paths that have differences between the working tree and the index file, and paths in the working tree that are not tracked by Git.

```
tejas@csye6225:~/vimrc$ git status
On branch master
Your branch is up-to-date with 'origin/master'. No changes
nothing to commit, working directory clean
tejas@csye6225:~/vimrc$ touch test.txt
tejas@csye6225:~/vimrc$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    test.txt          Added new file

nothing added to commit but untracked files present (use "git add" to track)
tejas@csye6225:~/vimrc$ vi vimrc
tejas@csye6225:~/vimrc$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   vimrc          Modified existing file

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    test.txt

no changes added to commit (use "git add" and/or "git commit -a")
tejas@csye6225:~/vimrc$ |
```

Tracking New File with Git

A new file must be added to Git repo using the command **git add <FILENAME>**. You can track all new files using -A

Example:

```
$ git add -A :/
```

Stage Modified Files

- Existing files must be staged using the same **git add** command.

Committing Your Changes (Locally)

Once you have staged all your new and modified files, it is time to commit them using the **git commit** command.

```
$ git commit -m "commit message goes here"
```



<https://xkcd.com/1296/>

COMMENT	DATE
CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
ENABLED CONFIG FILE PARSING	9 HOURS AGO
MISC BUGFIXES	5 HOURS AGO
CODE ADDITIONS/EDITS	4 HOURS AGO
MORE CODE	4 HOURS AGO
HERE HAVE CODE	4 HOURS AGO
AAAAAAA	3 HOURS AGO
ADKFJSLKDFJSDFKLJ	3 HOURS AGO
MY HANDS ARE TYPING WORDS	2 HOURS AGO
HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Git commits
1st line -> < 50 chars
Next few lines explain
(do not put the whole content in the first line)

Working with Remotes

- Remote repositories are versions of your project that are hosted on the Internet or network somewhere.
- To add a new remote Git repository use following command
- **git remote add alias <url>**
- Note: A git repository can have more than one remote.

Showing Your Remotes

- **git remote -v** command will show you which remote server you have configured.

*“git pull a day keeps
the conflicts away”*

Fetching and Pulling from Your Remotes

- To get data (changes) from remote project, you can use **git fetch** or **git pull** command.
- **git fetch** command only downloads the data to your local repository – it doesn't automatically merge it with any of your work or modify what you're currently working on. You have to merge it manually into your work when you're ready.
- **git pull** command automatically fetches and then merges remote branch into your current branch.

Pushing to Remotes

git push [remote-name] [branch-name] command is used to push committed changes from your local git repository to the one the server so others can pull it.

Example:

```
$ git push origin main
```

Git Branches

Nearly every VCS has some form of branching support. Branching means you diverge from the main line of development and continue to do work without messing with that main line. In many VCS tools, this is a somewhat expensive process, often requiring you to create a new copy of your source code directory, which can take a long time for large projects.

Create & Checkout Git Branch

To create a branch and switch to it at the same time, you can run the git checkout command with the -b switch

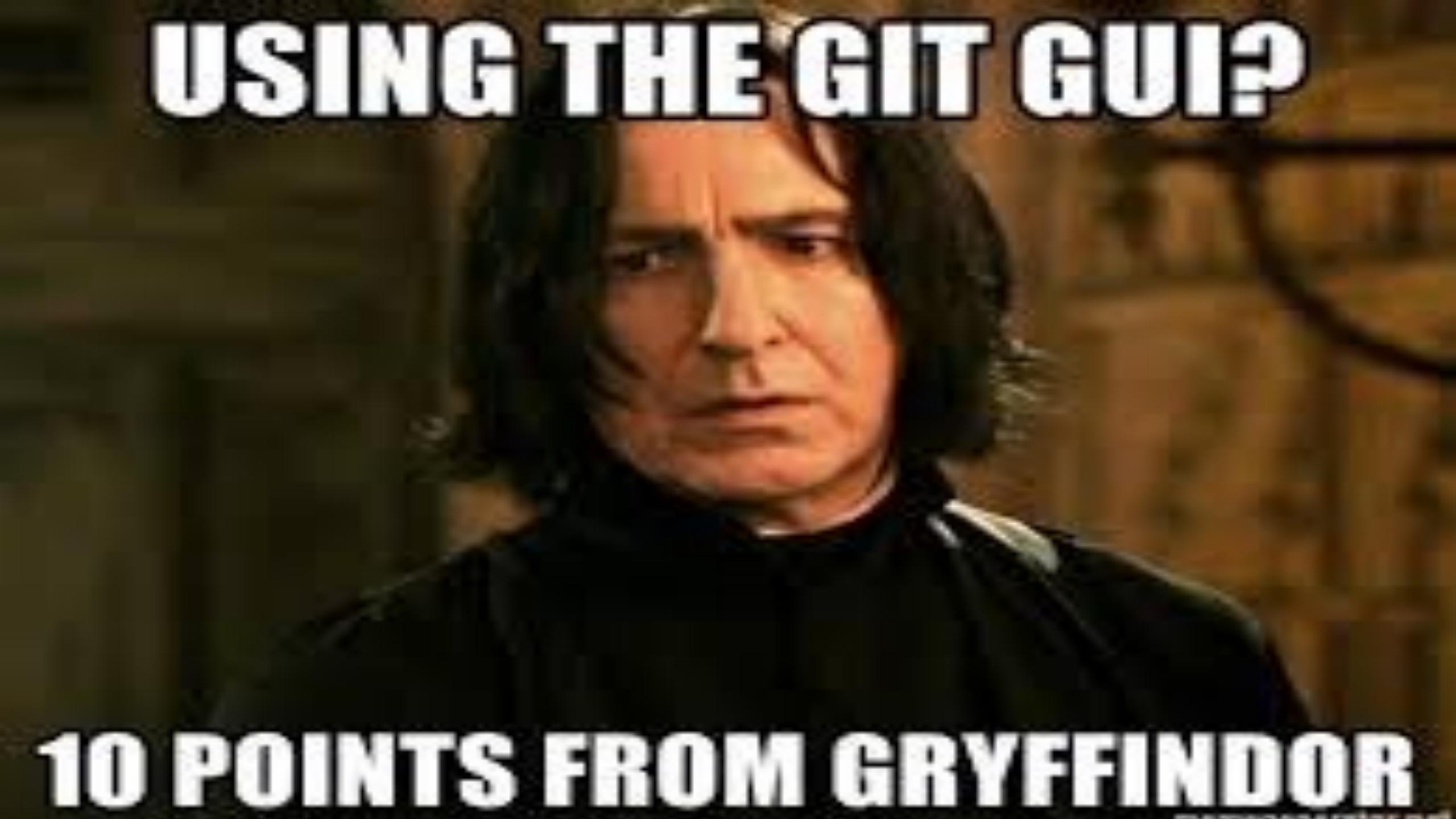
\$ git checkout -b BRANCH_NAME

This is shorthand for:

\$ git branch BRANCH_NAME

\$ git checkout BRANCH_NAME

USING THE GIT GUI?



10 POINTS FROM GRYFFINDOR

Github Pull Request

<https://help.github.com/articles/about-pull-requests/>



Luca Lanziani 🌎🔥
@lucalanziani

...

"Ask a programmer to review ten lines of code, he'll find ten issues. Ask him to do five hundred lines, and he'll say it looks good." - Gene Kim, The DevOps Handbook.

Merge Conflicts



I Am Devloper
@iamdevloper



ever see a git merge conflict so bad
you're convinced Netflix are gonna
make a true crime mini-series out of it?

9/7/18, 6:14 AM

GIT MERGE

A wide-angle photograph of a vast, flat landscape, likely a coastal plain or a large field. The foreground is a uniform, light-yellowish-green color, suggesting dry grass or sand. In the middle ground, there are low, dark, scrubby bushes or small trees scattered across the horizon. The background is a clear, pale blue sky with no visible clouds.

In case of fire



1. git commit



2. git push



3. leave building

Forking Workflow

<https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow>

Additional Resources

See Lecture Page

GitHub Actions

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225

Northeastern University



GitHub Actions

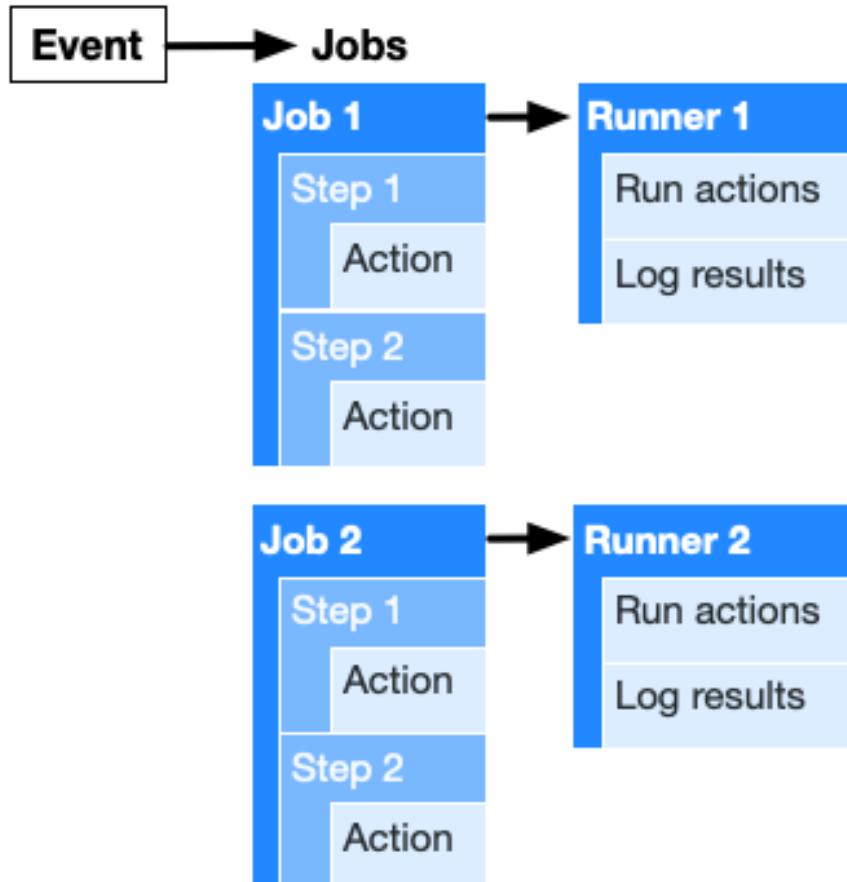
Automate your workflow from idea to production

GitHub Actions makes it easy to automate all your software workflows, now with world-class CI/CD. Build, test, and deploy your code right from GitHub. Make code reviews, branch management, and issue triaging work the way you want.

GitHub Actions

- GitHub Actions help you automate tasks within your software development life cycle.
- GitHub Actions are event-driven, meaning that you can run a series of commands after a specified event has occurred.
- For example, every time someone creates a pull request for a repository, you can automatically run a command that executes a software testing script.

The components of GitHub Actions



- Workflows
- Events
- Jobs
- Steps
- Actions
- Runners

Workflow

- The workflow is an automated procedure that you add to your repository.
- Workflows are made up of one or more jobs and can be scheduled or triggered by an event.
- The workflow can be used to build, test, package, release, or deploy a project on GitHub.

Events

- An event is a specific activity that triggers a workflow.
- Examples
 - Activity can originate from GitHub when someone pushes a commit to a repository or when an issue or pull request is created.
 - You can also use the repository dispatch webhook to trigger a workflow when an external event occurs.

Events That Trigger Workflows

- Scheduled events
 - schedule
- Manual events
 - workflow_dispatch
 - repository_dispatch
- Webhook events
 - check_run
 - check_suite
 - create
 - delete
 - deployment
 - deployment_status
 - fork
 - gollum
- Webhook events (contd.)
 - issue_comment
 - issues
 - label
 - milestone
 - page_build
 - project
 - project_card
 - project_column
 - public
 - pull_request
 - pull_request_review
 - pull_request_review_comment
 - pull_request_target
 - push
 - registry_package
 - release
 - status
 - watch
 - workflow_run

Jobs

- A job is a set of steps that execute on the same runner.
- By default, a workflow with multiple jobs will run those jobs in parallel.
- You can also configure a workflow to run jobs sequentially.
- Example
 - A workflow can have two sequential jobs that build and test code, where the test job is dependent on the status of the build job. If the build job fails, the test job will not run.

Steps

- A step is an individual task that can run commands in a job.
- A step can be either an action or a shell command.
- Each step in a job executes on the same runner, allowing the actions in that job to share data with each other.

Actions

- Actions are standalone commands that are combined into steps to create a job.
- Actions are the smallest portable building block of a workflow.
- You can create your own actions, or use actions created by the GitHub community.
- To use an action in a workflow, you must include it as a step.

Runners

- A runner is a server that has the GitHub Actions runner application installed.
- You can use a runner hosted by GitHub, or you can host your own.
- A runner listens for available jobs, runs one job at a time, and reports the progress, logs, and results back to GitHub.
- GitHub-hosted runners are based on Ubuntu Linux, Microsoft Windows, and macOS, and each job in a workflow runs in a fresh virtual environment.

Writing GitHub Actions

- GitHub Actions uses **YAML** syntax to define the events, jobs, and steps.
- GitHub Actions YAML files are stored in your code repository, in a directory called **.github/workflows**.

Sample Workflow

```
name: learn-github-actions
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v1
      - run: npm install -g bats
      - run: bats -v
```

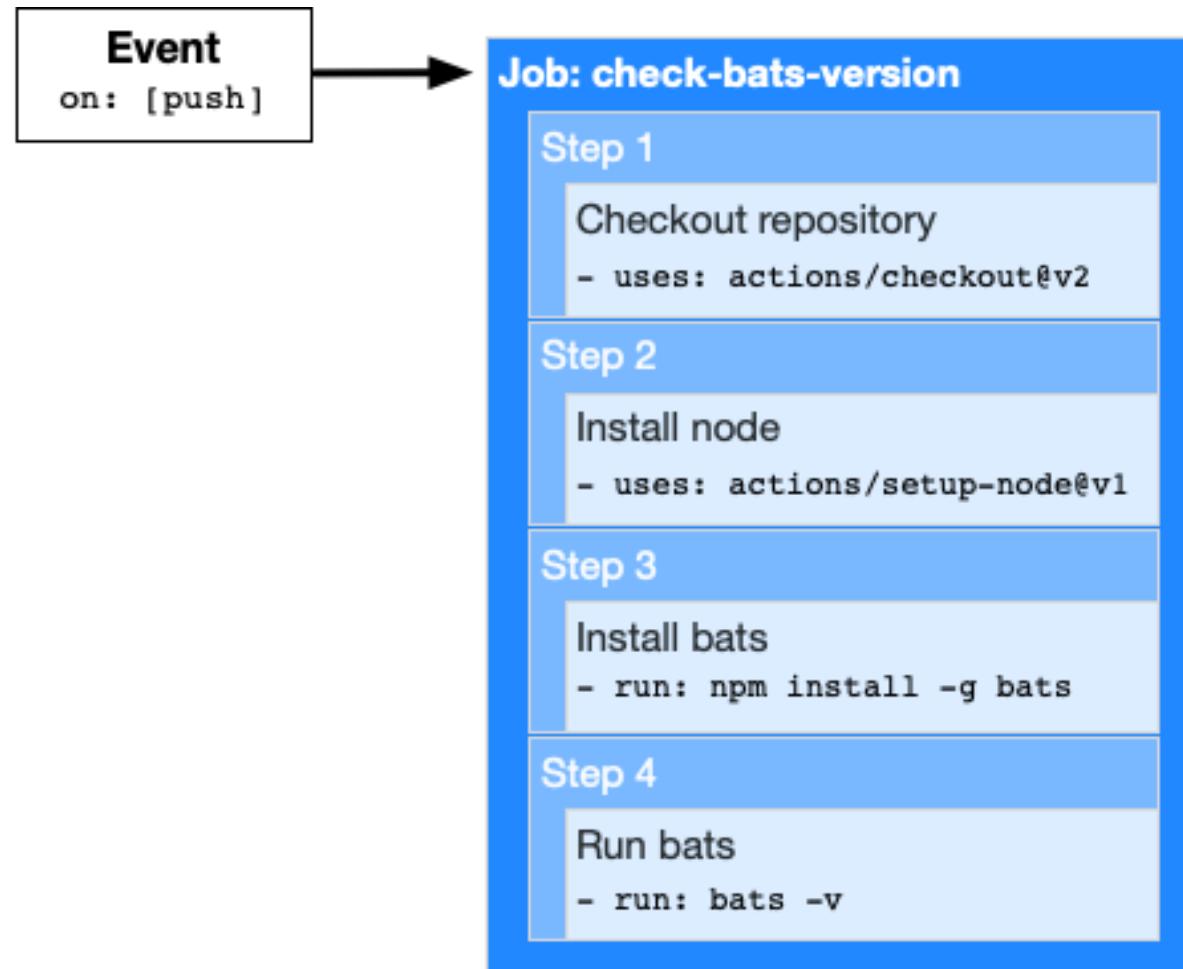
Understanding The Workflow File (1)

<code>name: learn-github-actions</code>	<p><i>Optional</i> - The name of the workflow as it will appear in the Actions tab of the GitHub repository.</p>
<code>on: [push]</code>	<p>Specify the event that automatically triggers the workflow file. This example uses the <code>push</code> event, so that the jobs run every time someone pushes a change to the repository. You can set up the workflow to only run on certain branches, paths, or tags. For syntax examples including or excluding branches, paths, or tags, see "Workflow syntax for GitHub Actions."</p>
<code>jobs:</code>	<p>Groups together all the jobs that run in the <code>learn-github-actions</code> workflow file.</p>
<code>check-bats-version:</code>	<p>Defines the name of the <code>check-bats-version</code> job stored within the <code>jobs</code> section.</p>
<code>runs-on: ubuntu-latest</code>	<p>Configures the job to run on an Ubuntu Linux runner. This means that the job will execute on a fresh virtual machine hosted by GitHub. For syntax examples using other runners, see "Workflow syntax for GitHub Actions."</p>

Understanding The Workflow File (contd)

steps:	Groups together all the steps that run in the <code>check-bats-version</code> job. Each item nested under this section is a separate action or shell command.
<code>- uses: actions/checkout@v2</code>	The <code>uses</code> keyword tells the job to retrieve <code>v2</code> of the community action named <code>actions/checkout@v2</code> . This is an action that checks out your repository and downloads it to the runner, allowing you to run actions against your code (such as testing tools). You must use the checkout action any time your workflow will run against the repository's code or you are using an action defined in the repository.
<code>- uses: actions/setup-node@v1</code>	This action installs the <code>node</code> software package on the runner, giving you access to the <code>npm</code> command.
<code>- run: npm install -g bats</code>	The <code>run</code> keyword tells the job to execute a command on the runner. In this case, you are using <code>npm</code> to install the <code>bats</code> software testing package.
<code>- run: bats -v</code>	Finally, you'll run the <code>bats</code> command with a parameter that outputs the software version.

Visualizing The Workflow File



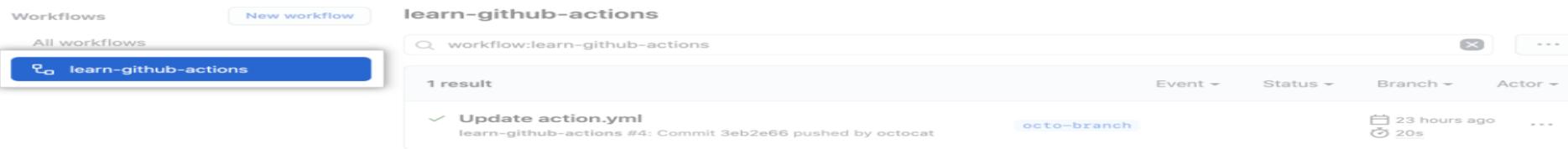
Viewing The Job's Activity

Once your job has started running, you can see a visualization graph of the run's progress and view each step's activity on GitHub.

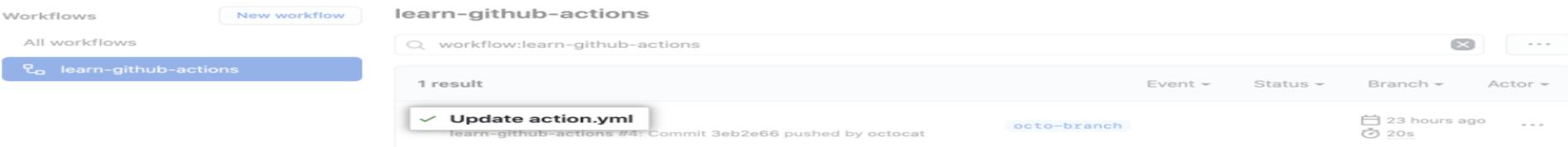
- 1 On GitHub, navigate to the main page of the repository.
- 2 Under your repository name, click Actions.



- 3 In the left sidebar, click the workflow you want to see.

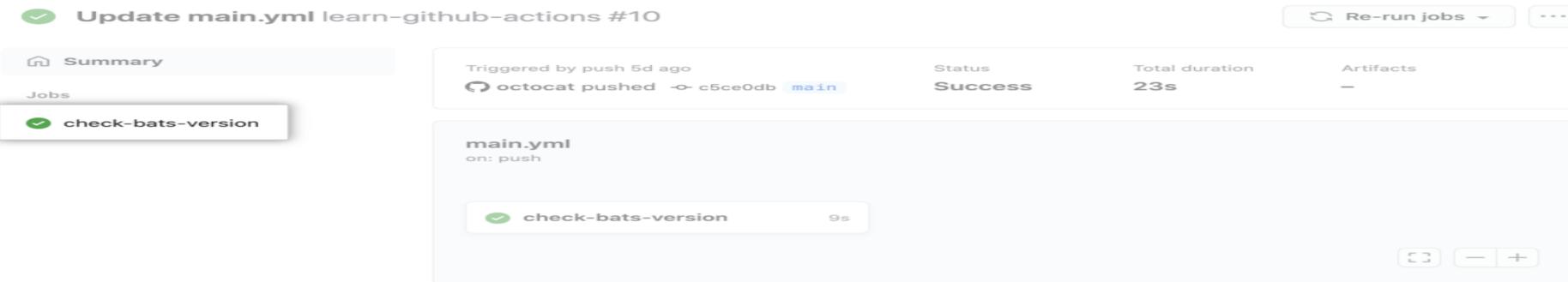


- 4 Under "Workflow runs", click the name of the run you want to see.

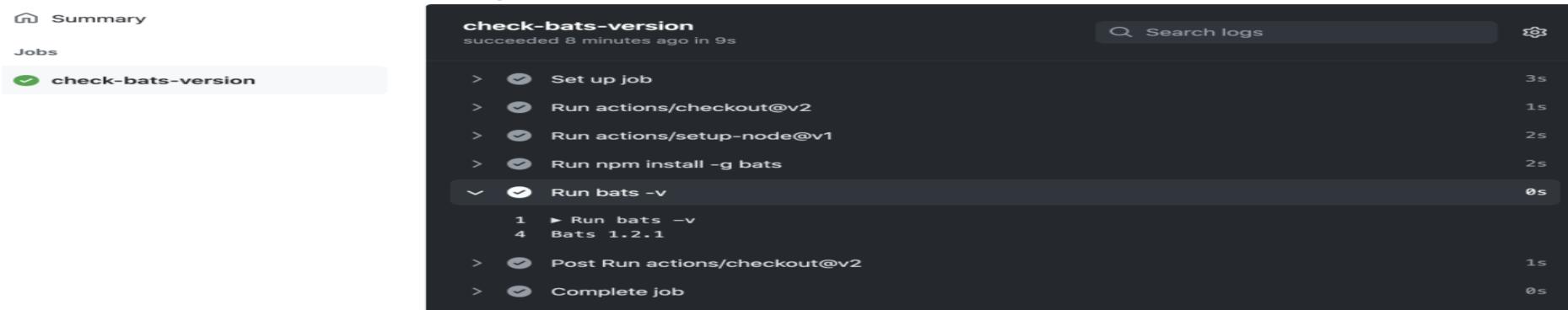


Viewing The Job's Activity (contd.)

5 Under Jobs or in the visualization graph, click the job you want to see.



6 View the results of each step.



Managing Workflow Runs

- You can view the status and results of each step in your workflow, cancel a pending workflow, review deployments, view billable job execution minutes, debug and re-run a failed workflow, search and download logs, and download artifacts.
- <https://docs.github.com/en/actions/managing-workflow-runs>

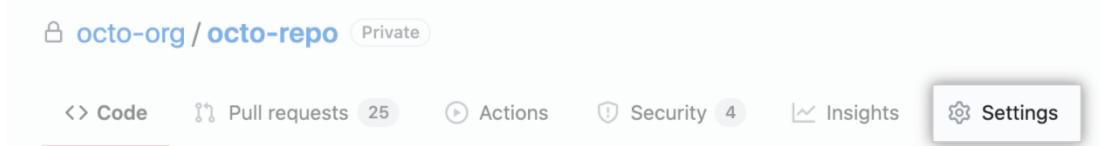
Encrypted Secrets

- Encrypted secrets allow you to store sensitive information in your organization, repository, or repository environments.
- Secrets are encrypted environment variables that you create in an organization, repository, or repository environment.
- The secrets that you create are available to use in GitHub Actions workflows.
- GitHub uses a libsodium sealed box to help ensure that secrets are encrypted before they reach GitHub and remain encrypted until you use them in a workflow.
- <https://docs.github.com/en/actions/reference/encrypted-secrets>

Creating Encrypted Secrets For A Repository

To create secrets for a user account repository, you must be the repository owner. To create secrets for an organization repository, you must have `admin` access.

- 1 On GitHub, navigate to the main page of the repository.
- 2 Under your repository name, click  **Settings**.



- 3 In the left sidebar, click **Secrets**.
- 4 Click **New repository secret**.
- 5 Type a name for your secret in the **Name** input box.
- 6 Enter the value for your secret.
- 7 Click **Add secret**.

Using Encrypted Secrets In A Workflow

- To provide an action with a secret as an input or environment variable, you can use the `secrets` context to access secrets you've created in your repository.
- Avoid passing secrets between processes from the command line, whenever possible.
- Command-line processes may be visible to other users (using the `ps` command) or captured by security audit events.
- To help protect secrets, consider using environment variables, STDIN, or other mechanisms supported by the target process.
- If you must pass secrets within a command line, then enclose them within the proper quoting rules.
- Secrets often contain special characters that may unintentionally affect your shell.
- To escape these special characters, use quoting with your environment variables.
- **Secrets are limited to 64 KB in size.**

```
steps:  
  - name: Hello world action  
    with: # Set the secret as an input  
          super_secret: ${{ secrets.SuperSecret }}  
    env: # Or as an environment variable  
         super_secret: ${{ secrets.SuperSecret }}
```

Example using Bash

```
steps:  
  - shell: bash  
    env:  
      SUPER_SECRET: ${{ secrets.SuperSecret }}  
    run: |  
        example-command "$SUPER_SECRET"
```

GitHub Actions Marketplace

- <https://github.com/marketplace?type=actions>

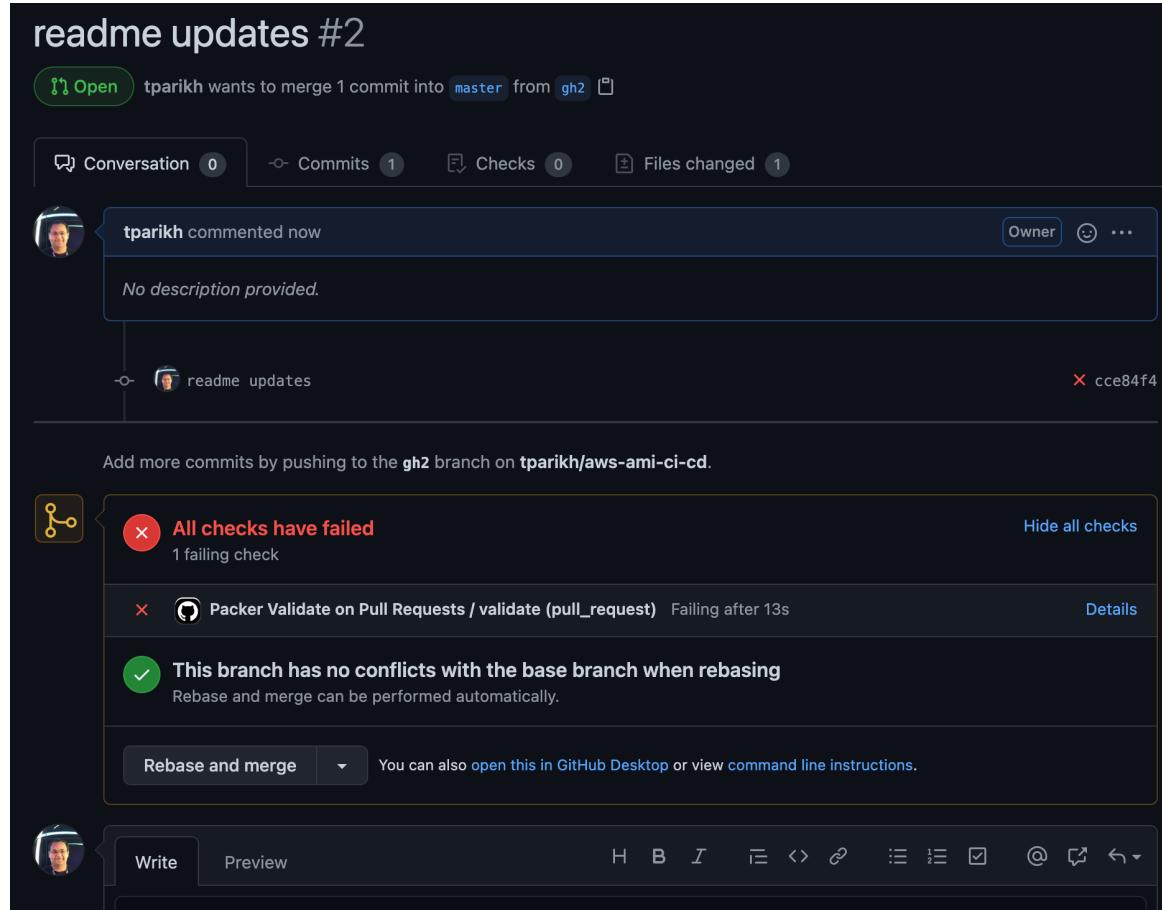
GitHub Action for AMI Pull Request

```
.github > workflows > pull-requests.yml > {} jobs > {} validate > [ ] steps
You, seconds ago | 1 author (You)
1 ---
2   name: Packer Validate on Pull Requests
3
4   # Controls when the action will run.
5   # Triggers the workflow on push or pull request events but only for the main branch
6   on:
7     # Workflow is triggered on pull requests
8     pull_request:
9       # Workflow will only be triggered for pull requests on "master" branch
10      branches: [ master ]
11
12 # A workflow run is made up of one or more jobs that can run sequentially or in parallel
13 jobs:
14   # This workflow contains a single job called "build"
15   validate:
16     # The type of runner that the job will run on
17     runs-on: ubuntu-latest
18
19     # Steps represent a sequence of tasks that will be executed as part of the job
20     steps:
21       # Checks out your repository under $GITHUB_WORKSPACE, so your job can access it
22       - name: Checkout Repository
23         uses: actions/checkout@v2
24
25       # Ref: https://github.com/marketplace/actions/packer-github-actions
26       - name: Validate Packer Template
27         uses: hashicorp/packer-github-actions@master
28         with:
29           command: validate
30           arguments: --syntax-only # only validate syntax
31           target: ami.json
32           env:
33             PACKER_LOG: 1 # enable debug log for packer
34
```

Repo:

<https://github.com/tparikh/aws-ami-ci-cd>

GitHub Status Checks

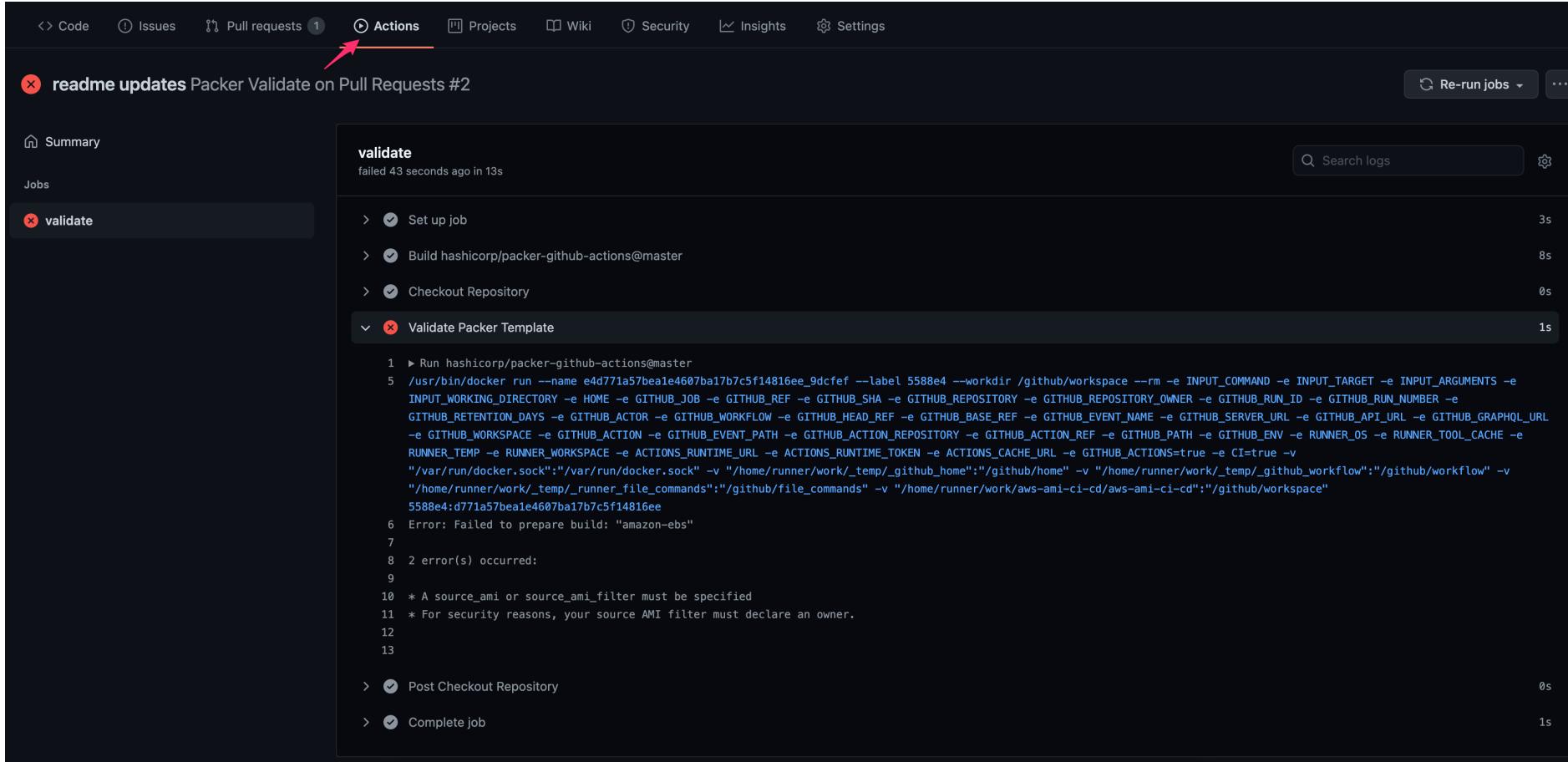


Debugging Failed Workflows

The screenshot shows the GitHub Actions interface. The top navigation bar includes links for Code, Issues, Pull requests (1), Actions (highlighted with a red arrow), Projects, Wiki, Security, Insights, and Settings. Below the navigation is a header with 'Workflows' and 'New workflow' buttons. A blue button labeled 'All workflows' is also highlighted with a red arrow. The main section is titled 'All workflows' and displays 'Showing runs from all workflows'. It includes a search bar for 'Filter workflows'. Below this, it shows '2 workflow runs' with the following details:

Workflow	Last Run	Event	Status	Branch	Actor
Packer Validate on Pull Requests	2 minutes ago 26s	gh2	✗ readme updates	Pull request #2 opened by tparikh	...
Packer Validate on Pull Requests	3 minutes ago 28s	gh1	✗ setup github actions	Pull request #1 opened by tparikh	...

Workflow Run Logs



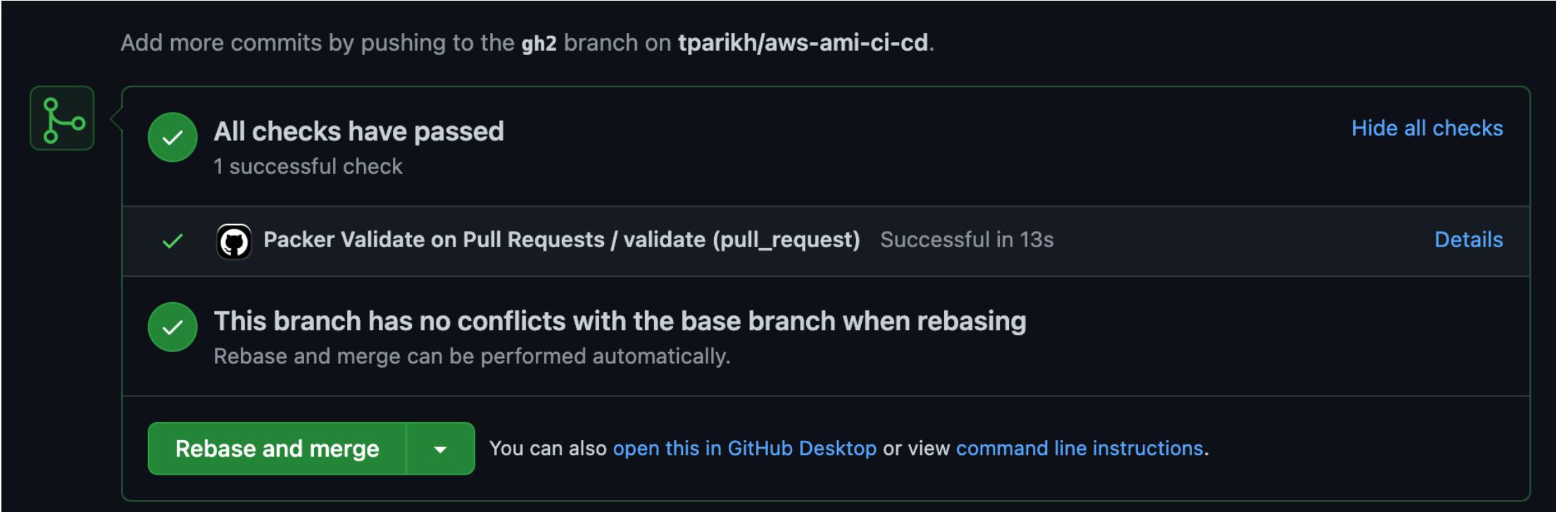
The screenshot shows the GitHub Actions interface for a workflow named "readme updates Packer Validate on Pull Requests #2". The "Actions" tab is selected, indicated by an orange arrow pointing to it. The workflow has one job named "validate" which failed 43 seconds ago in 13s. The log output for the "Validate Packer Template" step shows an error message:

```
1 ► Run hashicorp/packer-github-actions@master
5 /usr/bin/docker run --name e4d771a57bea1e4607ba17b7c5f14816ee_9dcfef --label 5588e4 --workdir /github/workspace --rm -e INPUT_COMMAND -e INPUT_TARGET -e INPUT_ARGUMENTS -e INPUT_WORKING_DIRECTORY -e HOME -e GITHUB_JOB -e GITHUB_REF -e GITHUB_SHA -e GITHUB_REPOSITORY -e GITHUB_REPOSITORY_OWNER -e GITHUB_RUN_ID -e GITHUB_RUN_NUMBER -e GITHUB_RETENTION_DAYS -e GITHUB_ACTOR -e GITHUB_WORKFLOW -e GITHUB_HEAD_REF -e GITHUB_EVENT_NAME -e GITHUB_SERVER_URL -e GITHUB_API_URL -e GITHUB_GRAPHQL_URL -e GITHUB_WORKSPACE -e GITHUB_ACTION -e GITHUB_EVENT_PATH -e GITHUB_ACTION_REPOSITORY -e GITHUB_ACTION_REF -e GITHUB_PATH -e GITHUB_ENV -e RUNNER_OS -e RUNNER_TOOL_CACHE -e RUNNER_TEMP -e RUNNER_WORKSPACE -e ACTIONS_RUNTIME_URL -e ACTIONS_RUNTIME_TOKEN -e ACTIONS_CACHE_URL -e GITHUB_ACTIONS=true -e CI=true -v
"/var/run/docker.sock":"/var/run/docker.sock" -v "/home/runner/work/_temp/_github_home":"/github/home" -v "/home/runner/work/_temp/_github_workflow":"/github/workflow" -v
"/home/runner/work/_temp/_runner_file_commands":"/github/file_commands" -v "/home/runner/work/aws-ami-ci-cd/aws-ami-ci-cd":"/github/workspace"
5588e4:d771a57bea1e4607ba17b7c5f14816ee
6 Error: Failed to prepare build: "amazon-ebs"
7
8 2 error(s) occurred:
9
10 * A source_ami or source_ami_filter must be specified
11 * For security reasons, your source AMI filter must declare an owner.
12
13
```

Following the error, the workflow continues with "Post Checkout Repository" and "Complete job" steps.

Successful Status Checks

Add more commits by pushing to the **gh2** branch on **tparikh/aws-ami-ci-cd**.



All checks have passed
1 successful check

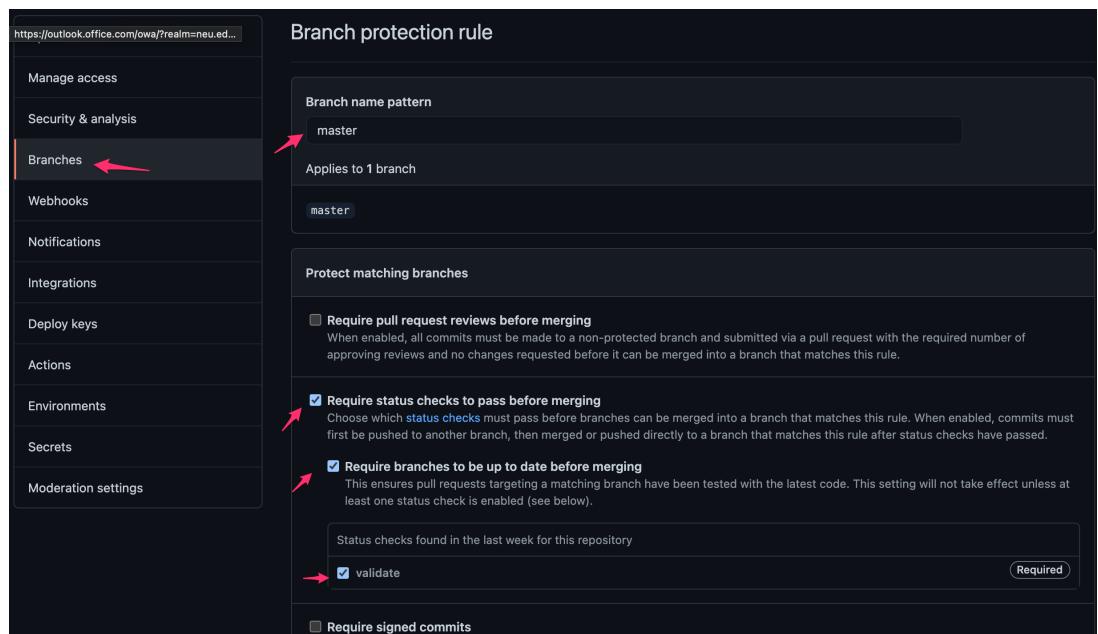
Packer Validate on Pull Requests / validate (pull_request) Successful in 13s [Details](#)

This branch has no conflicts with the base branch when rebasing
Rebase and merge can be performed automatically.

Rebase and merge ▾ You can also open this in GitHub Desktop or view command line instructions.

Enforcing Status Checks

Note that you will need GitHub Action workflow already committed to the repo to enable status checks.



Codebase:

Dependencies: Take a backup of the dependencies (for example the avj library or joi library)

Make sure we look at the packages before we add them to our project.

How can we protect our project incase if any dependences are deleted by their authors? This will break our app next time we run npm i

To solve this we can do mirroring of the packages, like your org can have its own artifactory that has a mirror of packages, this way we are always secure upto one level.

Configurations: Outside of the codebase like: Database env variables, port, hostname -> anything dynamic

Any things that is depended on environment, we have to start using config to inject it into our application

Backing Services: Treat backing services as attached resources

Processes: Execute the app as one or more stateless processes

Additional Resources

Port Binding: Export services via port binding

NOTE: The TCP/IP port numbers below 1024 are special in that normal users are not allowed to run servers on them. This is a security feature, in that if you connect to a service on one of these ports you can be fairly sure that you have the real thing, and not a fake which some hacker has put up for you.

http port is 443, privilege port is anything less than 1024, anything after that is not privileged and that's why many apps using ports like 3005, 8080 by default

Create a light-weight application (Maybe, Make sure that you are not dependent on too many libraries)

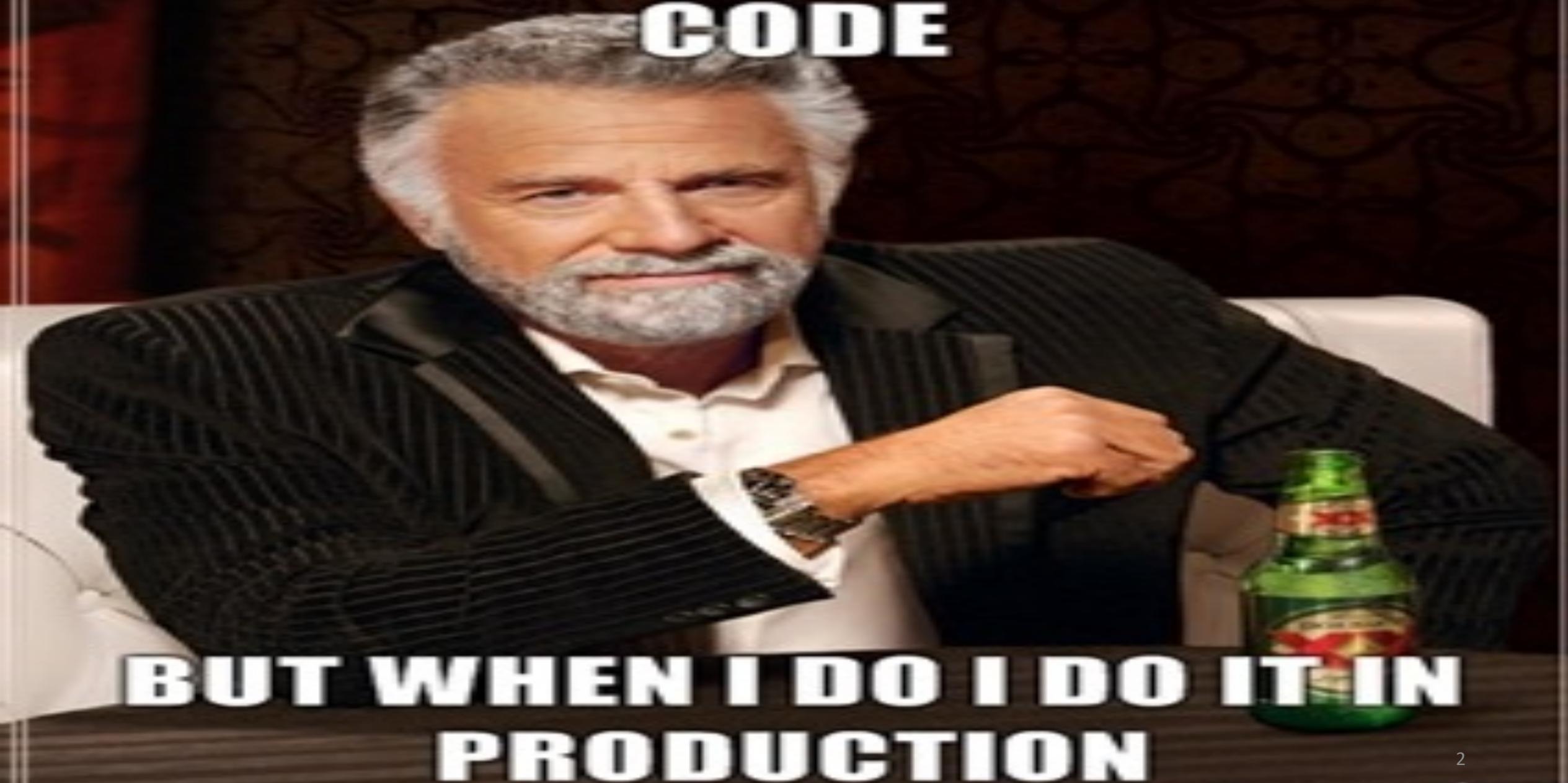
See Lecture Page

Unit, Integration, & Functional Testing

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University

I DON'T ALWAYS TEST MY
CODE



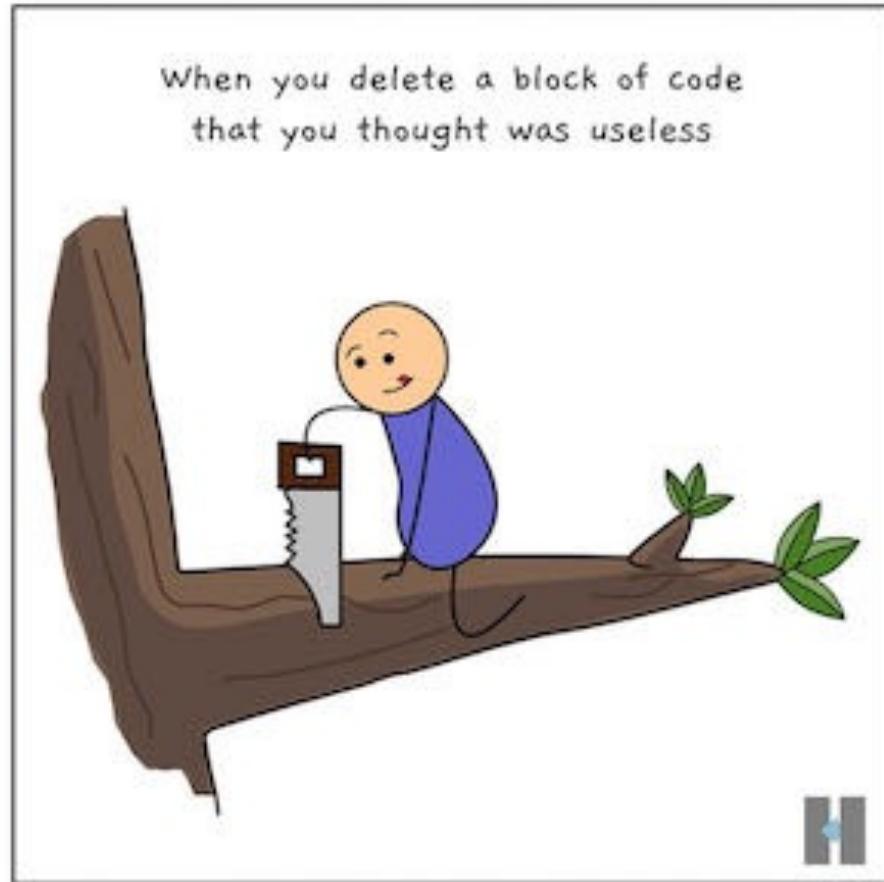
BUT WHEN I DO I DO IT IN
PRODUCTION

Fixing a bug in production.



Why Test?

- Testing reduces bugs
- Tests serve as good documentation
- Tests allow for safe refactoring
- Tests reduce the cost of making code changes
- Tests allow you to deliver code with confidence



f /techindustan

t /techindustan

g+ /techindustan



***whenever a programmer
comes up with a new bug**



HOUSTON

**FEATURE
WE HAVE A PROBLEM...**

When the bug can be called a feature



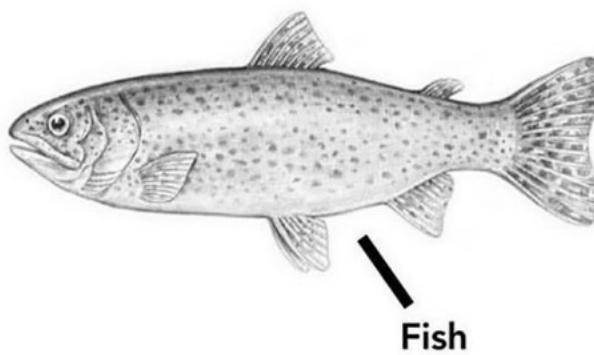
After all... Why not?



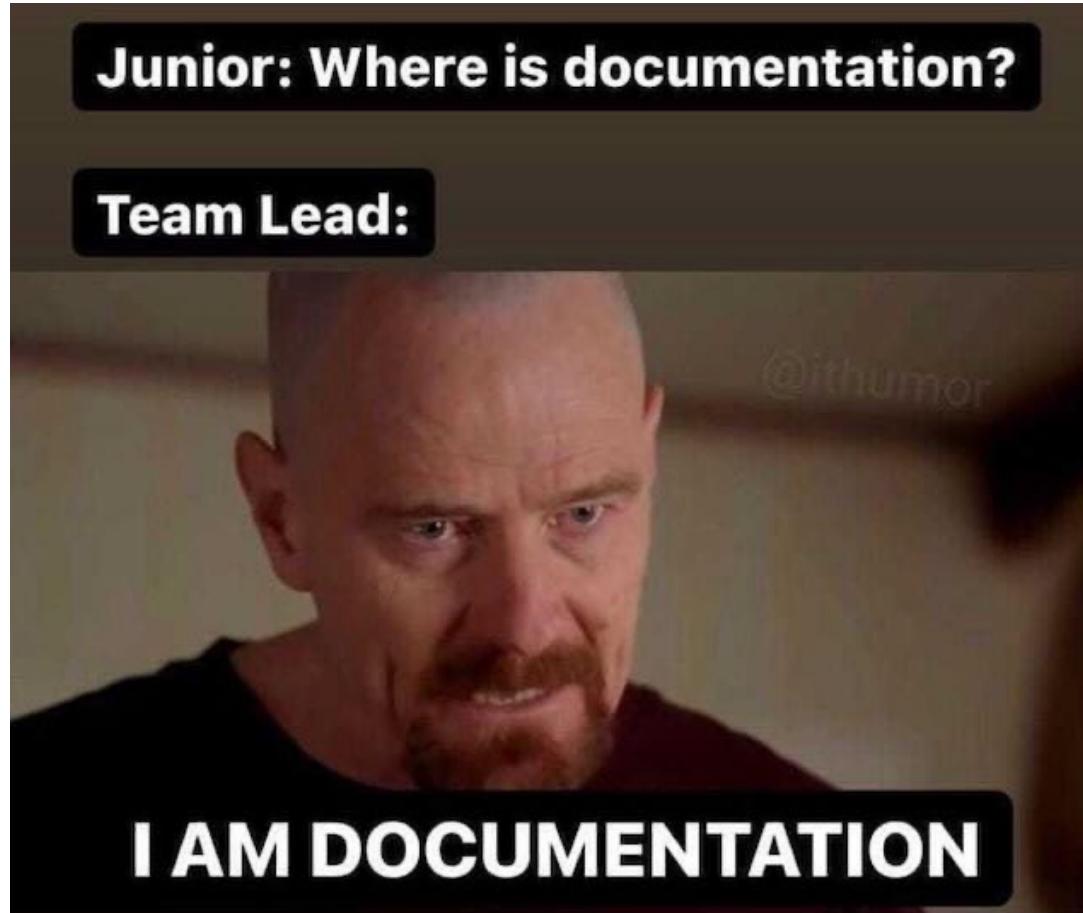
Why shouldn't I keep it?

Bad Code Comments

Fish Diagram



None/Poor Documentation

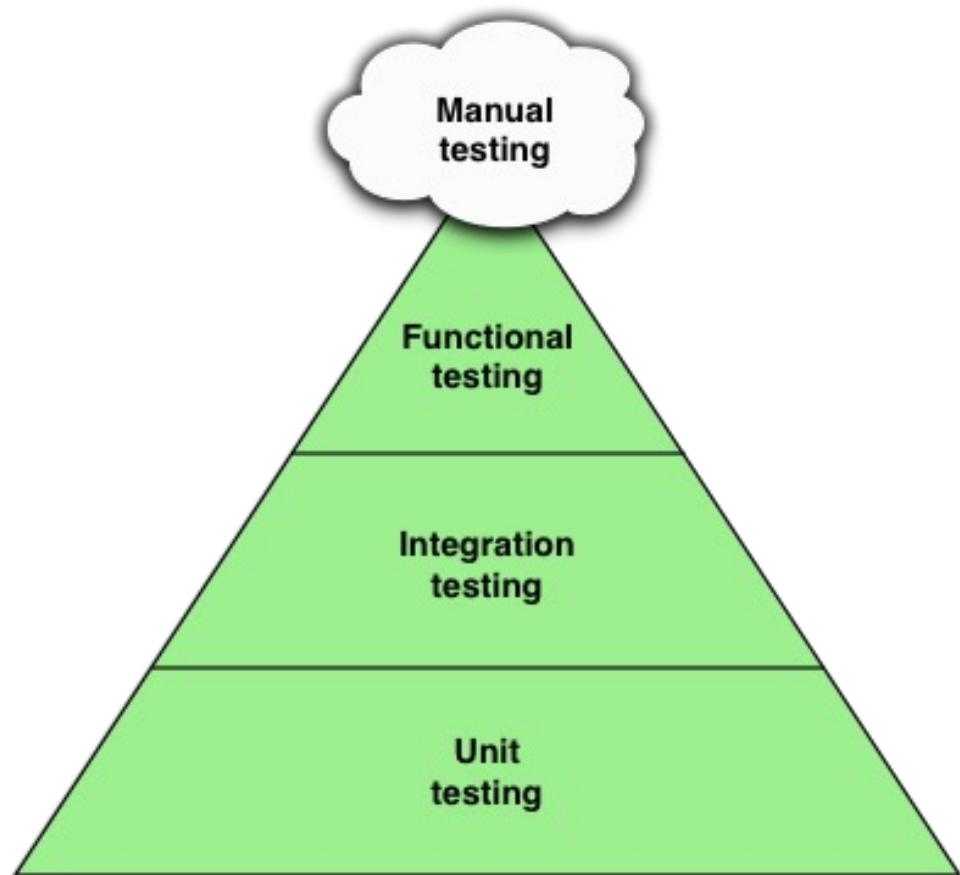


The entire code after I change
a single letter in a comment



Types of Tests

- There are way too many testing levels and types to list them here.
- We will focus on Unit and Integration tests.



What is Unit Test?

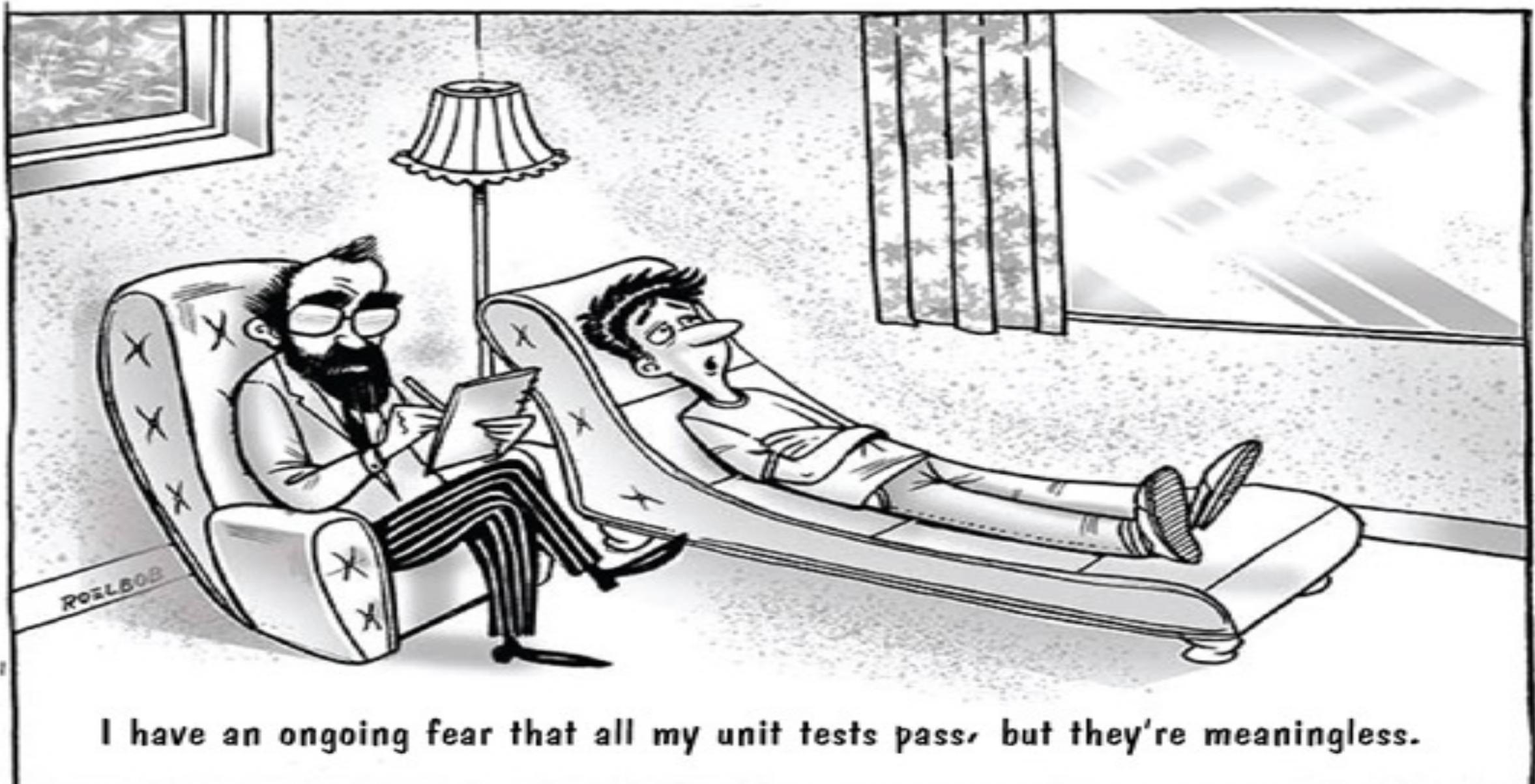
- Unit tests should not require access to any external systems such as network, databases, etc.
- All external systems such as database, file server, etc. are mocked out using specific test APIs and test data.

Unit Tests

- Isolate parts of programs
- Verify that independent part of programs are working correctly
- Unit tests are fast & reliable
- However, Unit tests
 - Take time to build
 - Require maintenance
- Both of these points require significant time and commitment. An incorrect unit test can let bug go thru unnoticed for long time.

**Sometimes my code
is like this.....**





I have an ongoing fear that all my unit tests pass, but they're meaningless.

What is Integration Test?

- Integration tests verify that interaction between multiple components (applications, services, modules, etc.) is working as expected.

Unit test vs. Integration test



Integration Test Challenges

- Difficult to test all critical paths
- Hard to find the source of errors
- Requires time and commitment from multiple component owners

Performance/Load/Stress Testing

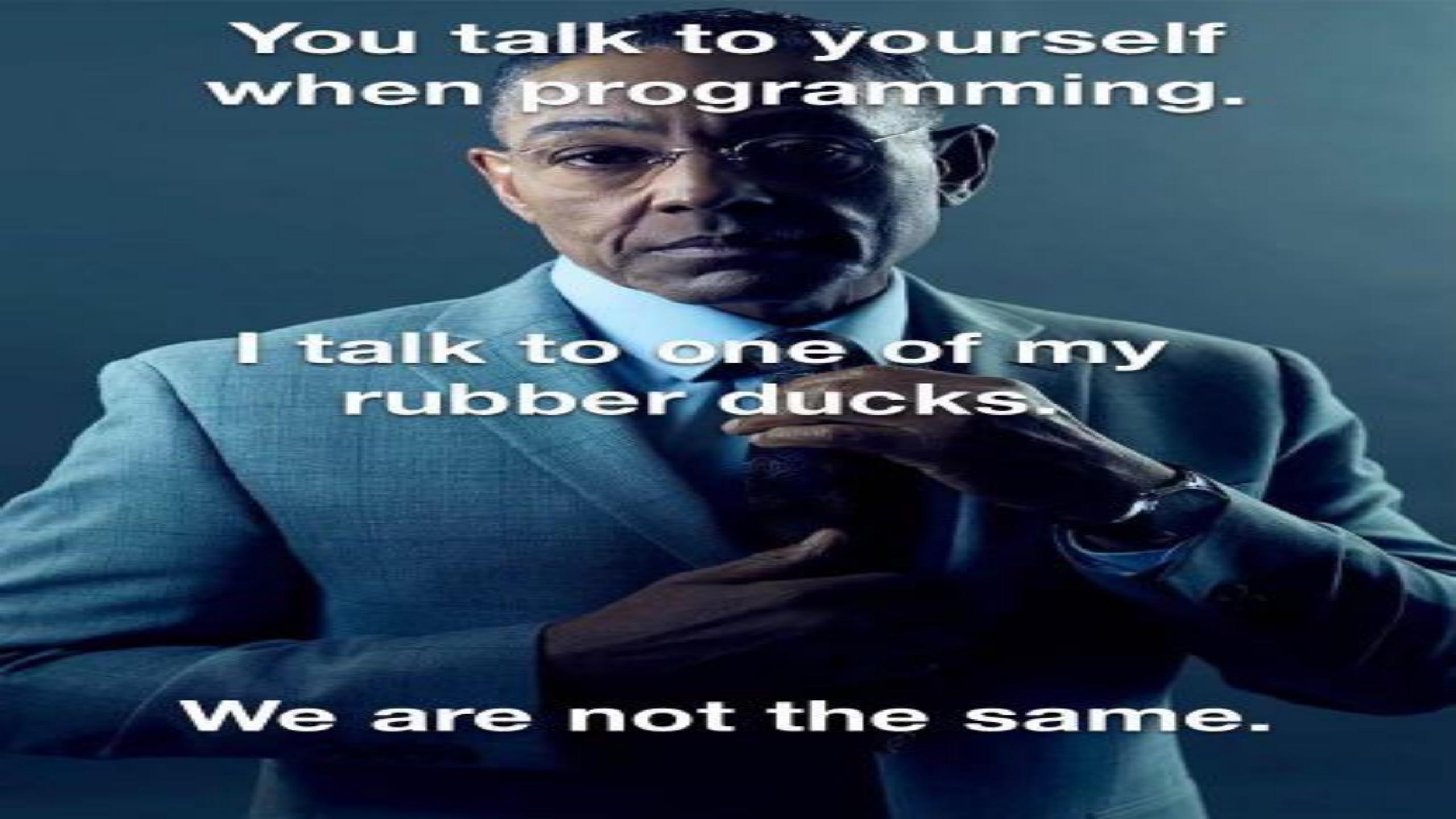
- Simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types.
- Load testing is also a way to perform a functional test on websites, databases, LDAPs, webservices etc.



Cher

@cherthedev

Am I testing my code
or is it testing me
10:09 PM · 20 Oct 20 · Twitter Web App

A black and white photograph of a man with glasses and a suit, holding a rubber duck. He is looking directly at the camera with a serious expression.

You talk to yourself
when programming.

I talk to one of my
rubber ducks.

We are not the same.

Additional Resources

See Lecture Page

Fundamentals of Computer Networking

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225

Northeastern University

What is Network?

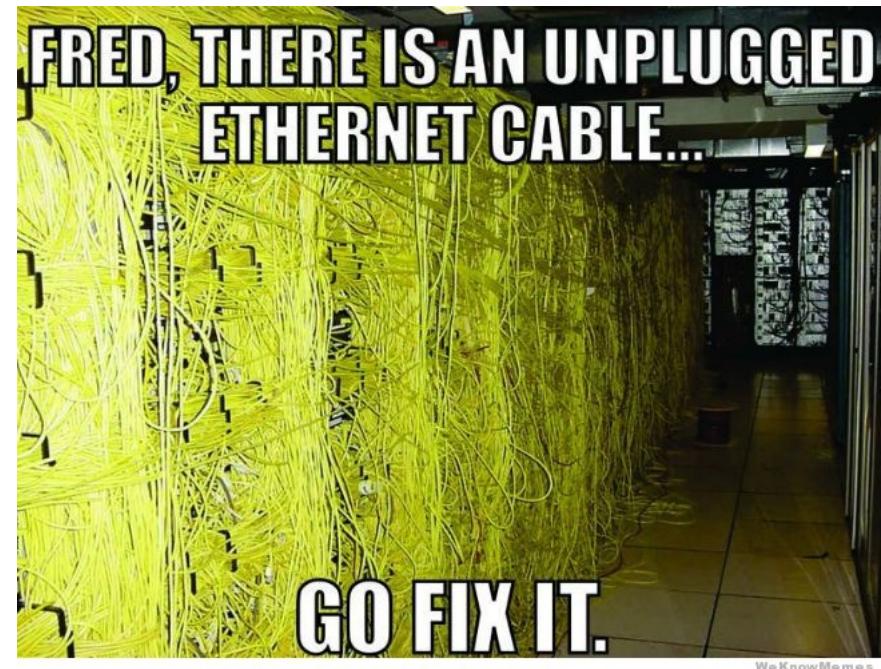
When defining network / vpc: (In the real world)
Always have custom network definitions, not default

Networking: config always mentions in bits per second

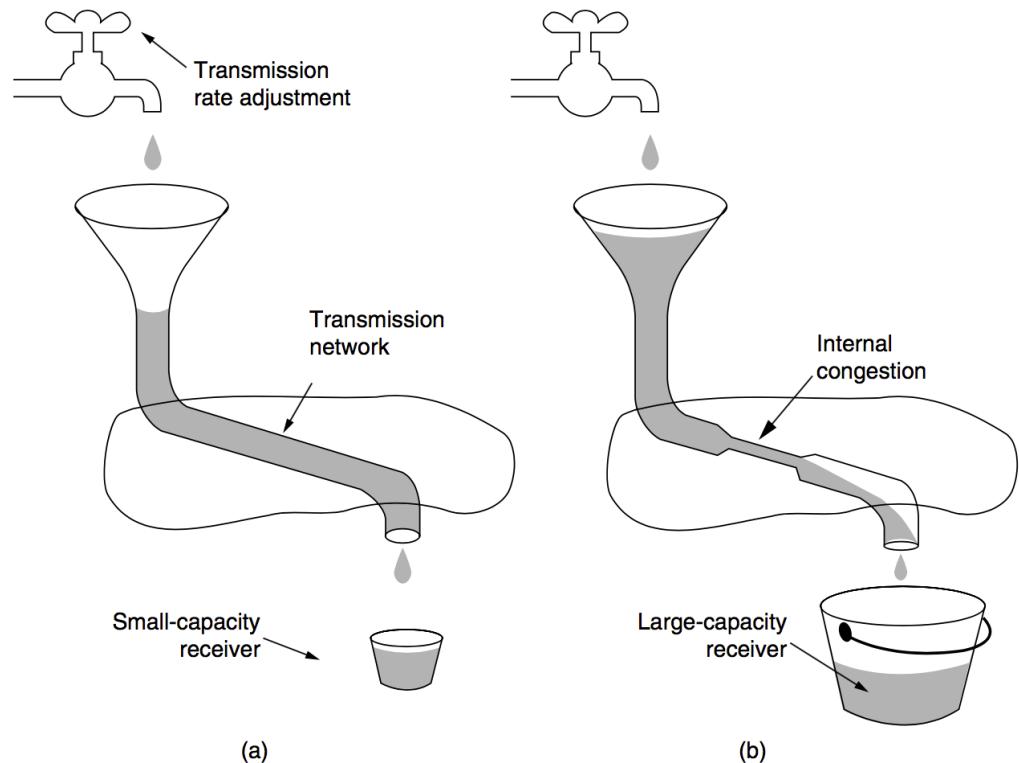
- A network is simply bunch of systems connected to each other via some kind of communication link.
- The common type of networks are:
 - LAN: Local Area Network
 - WAN: Wide Area Network
- The main difference between LAN and WAN is that LAN is usually confined to local geographic area whereas WAN is spread across a larger geographic area. WAN is usually made of multiple LANs.
- Systems are not connected directly to each other. Instead they are indirectly connected to each other via a switch or router.

Communication Link

- Systems are connected to each other via communication link such as coaxial, copper or fiber cable or radio waves.
- Different communication links can transmit data at different rates.
- Data transmission rate of a link is measured in bits/second.
- Note that we use bits/second and not bytes per second. 1 byte is 8 bits. 1byte is 8 bits. If B is capital then its bytes, if its lower case b its bits



Bandwidth



In computing, **bandwidth** is the maximum rate of data transfer across a given path.

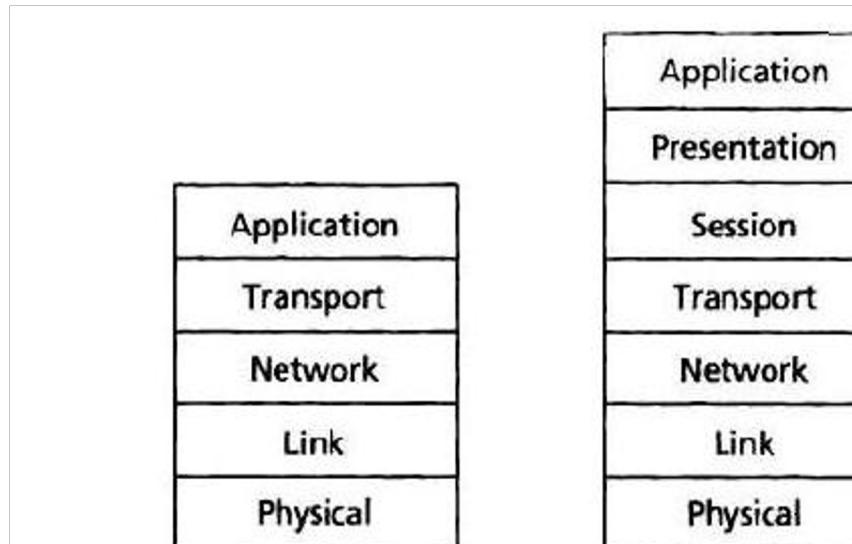
Network Protocols

Protocols define the format and order of messages exchanged between two or more communicating entities as well as the actions taken on transmission and/or receipt of a message or other event.

Network Protocol Stack & OSI Reference Model

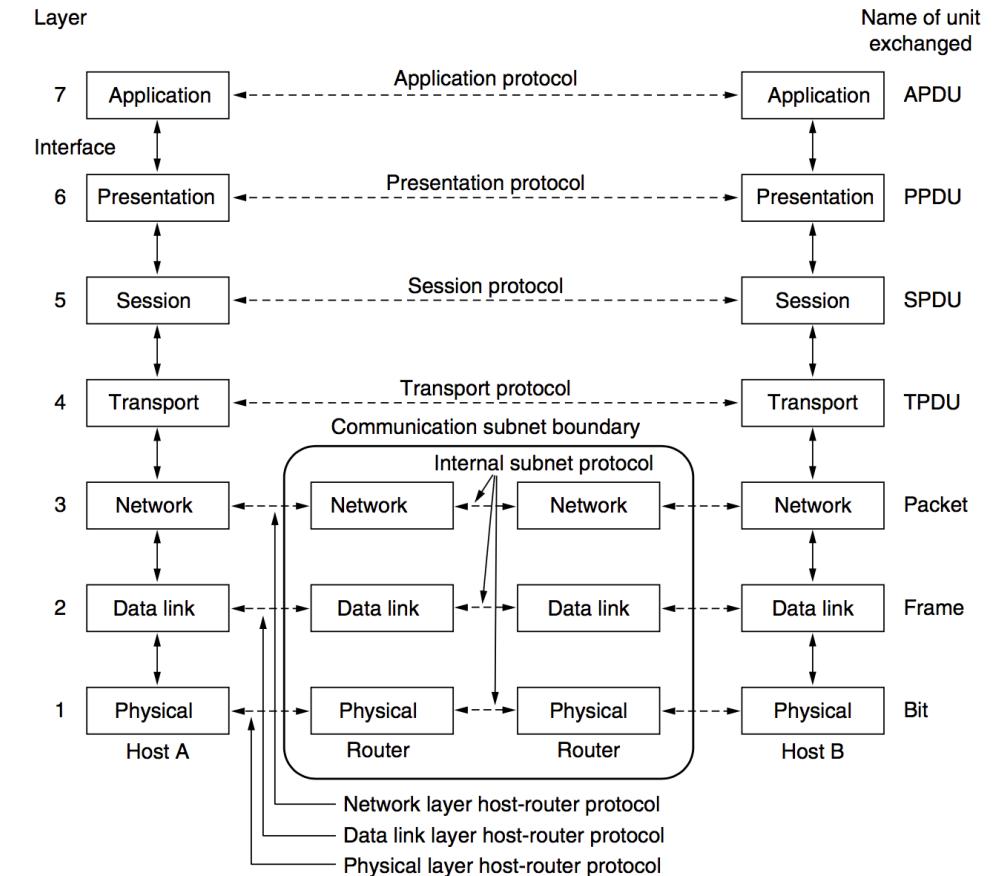
Learn more about it, every exam has OSI

Web services -> let you control till layer 4 max layer 3, not after Link



**b. Seven-layer
ISO OSI
reference model**

The Internet protocol stack (a) and OSI reference model (b)



Application Layer

This is the layer where network applications and application-layer protocols such as HTTP, FTP, etc. reside.

[SMTP](#)

Presentation & Session Layer

Socket direct protocol, session control protocol

- Session layer handles data related to particular user session.
- Presentation layer handles rendering of data. [MIDI](#), [MPEG](#) -> data formats
- Example of presentation layer protocols are audio and video codes, different image formats such as PNG, JPEG, etc.

[Presentation \(audio & video players\) & Session \(social media -> user is automatically logged in, session is stored i.e. session expires very later in time, but BFSI -> security & trust -> session expires very early in time\)](#) layer is not always present

Transport Layer

- The transport layer transports application layer messages between client and server side of the application.
- Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are examples of some popular transport layer protocols.

Network Layer

- The network layer is responsible for moving transport layer protocols from one host to another.
- The network layer defines the fields in data packet (datagram) as well as how the end systems and routers act on this field.
- If the data packet is too large, network layer might split it into multiple packets which will be combined into one at the receiving node.
- Most popular network layer protocol is the Internet Protocol (IP).

Link Layer

Link layer is responsible for moving packet from one node to the next node in the route.

Physical Layer

While the job of link layer is to move packets of data from one node to another, physical layer's job is to move the individual bits.

Transmission Control Protocol (TCP)

- Reliable Data Transfer - TCP ensures that data is delivered from sending process to receiving process correctly and in order.
- TCP is connection based - Sender will keep connection with receiver open until it is done successfully sending all the data packets.
- TCP does error checking and erroneous packets will be retransmitted from source to destination.
- TCP has congestion control. It will not send next chunk of data until it has received OK (ACK) from the receiver.
- Common application-layer protocol that use TCP/IP are HTTP, FTP, SMTP, etc.

User Datagram Protocol (UDP)

- UDP protocol is designed to do as little work as possible to send data.
- UDP has no congestion control. Data is transmitted as soon as it can.
- UDP has no reliability guarantee. It does not care if the message was successfully received.
- UDP has no overhead of connection establishment.
- No connection state is maintained as we are not tracking successful delivery of data or the order in which data is sent.
- Common application-layer protocols that use UDP are Domain Name Systems (DNS), Dynamic Host Configuration Protocol (DHCP)

TCP



UDP

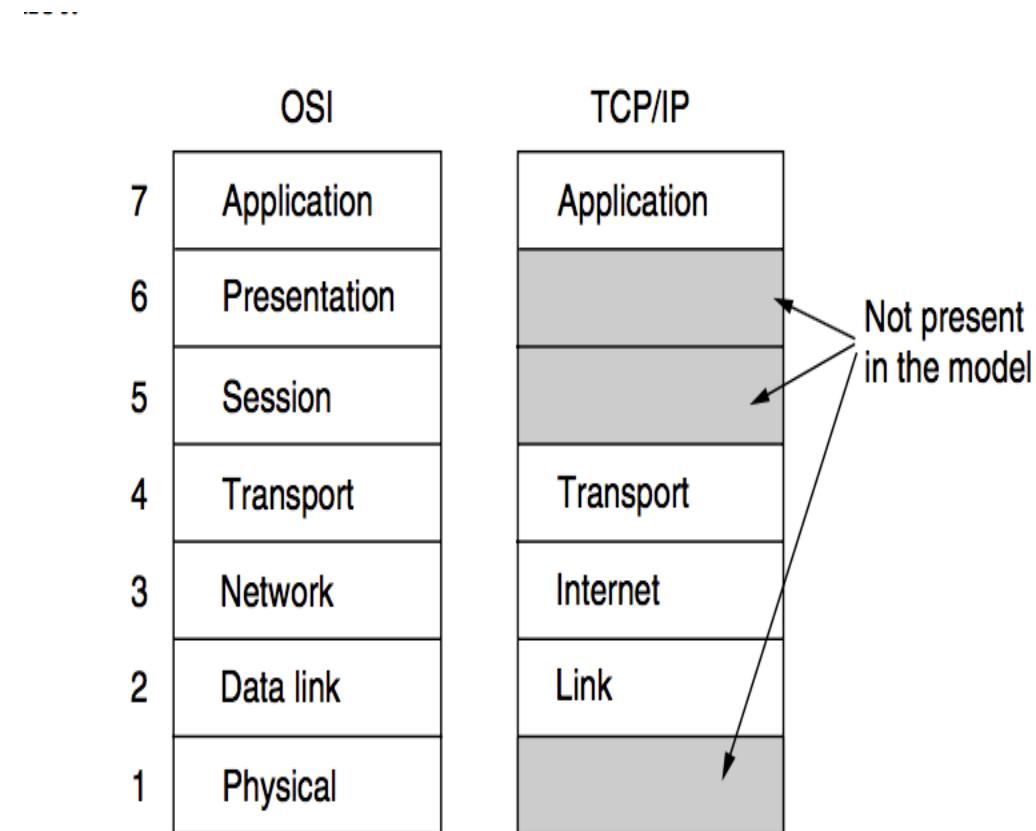


Internet Protocol (IP)

Layer 3 Protocol

- The Internet Protocol (IP) is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries.
- IP is best-effort delivery service, makes no guarantee about delivering correct data, the order in which it is delivered or avoidance of delivering duplicate data pretty much like UDP.
- First major version of IP is Internet Protocol Version 4 (IPv4) and its successor is Internet Protocol Version 6 (IPv6).

The TCP/IP reference model



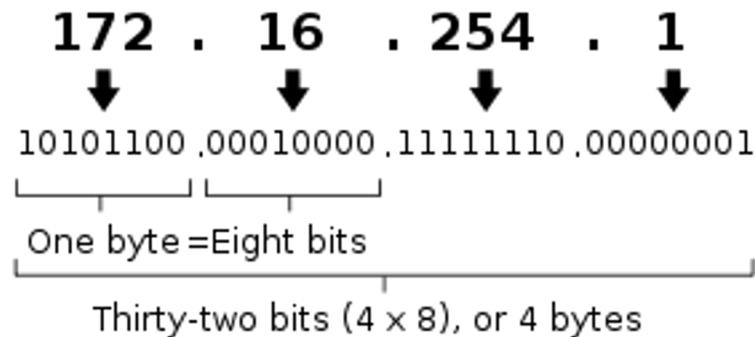
IPv4

- IPv4 uses 32 bit addressing system.
- IPv4 address space is 2^{32} which is approximately 4 billion IP addresses.
- Each system on network must have globally unique* IP address.

IPv4 Address Representation

IPv4 address is usually written in its decimal form and is separated by period (dot) from other bytes.

An IPv4 address (dotted-decimal notation)



IPv4 Address Exhaustion

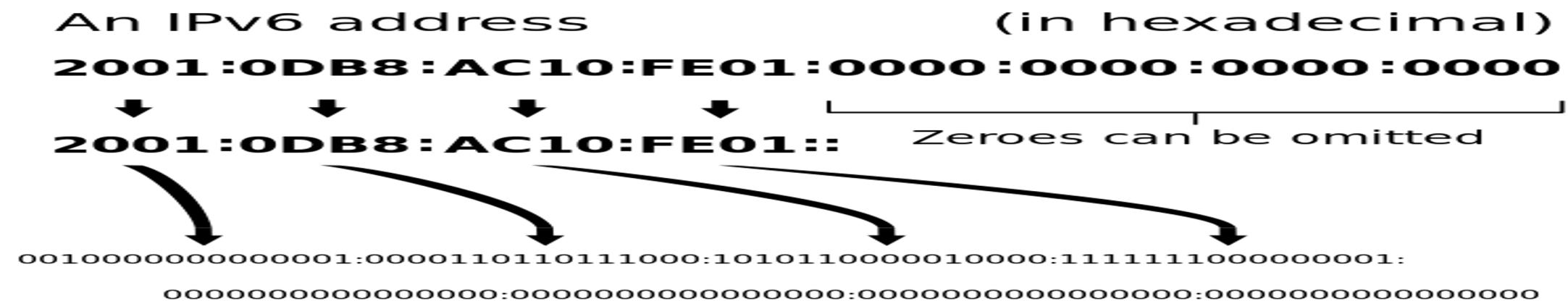
- IPv4 addresses have been exhausted since 2011.
- While IPv4 has approximately 4 billion IP addresses, not all of them are available for use. About ~18 million addresses are allocated for private network and ~270 million addresses for multicast addresses.

Internet Protocol version 6 (IPv6)

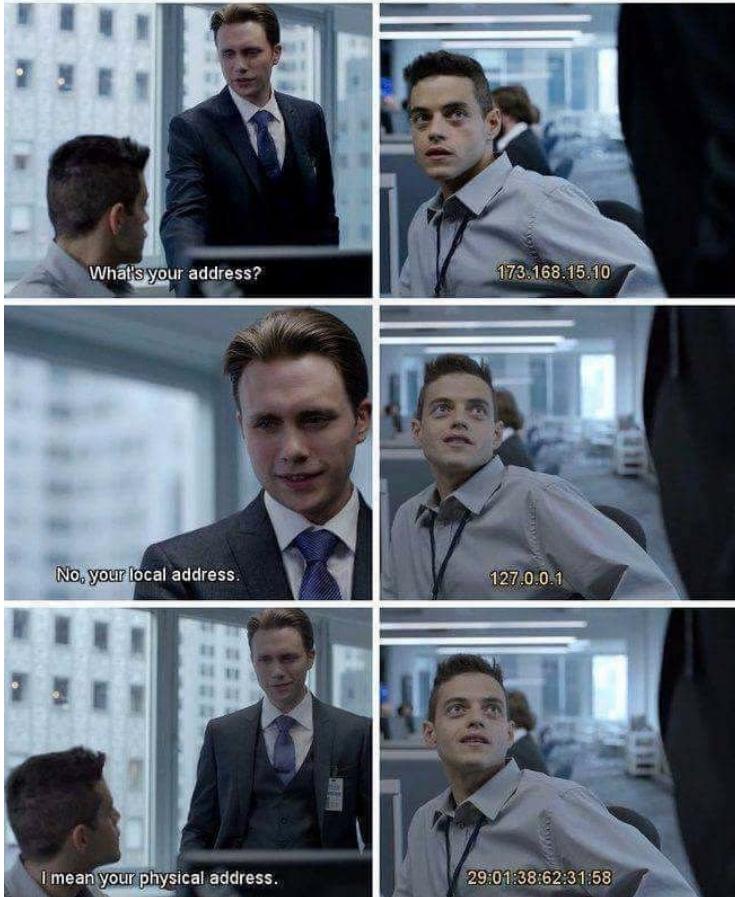
- Most recent version of Internet Protocol.
- IPv6 uses 128 bit addressing system theoretically allowing 2^{128} addresses.
- IPv6 does not support NAT.

IPv6 Address Representation

IPv6 addresses are represented as eight groups of four hexadecimal digits with the groups being separated by colons (:), for example
2001:0db8:0000:0042:0000:8a2e:0370:7334

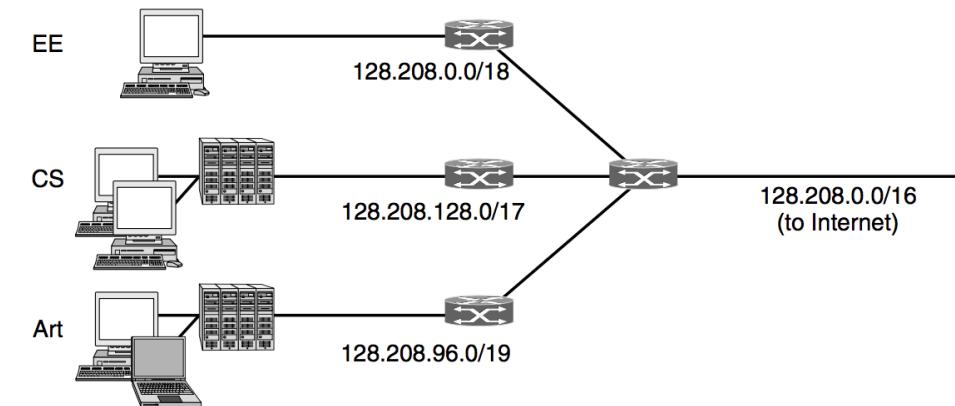


What is your address?



Subnet

- A subnet is isolated network.
- IP addresses are assigned to subnet with a subnet mask.
- Subnets are usually represented using CIDR notation that is written as the first address of a network, followed by a slash character (/), and ending with the bit-length of the prefix.
- For example, 192.168.1.0/24 is the prefix of the IPv4 network starting at the given address, having 24 bits allocated for the network prefix, and the remaining 8 bits reserved for host addressing.

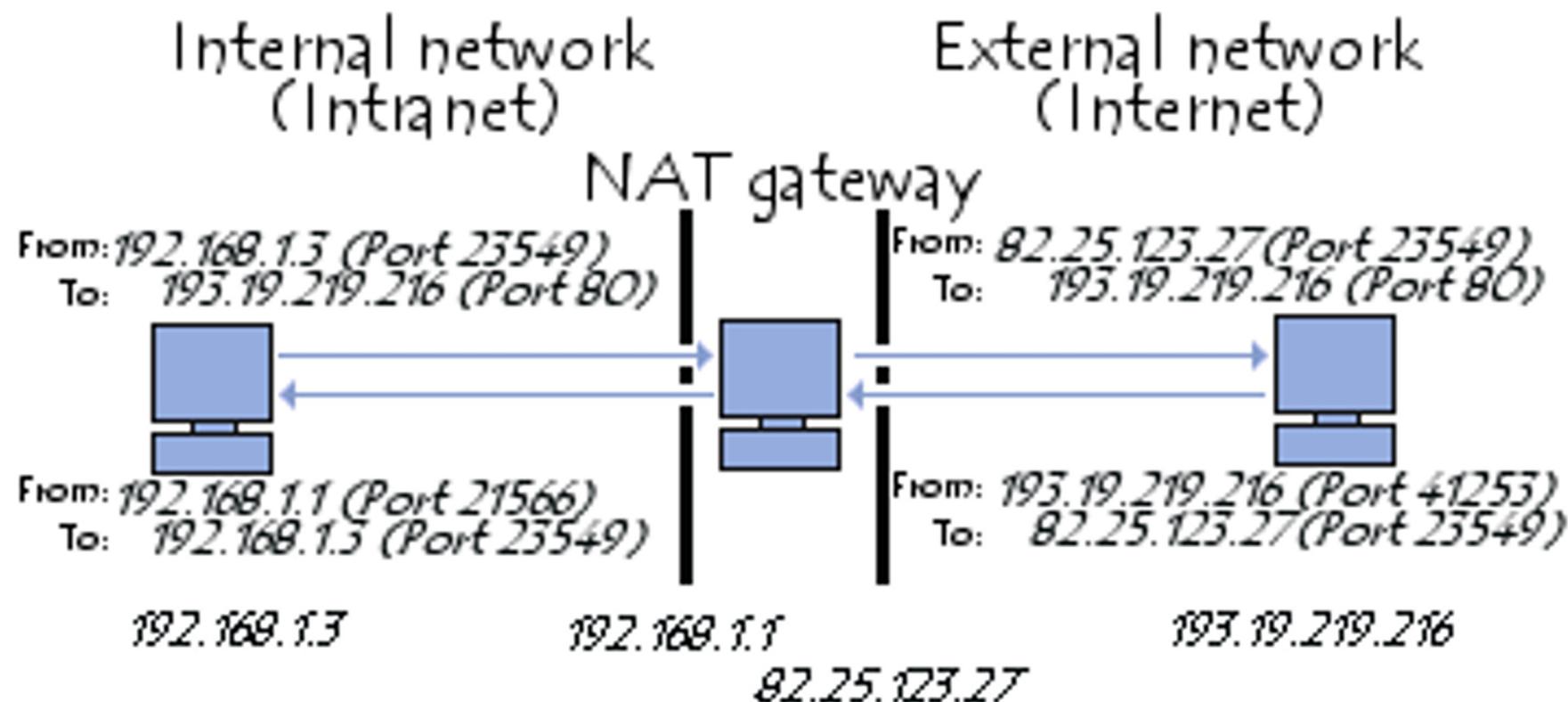


Cidr - Classless Inter-Domain Routing Single IP Range -> /32

Network Address Translation (NAT)

- NAT allows sharing of one Internet-routable IP address of a NAT gateway for an entire private network.
- NAT allows ISPs to support more systems with the limited availability of public IPs.
- Internet of Things (IoT) has significantly increased number of systems connected to internet.
- IPv6 adoption has been slow and older devices, systems and applications don't support it.
- NAT does not work very well with Peer-to-peer (P2P) routing as P2P usually requires 2 hosts to be able to communicate with each other directly.

NAT Table Example



Virtual Private Cloud & Its Components

Virtual Private Cloud (VPC)

- Virtual Private Cloud (VPC) allows you to launch resources into a virtual network you have defined.
- A VPC is a virtual network that belongs to you and is logically isolated from virtual networks that belong to other users.
- The default VPC on both AWS and GCP will include an internet gateway.
- An internet gateway enables your instances to connect to the internet.

VPC resource isolation resolves issues related to security, privacy, compliance, and performance in cloud computing environments by providing a dedicated and isolated network environment for your resources.

Virtual Private Cloud (VPC) contd.

A VPC allows you to create or configure following:

- firewall
- routing table
- public or private subnets
- network gateway

Elastic Network Interfaces

An elastic network interface (referred to as a network interface in this documentation) is a virtual network interface that can include the following attributes:

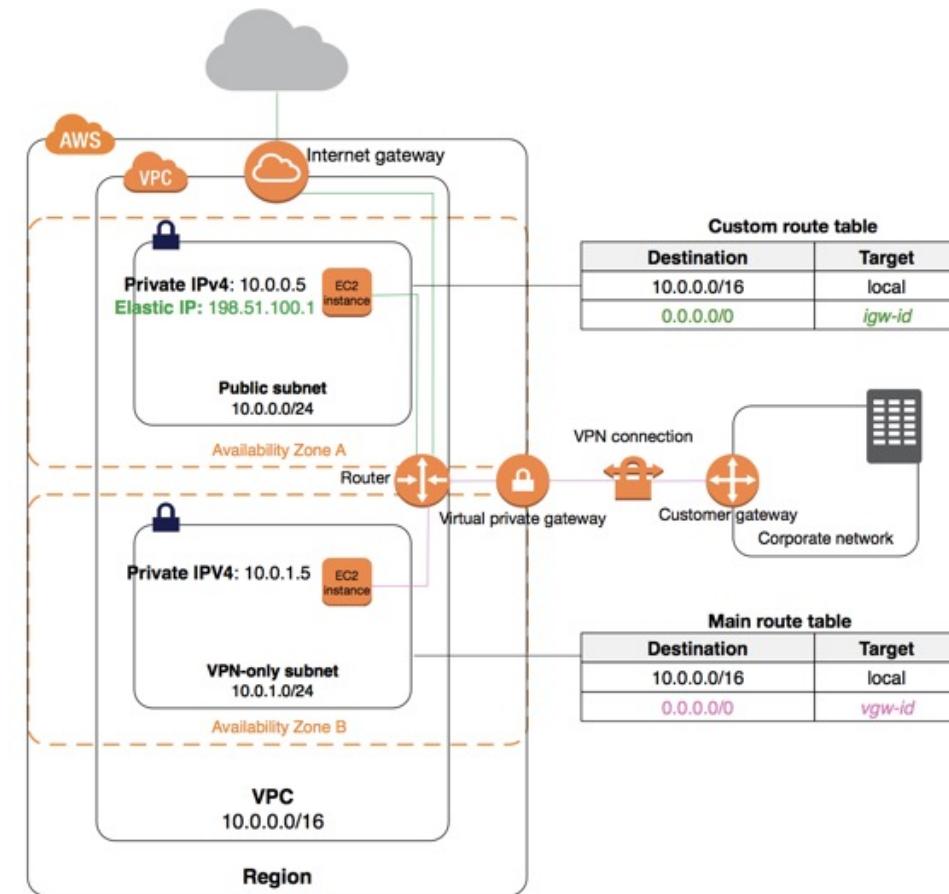
1. a primary private IPv4 address
2. one or more secondary private IPv4 addresses
3. one Elastic IP address per private IPv4 address
4. one public IPv4 address, which can be auto-assigned to the network interface for eth0 when you launch an instance
5. one or more IPv6 addresses
6. one or more security groups
7. a MAC address source/destination check flag
8. a description

Elastic Network Interfaces contd.

- You can create a network interface, attach it to an instance, detach it from an instance, and attach it to another instance.
- A network interface's attributes follow it as it is attached or detached from an instance and reattached to another instance.
- When you move a network interface from one instance to another, network traffic is redirected to the new instance.
- Each instance in your VPC has a default network interface (the primary network interface) that is assigned a private IPv4 address from the IPv4 address range of your VPC.
- You cannot detach a primary network interface from an instance.
- You can create and attach an additional network interface to any instance in your VPC. The number of network interfaces you can attach varies by instance type.

Route Tables

- A route table contains a set of rules, called routes, that are used to determine where network traffic is directed.
- Each subnet in your VPC must be associated with a route table; the table controls the routing for the subnet.
- A subnet can only be associated with one route table at a time, but you can associate multiple subnets with the same route table.



Route Table contd.

Destination	Target
10.0.0.0/16	Local
172.31.0.0/16	pcx-1a2b1a2b
0.0.0.0/0	igw-11aa22bb

VPC
peering
(specific)
Internet
Gateway

- Any traffic from the subnet that's destined for the 172.31.0.0/16 IP address range uses the peering connection, because this route is more specific than the route for Internet gateway.
- Any traffic destined for a target within the VPC (10.0.0.0/16) is covered by the Local route, and therefore routed within the VPC.
- All other traffic from the subnet uses the Internet gateway.

Internet Gateways

Redundancy is implemented to increase reliability, fault tolerance, and availability of services.

In simpler terms, a redundant Internet Gateway means that there is more than one gateway available to handle internet traffic, and if one gateway fails, there are backups in place to ensure uninterrupted connectivity and service availability.

- An Internet gateway is a horizontally scaled, redundant, and highly available VPC component that allows communication between instances in your VPC and the Internet.
- It therefore imposes no availability risks or bandwidth constraints on your network traffic.
- An Internet gateway serves two purposes:
 - to provide a target in your VPC route tables for Internet-routable traffic, and
 - to perform network address translation (NAT) for instances that have been assigned public IPv4 addresses.
- An Internet gateway supports IPv4 and IPv6 traffic.

Elastic IP Addresses

- An Elastic IP address is a static, public IPv4 address designed for dynamic cloud computing.
- You can associate an Elastic IP address with any instance or network interface for any VPC in your account.
- With an Elastic IP address, you can mask the failure of an instance by rapidly remapping the address to another instance in your VPC.
- AWS does not support Elastic IP addresses for IPv6.

Security Groups

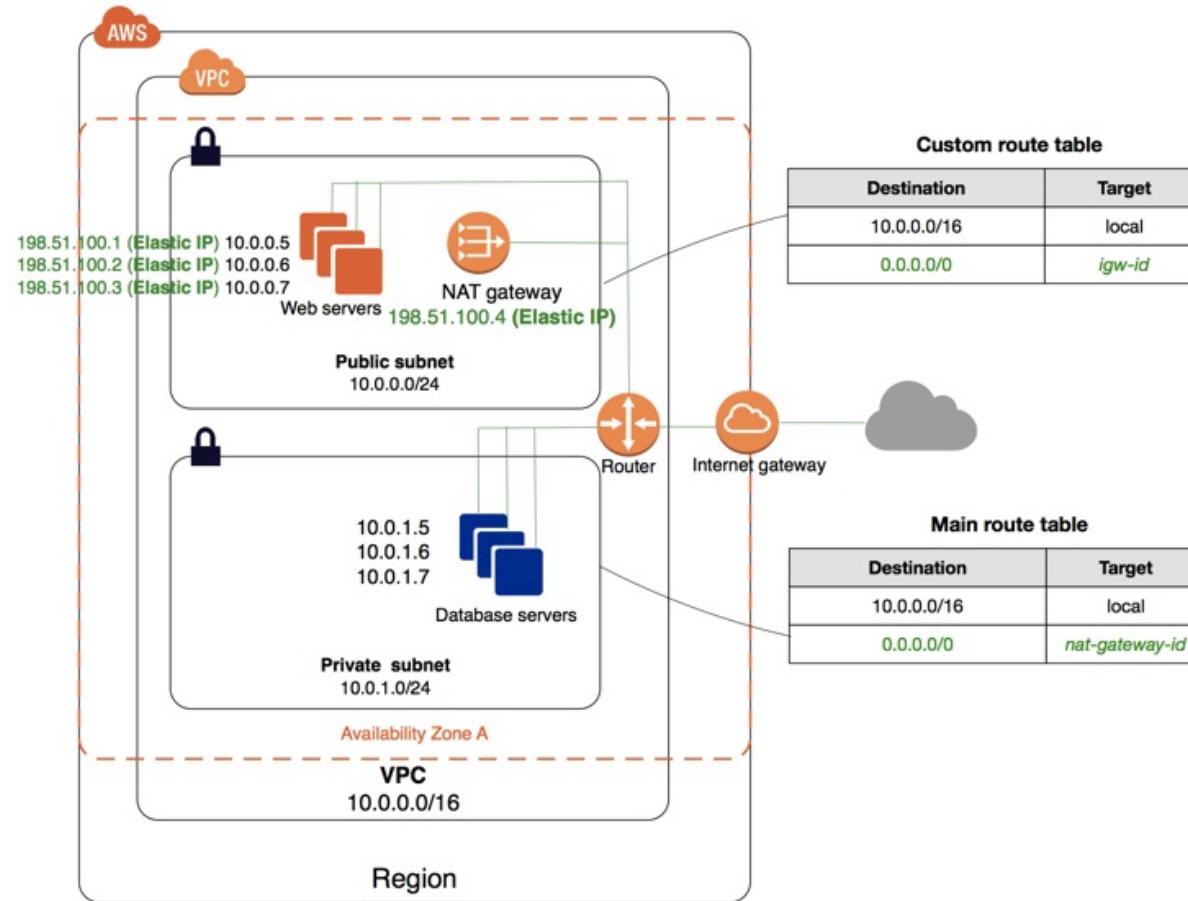
- A security group acts as a virtual firewall for your instance to control inbound and outbound traffic.
- When you launch an instance in a VPC, you can assign up to five security groups to the instance.
- Security groups act at the instance level, not the subnet level. Therefore, each instance in a subnet in your VPC could be assigned to a different set of security groups. If you don't specify a particular group at launch time, the instance is automatically assigned to the default security group for the VPC.
- For each security group, you add rules that control the inbound traffic to instances, and a separate set of rules that control the outbound traffic.

Ingress -> traffic incoming restrictions
Egress -> traffic outgoing restrictions

Security Groups Example

Inbound			
Source	Protocol	Port Range	Comments
0.0.0.0/0	TCP	80	Allow inbound HTTP access from all IPv4 addresses
::/0	TCP	80	Allow inbound HTTP access from all IPv6 addresses
0.0.0.0/0	TCP	443	Allow inbound HTTPS access from all IPv4 addresses
::/0	TCP	443	Allow inbound HTTPS access from all IPv6 addresses
Your network's public IPv4 address range	TCP	22	Allow inbound SSH access to Linux instances from IPv4 IP addresses in your network (over the Internet gateway)
Your network's public IPv4 address range	TCP	3389	Allow inbound RDP access to Windows instances from IPv4 IP addresses in your network (over the Internet gateway)
Outbound			
Destination	Protocol	Port Range	Comments
The ID of the security group for your database servers	TCP	1433	Allow outbound Microsoft SQL Server access to instances in the specified security group
The ID of the security group for your MySQL database servers	TCP	3306	Allow outbound MySQL access to instances in the specified security group

VPC with Public and Private Subnets



Additional Resources

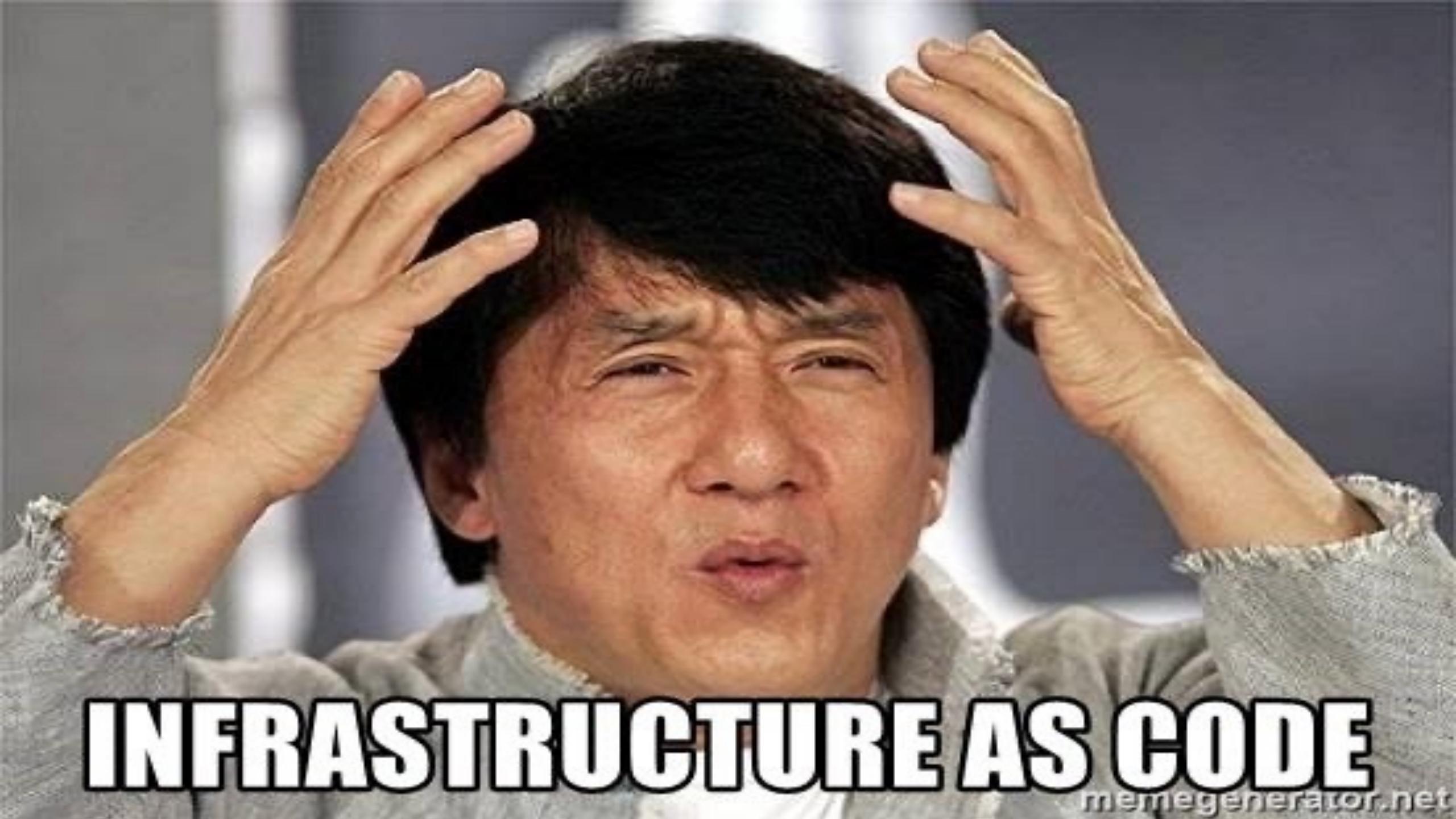
See Lecture Page

Infrastructure as Code (IaC)

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225

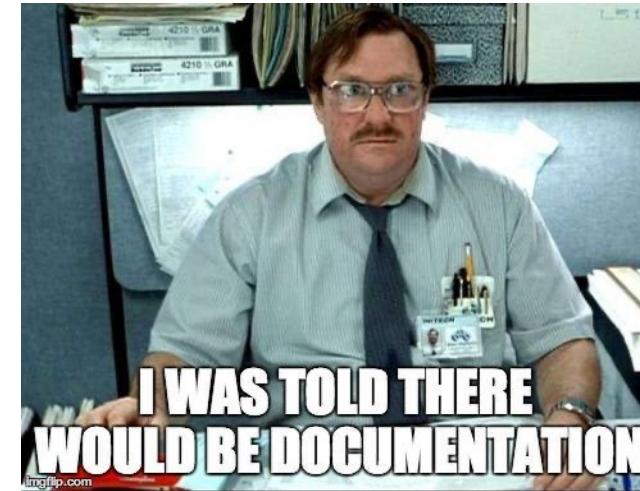
Northeastern University



INFRASTRUCTURE AS CODE

Why Infrastructure as Code?

- Virtualization, cloud, containers, server automation, and software-defined networking should simplify IT operations work.
- It should take less time and effort to provision, configure, update, and maintain services.
- Problems should be quickly detected and resolved, and systems should all be consistently configured and up to date.
- IT staff should spend less time on routine drudgery, having time to rapidly make changes and improvements to help their organizations meet the ever-changing needs of the modern world.



menial

Why Infrastructure as Code?

- IT operations teams find that they can't keep up with their daily workload.
- They don't have the time to fix longstanding problems with their systems, much less revamp them to make the best use of new tools.
- In fact, cloud and automation often makes things worse.
- The ease of provisioning new infrastructure leads to an ever-growing portfolio of systems, and it takes an ever-increasing amount of time just to keep everything from collapsing.

Why Infrastructure as Code?

- Adopting cloud and automation tools immediately lowers barriers for making changes to infrastructure.
- Managing changes in a way that improves consistency and reliability doesn't come out of the box with the software. It takes people to think through how they will use the tools and put in place the systems, processes, and habits to use them effectively.
- Change management processes are commonly ignored, bypassed, or overruled by people who need to get things done.
- Organizations that are more successful in enforcing these processes are increasingly seeing themselves outrun by more technically nimble competitors.

What Is Infrastructure as Code?

- Infrastructure as code is an approach to infrastructure automation based on practices from software development.
- It emphasizes consistent, repeatable routines for provisioning and changing systems and their configuration.
- Changes are made to definitions and then rolled out to systems through unattended processes that include thorough validation.

What Is Infrastructure as Code?

- Modern tooling can treat infrastructure as if it were software and data.
- Apply software development tools such as version control systems (VCS), automated testing libraries, and deployment orchestration to manage infrastructure.
- It also opens the door to exploit development practices such as test-driven development (TDD), continuous integration (CI), and continuous delivery (CD).



Goals of Infrastructure as Code

- To support changes smoothly.
- To ensure system changes are routine, stress-free for users and IT staff, so that IT staff can focus on engaging tasks, not repetitive ones.
- To ensure users can manage resources without IT assistance.
- To allow teams recover quickly from failures.
- Continuous improvements are prioritized over big bang projects (more intensive)
- Solutions are tested rather than discussed endlessly.

- IT infrastructure supports and enables change, rather than being an obstacle or a constraint.
- Changes to the system are routine, without drama or stress for users or IT staff.
- IT staff spends their time on valuable things that engage their abilities, not on routine, repetitive tasks.
- Users are able to define, provision, and manage the resources they need, without needing IT staff to do it for them.
- Teams are able to easily and quickly recover from failures, rather than assuming failure can be completely prevented.
- Improvements are made continuously, rather than done through expensive and risky “big bang” projects.
- Solutions to problems are proven through implementing, testing, and measuring them, rather than by discussing them in meetings and documents.

Challenges with Dynamic Infrastructure

1. Stability, security and reliability is more important than getting on board with new-latest technology
2. Learn how a server works based on the components discussed in the specifications at
https://www.dell.com/en-us/shop/servers-storage-and-networking/poweredge-r550-rack-server/spd/poweredge-r550/pe_r550_tmb_vi_vp

Server Sprawl

- Cloud and virtualization can make it **very easy** to provision new servers from a pool of resources. This can lead to the number of servers growing faster than the ability of the team to manage them.
- When this happens, teams struggle to keep servers patched and up to date, leaving systems vulnerable to known exploits.
- When problems are discovered, fixes may not be rolled out to all of the systems that could be affected by them.
- Differences in versions and configurations across servers mean that software and scripts that work on some machines don't work on others.
- This leads to inconsistency across the servers, called *configuration drift*.

Configuration Drift

- Even when servers are initially created and configured consistently, differences can creep in over time:
- Someone makes a fix to one of the Oracle servers to fix a specific user's problem, and now it's different from the other Oracle servers.
- A new version of JIRA needs a newer version of Java, but there's not enough time to test all of the other Java-based applications so that everything can be upgraded.
- Three different people install IIS on three different web servers over the course of a few months, and each person configures it differently.
- One JBoss server gets more traffic than the others and starts struggling, so someone tunes it, and now its configuration is different from the other JBoss servers.
- Being different isn't bad. The heavily loaded JBoss server probably should be tuned differently from ones with lower levels of traffic. But variations should be captured and managed in a way that makes it easy to reproduce and to rebuild servers and services.
- Unmanaged variation between servers leads to *snowflake servers* and *automation fear*.

Snowflake Servers

- A snowflake server is different from any other server on your network. It's special in ways that can't be replicated.
- The problem is when the team that owns the server doesn't understand how and why it's different, and wouldn't be able to rebuild it. An operations team should be able to confidently and quickly rebuild any server in their infrastructure. If any server doesn't meet this requirement, constructing a new, reproducible process that can build a server to take its place should be a leading priority for the team.

Fragile Infrastructure

- A fragile infrastructure is easily disrupted and not easily fixed. This is the snowflake server problem expanded to an entire portfolio of systems.
- The solution is to migrate everything in the infrastructure to a reliable, reproducible infrastructure, one step at a time. The Visible Ops Handbook³ outlines an approach for bringing stability and predictability to a difficult infrastructure.
- DON'T TOUCH THAT SERVER. DON'T POINT AT IT. DON'T EVEN LOOK AT IT.



Principles of Infrastructure as Code

Systems Can Be Easily Reproduced

- It should be possible to effortlessly and reliably rebuild any element of an infrastructure. Effortlessly means that there is no need to make any significant decisions about how to rebuild the thing. Decisions about which software and versions to install on a server, how to choose a hostname, and so on should be captured in the scripts and tooling that provision it.
- The ability to effortlessly build and rebuild any part of the infrastructure is powerful. It removes much of the risk, and fear, when making changes. Failures can be handled quickly and with confidence. New services and environments can be provisioned with little effort.
- Approaches for reproducibly provisioning servers and other infrastructure elements are discussed in Part II of this book.

Systems Are Disposable

- One of the benefits of dynamic infrastructure is that resources can be easily created, destroyed, replaced, resized, and moved. In order to take advantage of this, systems should be designed to assume that the infrastructure will always be changing. Software should continue running even when servers disappear, appear, and when they are resized.
- The ability to handle changes gracefully makes it easier to make improvements and fixes to running infrastructure. It also makes services more tolerant to failure. This becomes especially important when sharing large-scale cloud infrastructure, where the reliability of the underlying hardware can't be guaranteed.

CATTLE, NOT PETS

A popular expression is to “treat your servers like cattle, not pets”



- Pets are given names
- Pets are unique, lovingly raised and cared for
- When they get ill, you nurse them back to health
- Cattles are given numbers
- Cattles are almost identical to other cattle
- When they get ill, you get another one

A fundamental difference between the iron age and cloud age is the move from unreliable software, which depends on the hardware to be very reliable, to software that runs reliably on unreliable hardware.

Systems Are Consistent

- Given two infrastructure elements providing a similar service—for example, two application servers in a cluster—the servers should be nearly identical. Their system software and configuration should be the same, except for those bits of configuration that differentiate them, like their IP addresses.
- Letting inconsistencies slip into an infrastructure keeps you from being able to trust your automation. This encourages doing special things for servers that don't quite match, which leads to unreliable automation.
- Teams that implement the reproducibility principle can easily build multiple identical infrastructure elements.
- Being able to build and rebuild consistent infrastructure helps with configuration drift.

Processes Are Repeatable

- Building on the reproducibility principle, any action you carry out on your infrastructure should be repeatable.
 - This is an obvious benefit of using scripts and configuration management tools rather than making changes manually, but it can be hard to stick to doing things this way.
- Effective infrastructure teams have a strong scripting culture.
 - If a task can be scripted, script it.
 - If a task is hard to script, drill down and see if there's a technique or tool that can help, or whether the problem the task is addressing can be handled in a different way.

Design Is Always Changing

- With iron-age IT, making a change to an existing system is difficult and expensive. So limiting the need to make changes to the system once it's built makes sense. This leads to the need for comprehensive initial designs that take various possible requirements and situations into account.
- Because it's impossible to accurately predict how a system will be used in practice, and how its requirements will change over time, this approach naturally creates overly complex systems. Ironically, this complexity makes it more difficult to change and improve the system, which makes it less likely to cope well in the long run.
- With cloud-age dynamic infrastructure, making a change to an existing system can be easy and cheap. However, this assumes everything is designed to facilitate change. Software and infrastructure must be designed as simply as possible to meet current requirements. Change management must be able to deliver changes safely and quickly.
- The most important measure to ensure that a system can be changed safely and quickly is to make changes frequently. This forces everyone involved to learn good habits for managing changes, to develop efficient, streamlined processes, and to implement tooling that supports doing so.

Practicing Infrastructure as Code



Stack

- A stack is a defined collection of infrastructure elements, regardless of size, managed as a unit.
- It can range from a single server to an entire data center.
- Historically, manual infrastructures didn't emphasize stack concepts, relying on organic growth.
- Automation tools necessitate explicit infrastructure groupings, challenging traditional networking boundaries.
- Various organizing approaches exist beyond network boundaries, urging consideration of more effective patterns.

- A stack is a collection of infrastructure elements that are defined as a unit.
- A stack can be any size. It could be a single server. It could be a pool of servers with their networking and storage. It could be all of the servers and other infrastructure involved in a given application. Or it could be everything in an entire data center. What makes a set of infrastructure elements a stack isn't the size, but whether it's defined and changed as a unit.
- The concept of a stack hasn't been commonly used with manually managed infrastructures. Elements are added organically, and networking boundaries are naturally used to think about infrastructure groupings.
- But automation tools force more explicit groupings of infrastructure elements. It's certainly possible to put everything into one large group. And it's also possible to structure stacks by following network boundaries. But these aren't the only ways to organize stacks.
- Rather than simply replicating patterns and approaches that worked with iron-age static infrastructure, it's worth considering whether there are more effective patterns to use.

Environments

- The term “environment” is typically used when there are multiple stacks that are actually different instances of the same service or set of services.
- An application or service may have “development,” “test,” “preproduction,” and “production” environments.
- The infrastructure for these should generally be the same, with perhaps some variations due to scale.
- Keeping multiple environments configured consistently is a challenge for most IT organizations, and one which infrastructure as code is particularly well suited to address.

VCS for Infrastructure Management

- A version control system (VCS) is a core part of an infrastructure-as-code regime.
- A VCS provides traceability of changes, rollback, correlation of changes to different elements of the infrastructure, visibility, and can be used to automatically trigger actions such as testing.

What to Manage in a VCS

- Put everything in version control that is needed to build and rebuild elements of your infrastructure.
- Ideally, if your entire infrastructure were to disappear other than the contents of version control, you could check everything out and run a few commands to rebuild everything, probably pulling in backup data files as needed.
- Some of the things that may be managed in VCS include:
 - Configuration definitions (cookbooks, manifests, playbooks, etc.)
 - Configuration files and templates
 - Test code
 - CI and CD job definitions
 - Utility scripts
 - Source code for compiled utilities and applications
 - Documentation

What NOT to Manage in a VCS

- Things that might not need to be managed in the VCS include the following:
 - Software artifacts should be stored in a repository (e.g., a Maven repository for Java artifacts, an APT or YUM repository, etc.). These repositories should be backed up or have scripts (in VCS) that can rebuild them.
 - Software and other artifacts that are built from elements already in the VCS don't need to be added to the VCS themselves, if they can be reliably rebuilt from source.
 - Data managed by applications, logfiles, and the like don't belong in VCS. They should be stored, archived, and/or backed up as relevant. Chapter 14 covers this in detail.
 - Passwords and other security secrets should never be stored in a VCS. Tools for managing encrypted keys and secrets in an automated infrastructure should be used instead.

Additional Resources

See Lecture Page

Infrastructure as Code w/Terraform

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225

Northeastern University

Install Terraform

- Terraform is distributed as a single binary.
- Install Terraform by unzipping it and moving it to a directory included in your system's PATH .

Setup Environment for AWS

- For Terraform to be able to make changes in your AWS account, you will need to set the AWS credentials for the IAM user as the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.

```
$ export AWS_ACCESS_KEY_ID=(your access key id)
$ export AWS_SECRET_ACCESS_KEY=(your secret access key)
```

- In addition to environment variables, Terraform supports the same authentication mechanisms as all AWS CLI and SDK tools. Therefore, it'll also be able to use credentials in `$HOME/.aws/credentials`.

Writing Terraform Files

- Terraform code is written in the HashiCorp Configuration Language (HCL) in files with the extension `.tf`.
- You can write Terraform code in just about any text editor.

Provider

- The first step to using Terraform is to configure the provider(s) you want to use.

```
provider "aws" {  
    region = "us-east-2"  
}
```

Initialize Working Directory

- The terraform binary contains the basic functionality for Terraform, but it does not come with the code for any of the providers (e.g., the AWS provider, Azure provider, GCP provider, etc.), so when you're first starting to use Terraform, you need to run `terraform init` to tell Terraform to scan the code, figure out which providers you're using, and download the code for them.
- By default, the provider code will be downloaded into a `.terraform` folder, which is Terraform's scratch directory (you may want to add it to `.gitignore`).
- You need to run `init` any time you start with new Terraform code.
- It is safe to run `init` multiple times (the command is idempotent).

Create an Execution Plan

- The `terraform plan` command is used to create an execution plan.
- Terraform performs a refresh, unless explicitly disabled, and then determines what actions are necessary to achieve the desired state specified in the configuration files.
- This command is a convenient way to check whether the execution plan for a set of changes matches your expectations without making any changes to real resources or to the state.
- For example, `terraform plan` might be run before committing a change to version control, to create confidence that it will behave as expected.

Apply Changes

The terraform apply command is used to apply the changes required to reach the desired state of the configuration, or the pre-determined set of actions generated by a terraform plan execution plan.



Terraform State

- Terraform must store state about your managed infrastructure and configuration.
- This state is used by Terraform to map real world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures.
- This state is stored by default in a local file named "terraform.tfstate", but it can also be stored remotely, which works better in a team environment.
- Terraform uses this local state to create plans and make changes to your infrastructure.
- Prior to any operation, Terraform does a refresh to update the state with the real infrastructure.
- Every time you run Terraform, it can fetch the latest status of resource from AWS and compare that to what's in your Terraform configurations to determine what changes need to be applied.
- The output of the plan command is a diff between the code on your computer and the infrastructure deployed in the real world, as discovered via IDs in the state file.

Destroy Terraform Managed Infrastructure

- The `terraform destroy` command is used to destroy the Terraform-managed infrastructure.

Configuring .gitignore

.terraform

*.tfstate

*.tfstate.backup

Git will ignore the *.terraform* folder, which Terraform uses as a temporary scratch directory, as well as *.tfstate files, which Terraform uses to store state.

Resources & Modules

- The main purpose of the Terraform language is declaring resources.
- All other language features exist only to make the definition of resources more flexible and convenient.
- A group of resources can be gathered into a *module*, which creates a larger unit of configuration.
- A resource describes a single infrastructure object, while a module might describe a set of objects and the necessary relationships between them in order to create a higher-level system.
- A Terraform configuration consists of a root module, where evaluation begins, along with a tree of child modules created when one module calls another.

Resource

- The general syntax for creating a resource in Terraform is

```
resource "<PROVIDER>_<TYPE>" "<NAME>" {  
    [CONFIG ...]  
}
```

Referencing Resources

Syntax: <PROVIDER>_<TYPE>.<NAME>.<ATTRIBUTE>

Variables

Declaring an Input Variable

Each input variable accepted by a module must be declared using a `variable` block:

```
variable "image_id" {
  type = string
}

variable "availability_zone_names" {
  type    = list(string)
  default = ["us-west-1a"]
}

variable "docker_ports" {
  type = list(object({
    internal = number
    external = number
    protocol = string
  }))
  default = [
    {
      internal = 8300
      external = 8300
      protocol = "tcp"
    }
  ]
}
```

The label after the `variable` keyword is a name for the variable, which must be unique among all variables in the same module. This name is used to assign a value to the variable from outside and to reference the variable's value from within the module.

Using Input Variable Values

Within the module that declared a variable, its value can be accessed from within `expressions` as `var.<NAME>`, where `<NAME>` matches the label given in the declaration block:

```
resource "aws_instance" "example" {
  instance_type = "t2.micro"
  ami           = var.image_id
}
```

The value assigned to a variable can be accessed only from expressions within the module where it was declared.

Variable – Type Constraints

- The type argument in a variable block allows you to restrict the type of value that will be accepted as the value for a variable.
- If no type constraint is set then a value of any type is accepted.
- While type constraints are optional, it is recommended specifying them; they serve as easy reminders for users of the module, and allow Terraform to return a helpful error message if the wrong type is used.

Additional Resources

See Lecture Page

Virtualization

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225

Northeastern University

Virtualization

Understanding Virtualization

In computing, virtualization refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources.

Bin packing -> most num of items in a bag -> how to make & utilize every bit of a server

Immutable Server Deployments: TODO -> Will come for midterm
for each env -> dev, stage, pre-prod, prod (diff in size, pre-prod mimics the prod)
Server has > 1 TB storage

Never Ssh and run commands in the VM, if instance becomes bad then we can just destroy it as long as it is stateless

Bare metal server: it means we are running OS on the physical hardware

Limitations of Bare-metal (physical) Server

Servers are designed to run one Operating System and application at a time.

Physical Server has only a single OS

But we can't test OS Compatibility using a single OS -> applications need to be run on diff OS for diff users

Stability -> new tech needs to run in diff OS (and even old OS, older versions are even more secure)

Reliability -> disaster recovery, needs to have a backup (maybe physical server) that we need to switch back to

Security ->

Benefits of Virtualization

- Effective way to reduce IT expenses while boosting efficiency and agility for all size businesses
- Reduce capital and operating costs.
- Minimize or eliminate downtime.
- Increase IT productivity, efficiency, agility and responsiveness.
- Provision applications and resources faster.
- Enable business continuity and disaster recovery.
- Simplify data center management.
- Build a true Software-Defined Data Center.

Flexibility to scale -> (up or down) servers

How Virtualization Works

- Virtualization uses software to simulate the existence of hardware and create a virtual computer system.
- Doing this allows businesses to run more than one virtual system and multiple operating systems and applications on a single server.
- This can provide economies of scale and greater efficiency.

The Virtual Machine

- A virtual computer system is known as a “virtual machine” (VM): a tightly isolated software container with an operating system and application inside.
- Each self-contained VM is completely independent.
- Putting multiple VMs on a single computer enables several operating systems and applications to run on just one physical server, or “host”.

Hypervisor

A thin layer of software called a hypervisor decouples the virtual machines from the host and dynamically allocates computing resources to each virtual machine as needed.

We install hypervisor (it is a lightweight special OS) that manages resources on a server

2 types of hypervisor -> one that uses existing OS to manage resources and other where hypervisor is directly installed on the server

Key Properties of Virtual Machines

- Partitioning
 - Run multiple operating systems on one physical machine
 - Divide system resources between virtual machines
- Isolation
 - Provide fault and security isolation at the hardware level
 - Preserve performance with advanced resource controls
- Encapsulation
 - Save the entire state of a virtual machine to files
 - Move and copy virtual machines as easily as moving and copying files
- Hardware Independence
 - Provision or migrate any virtual machine to any physical server

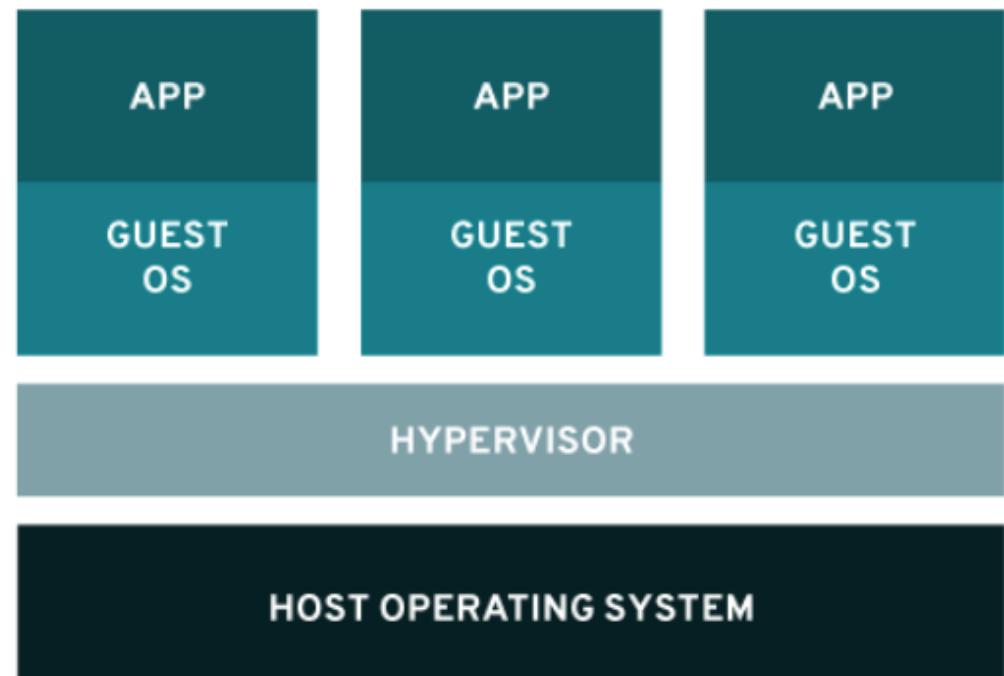
Server Consolidation

Using server virtualization, a company can maximize the use of its server resources and reduce the number of servers required. The result is server consolidation, which improves efficiency and cuts costs.

Virtualization Is Not Cloud Computing

- Cloud computing is not the same thing as virtualization; rather, it's something you can do using virtualization.
- Cloud computing describes the delivery of shared computing resources (software and/or data) on demand through the Internet.
- Whether or not you are in the cloud, you can start by virtualizing your servers and then move to cloud computing for even more agility and increased self-service.

VIRTUALIZATION



Infrastructure as a service (IaaS)

Amazon Elastic Compute Cloud (Amazon EC2) & Google Compute Engine

Amazon EC2

- Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.
- Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction.
- Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change.
- Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use.
- Amazon EC2 provides developers the tools to build failure resilient applications and isolate them from common failure scenarios.
- **Amazon EC2 is a virtual machine in the AWS cloud platform.**

Google Compute Engine

- Compute Engine instances can run the public images for Linux and Windows Server that Google provides as well as private custom images that you can create or import from your existing systems.
- Regardless of the region where you create your VM instance, the default time for your VM instance is Coordinated Universal Time (UTC).

Benefits of VMs in Cloud

- Elastic Computing
- Complete Control
- Flexibility
- Reliable
- Secure
- Pay-as-you-go
- Integrated*
- Easy to Start*

VM Machine Types

Instance/Machine Types

- AWS & GCP provides a wide selection of instance types optimized to fit different use cases.
- Instance types comprise varying combinations of CPU, memory, storage, and networking capacity and give you the flexibility to choose the appropriate mix of resources for your applications.
- Each instance type includes one or more instance sizes, allowing you to scale your resources to the requirements of your target workload.

Amazon EC2 Instance Type Categories

- General Purpose
- Compute Optimized
- Memory Optimized
- Accelerated Computing (GPU)
- Storage Optimized



Dedicated Instances

- Dedicated Instances are Amazon EC2 instances that run in a VPC on hardware that's dedicated to a single customer.
- Your Dedicated instances are physically isolated at the host hardware level from instances that belong to other AWS accounts.
- Dedicated instances may share hardware with other instances from the same AWS account that are not Dedicated instances.

On cloud: many resources are shared - compute with another or memory or disk

Dedicated Hosts

- A Dedicated Host is a physical EC2 server dedicated for your use.
- Dedicated Hosts can help you reduce costs by allowing you to use your existing server-bound software licenses, including Windows Server, SQL Server, and SUSE Linux Enterprise Server (subject to your license terms), and can also help you meet compliance requirements

Vertical Scaling -> always have a downtime -> say RDS

Application -> can be horizontally scaled -> Application servers are stateless

Auto scaler -> only performs horizontal scaling

Size of machines -> bigger ones -> have more bandwidth, smaller ones have throttled

Dedicated Hosts vs. Dedicated Instances

Characteristic	Dedicated Instances	Dedicated Hosts
Enables the use of dedicated physical servers	x	x
Per instance billing (subject to a \$2 per region fee)	x	
Per host billing		x
Visibility of sockets, cores, host ID		x
Affinity between a host and instance		x
Targeted instance placement		x
Automatic instance placement	x	x
Add capacity using an allocation request		x

GCE Machine Types

- **General-purpose** —best price-performance ratio for a variety of workloads.
- **Compute-optimized** —highest performance per core on Compute Engine and optimized for compute-intensive workloads.
- **Memory-optimized** —ideal for memory-intensive workloads, offering more memory per core than other machine families, with up to 12 TB of memory.
- **Accelerator-optimized** —ideal for massively parallelized Compute Unified Device Architecture (CUDA) compute workloads, such as machine learning (ML) and high performance computing (HPC). This family is the best option for workloads that require GPUs.

Pricing Model

AWS EC2 Pricing Model



Amazon EC2 Pricing

Various pricing models

- On-Demand
- Spot Instances
- Reserved Instances
- Dedicated Hosts
- Dedicated Instances

On-Demand Instances

- With On-Demand instances, you pay for compute capacity by the hour with no long-term commitments or upfront payments. You can increase or decrease your compute capacity depending on the demands of your application and only pay the specified hourly rate for the instances you use.
- On-Demand instances are recommended for:
 - Users that prefer the low cost and flexibility of Amazon EC2 without any up-front payment or long-term commitment
 - Applications with short-term, spiky, or unpredictable workloads that cannot be interrupted
 - Applications being developed or tested on Amazon EC2 for the first time

Reserved Instances

- Reserved Instances provide you with a significant discount (up to 75%) compared to On-Demand instance pricing.
- In addition, when Reserved Instances are assigned to a specific Availability Zone, they provide a capacity reservation, giving you additional confidence in your ability to launch instances when you need them.
- For applications that have steady state or predictable usage, Reserved Instances can provide significant savings compared to using On-Demand instances.
- Reserved Instances are recommended for:
 - Applications with steady state usage
 - Applications that may require reserved capacity
 - Customers that can commit to using EC2 over a 1 or 3 year term to reduce their total computing costs

Spot Instances

- Amazon EC2 Spot instances allow you to bid on spare Amazon EC2 computing capacity for up to 90% off the On-Demand price.
- Spot instances are recommended for:
 - Applications that have flexible start and end times
 - Applications that are only feasible at very low compute prices
 - Users with urgent computing needs for large amounts of additional capacity

GCE Pricing Model

GCE Billing Model

- All vCPUs, GPUs, and GB of memory are charged a minimum of *1 minute*. For example, if you run your virtual machine for 30 seconds, you will be billed for 1 minute of usage.
- After 1 minute, instances are charged in *1 second increments*.

<https://cloud.google.com/compute/vm-instance-pricing>



Google Cloud

GCE Discounts

vCPU and memory usage for each machine type use the on-demand price unless that usage qualifies for a discount. vCPU and memory usage for each machine type can receive one of the following discounts:

- **Discounts for Spot VMs** (and preemptible VMs): automatic discount between 60-91%. Spot prices change over time, but the discounts for machine types are always in this range.
- **Committed use discounts (CUDs)**: up to a 70% discount for memory-optimized machine types and up to a 57% discount for all other machine types.
- **Sustained use discounts (SUDs)**: automatic discount of up to 30% on resources that are used for more than 25% of a month and are not receiving any other discounts.
- **Discount types cannot be combined**. Spot prices apply to all Spot VMs (and preemptible VMs), so Spot VMs (and preemptible VMs) cannot receive sustained use discounts or CUDs.

Cloud VM Features

- Secure login using key pairs
- Temporary & Permanent Storage Volumes
- Security Groups (firewalls)
- Static IPv4 address (Elastic IP address)
- Metadata, known as tags
- Subnets (virtual networks)

Should never have pwd based auth
Ex: Digital Ocean: should not give access of root server ->not how you deploy

Cheaper way is to pay for a NAT gateway -> instead of paying for an IP Address

Machine Images

Custom Machine Images

Additional Resources

See Lecture Page

Hashicorp Packer & Custom Machine Images

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225

Northeastern University

Creating Your Own AMI

- You can launch an instance from an existing AMI, customize the instance, and then save this updated configuration as a custom AMI.
- Instances launched from this new custom AMI include the customizations that you made when you created the AMI.

First Security team installs
Then Sources of the company
Then lang - java, node, etc
Then dependencies & applications

HashiCorp Packer

- Packer is an open-source tool for creating identical machine images for multiple platforms from a single source configuration.

Packer is basically Image as Code

Always include vendor provided actions while working on workflows

Custom Machine Image Use Cases

- **Golden Images** – Secure, immutable OS images are needed especially in Enterprise world where compliance issues matters.
- **Environment Parity** – Keep all dev/test/prod environment as similar as possible.
- **Auto-scaling Acceleration** – Launch completely provisioned and configured instances in seconds, rather than minutes or even hours.

Installing Packer

- <https://www.packer.io/intro/getting-started/install.html>
- Packer may be installed in the following ways:
 - Using a precompiled binary; We release binaries for all supported platforms and architectures. This method is recommended for most users.
 - Installing from source This method is only recommended for advanced users.
 - An unofficial alternative installation method

Build an Image – The Template

- The configuration file used to define what image we want built and how is called a *template* in Packer terminology.
- Packer uses the HashiCorp Configuration Language - **HCL** - designed to allow concise descriptions of the required steps to get to a build file.

Provide support with tags, which our key: value pairs

tags ([]string) - Assign network tags to apply firewall rules to VM instance.

Packer will setup Temporary SSH Key, and temporary firewalls -> you don't need to set it up

Packer Docs

- <https://www.packer.io/docs/templates>
 - https://www.packer.io/docs/templates/hcl_templates
 - https://www.packer.io/docs/templates/hcl_templates/blocks
 - https://www.packer.io/docs/templates/hcl_templates/functions
- <https://www.packer.io/plugins>
 - <https://www.packer.io/plugins/builders/amazon/ebs>
- <https://www.packer.io/docs/provisioners>
 - <https://www.packer.io/docs/provisioners/shell>

Compute engine resources are zonal

Book Disk Type > Balanced ... -> These are cheaper

For our application while writing the system script -> make sure to have all of the networking setup using the “after” keyword and then proceed to configure the rest

“After = network.target”

Additional Resources

In terraform repo -> you have to use something called as depends on

https://developer.hashicorp.com/terraform/language/meta-arguments/depends_on

Depends_on -> All resources that you have in your terraform template need to be there -> BE EXPLICIT

See Lecture Page

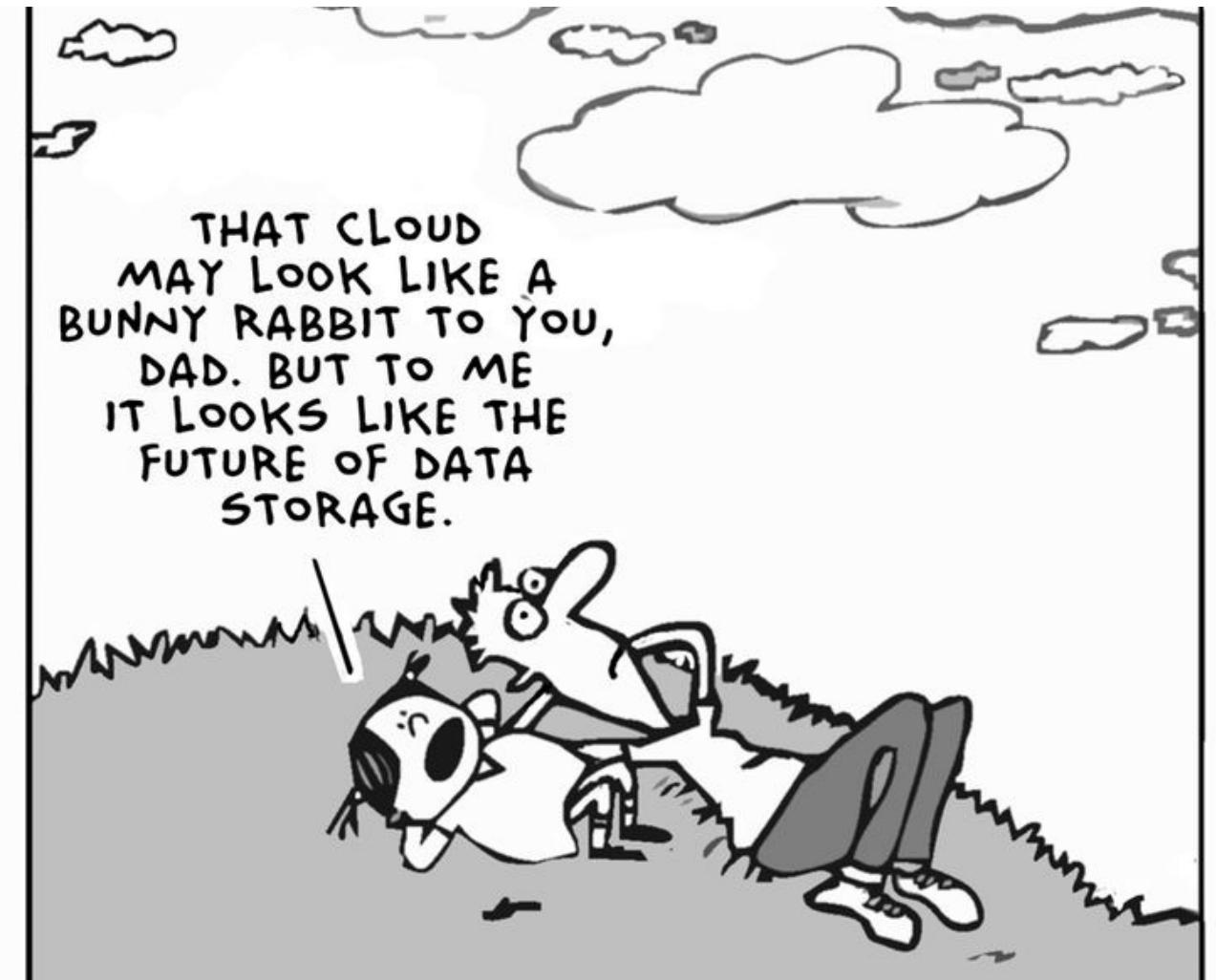
systemD -> will be there for midterm

Cloud Storage Solutions

CSYE 6225

Tejas Parikh t.parikh@northeastern.edu

Northeastern University



Cloud Storage Options

- Block-level Storage
- Object Storage Service
- Relational Databases
- NoSQL Databases
- (and more)

Mid term -> One question around different types of storages will be there

<https://www.pluralsight.com/resources/blog/cloud/storage-showdown-aws-vs-azure-vs-gcp-cloud-comparison>



Block-level Storage

Amazon EBS, Amazon EFS

What is Block-level Storage

- Block-level storage with a disk file system (FAT32, NTFS, ext3, ext4, XFS, and so on) can be used to store files as you do on a personal computer.
- A block is a sequence of bytes and the smallest addressable unit.
- The OS is the intermediary between the application that wants to access files and the underlying file system and block-level storage.
- The disk file system manages where (at what block address) your files are persisted on the underlying block-level storage.
- You can use block-level storage only in combination with an VM (EC2) instance where the OS runs.

OS (File System -
virtual concept)
->
Disk

Accessing Block-level Storage

- The OS provides access to block-level storage via open, write, and read system calls.
- The simplified flow of a read request goes like this:
 1. An application wants to read the file /path/to/file.txt and makes a read system call.
 2. The OS forwards the read request to the file system.
 3. The file system translates /path/to/file.txt to the block on the disk where the data is stored.

Why do we need Block-level Storage?

- Applications like databases that read or write files by using system calls must have access to block-level storage for persistence.
- You can't tell a MySQL database to store its files in an object store because MySQL uses system calls to access files.

Block-level Storage on AWS

AWS provides two kinds of block-level storage:

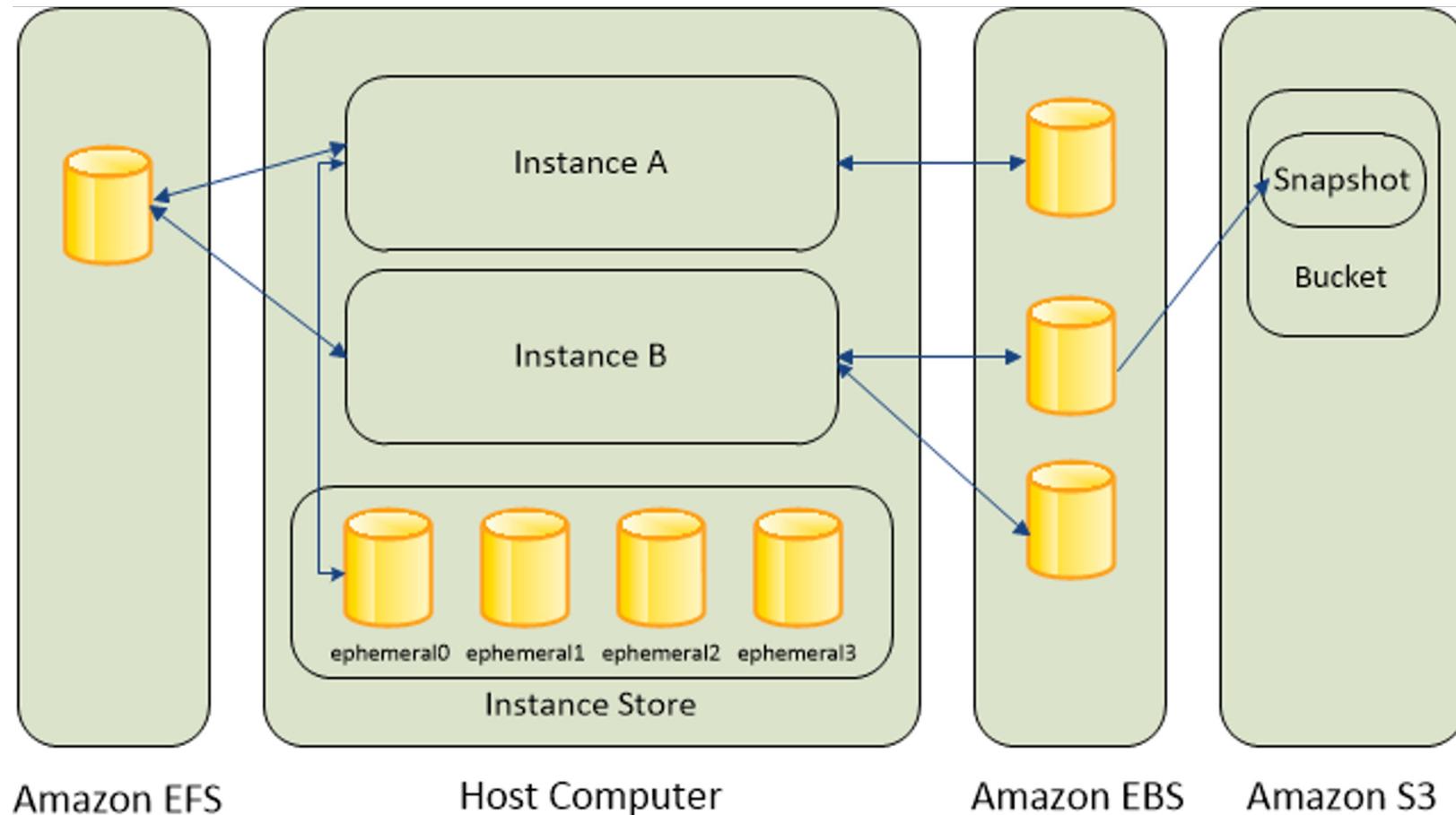
1. Network Attached Storage (NAS) via AWS EBS service
2. Instance Storage

under Compute Engine -> Storage

GCP: Persistent Disk (pd)
Solid State Disk (ssd)

Azure: Managed Disk

Amazon Elastic Block Storage



Amazon Elastic Block Store (Amazon EBS)

- Amazon Elastic Block Store (Amazon EBS) provides persistent block storage volumes for use with Amazon EC2 instances in AWS.
- You can think of EBS as disks that you attach to your virtual machine.

Disk is also zonal, we can only attach Disk to VM in the same zone

Persistent Disk -> Network Attached Storage (latency exists)

We are allowed to have snapshots

Multiple disks can be attached to a VM

If VM reboots -> disk data is persistent

Ex: elastic storage, cassandra

Amazon EBS Use Cases

Amazon EBS is an excellent choice for applications that need persistent block storage ideal for databases, data warehousing, big data applications and other low latency interactive applications that demand the highest IOPS or throughput and low latency with consistent, predictable performance.

low latency -> fast retrieval

IOPS -> input output operations per second

EBS Lifetime

- EBS Volumes are NOT part of your EC2 instances.
- EBS Volumes are attached to your EC2 instances via a network connection.
- You can choose to terminate EC2 instance without terminating your EBS volumes.
- An EBS volume can be attached to NO EC2 instance or one EC2 instance at a time.
- EBS volumes can be used like normal hard disks.

A disk cannot be attached to multiple VMs, it can be isolate or attached only to 1 VM

Availability

- Each Amazon EBS volume is designed for 99.999% availability.
- Each Amazon EBS volume is automatically replicated within its Availability Zone to protect you from component failure, offering high availability and durability.

There exists a backup, anything that you apply on the disk, same is duplicated to the stand-by backup disk. (You don't pay extra for it) This is done, if there is a hardware issue, Cloud Provider just gives you the backup machine

Snapshots

- Amazon EBS provides the ability to save point-in-time snapshots of your volumes to Amazon S3. *Auto: once a day (configurable), not chargeable, deleted when you delete the disk*
- Amazon EBS Snapshots are stored incrementally: only the blocks that have changed after your last snapshot are saved, and you are billed only for the changed blocks. *You can take manual snapshots, that is chargeable*
- Example: If you have a device with 100 GB of data but only 5 GB has changed after your last snapshot, a subsequent snapshot consumes only 5 additional GB and you are billed only for the additional 5 GB of snapshot storage, even though both the earlier and later snapshots appear complete. *1s - D1 - 2s - D2 - 3s - D3 - 4s : D3 = D2 + D1 (doesn't take diff from 4s - 1s, takes delta diff (how snapshots work: stores diff))*

Encryption

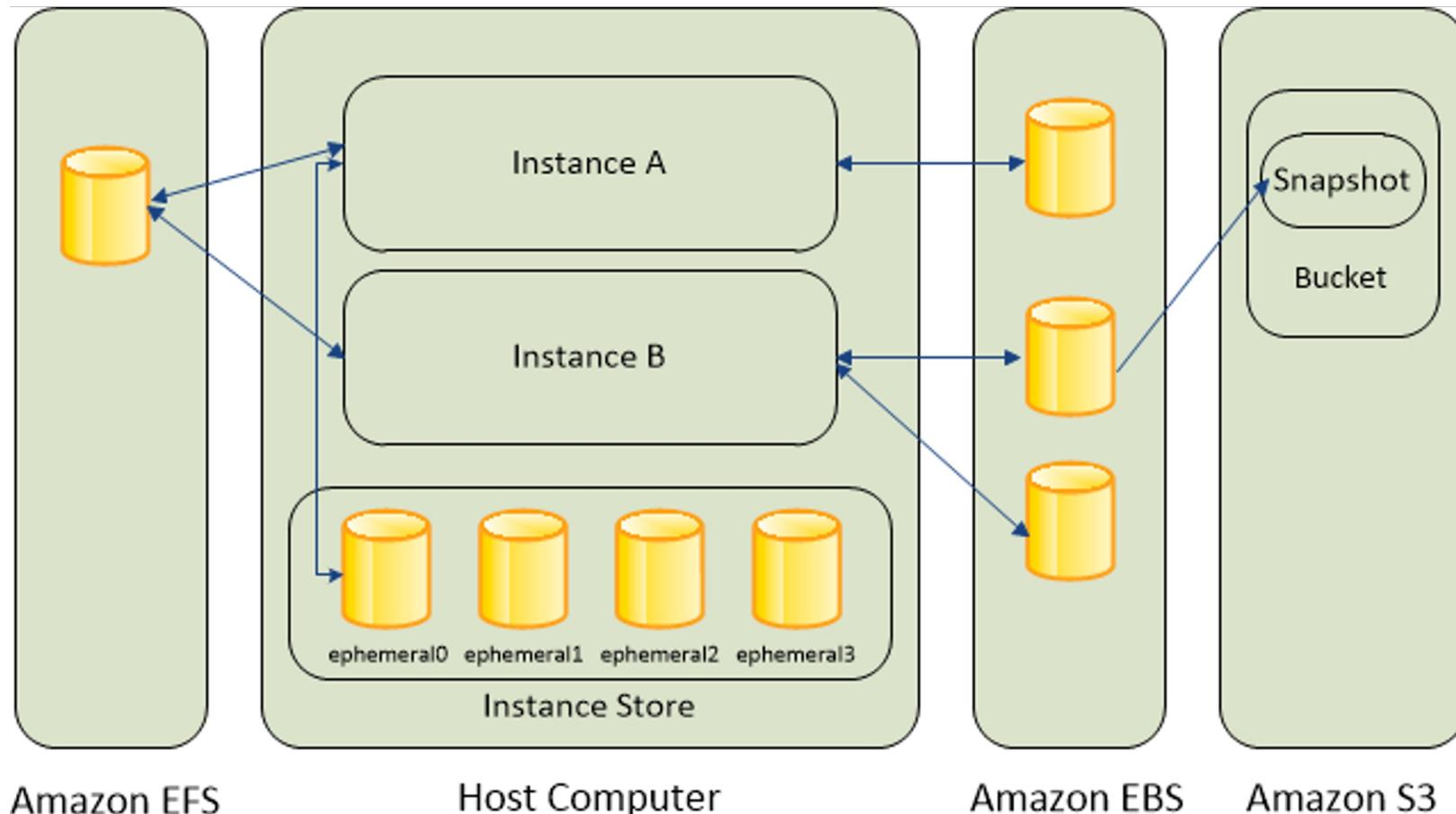
Amazon EBS encryption provides seamless support for data-at-rest and data-in-transit between EC2 instances and EBS volumes.

Data-at-rest Encryption: When your data is stored on the EBS volumes, it's encrypted. This means that even if someone gains access to the physical storage devices, they won't be able to read the data without the encryption key.

Data-in-transit Encryption: When data is moving between your EC2 instances and the EBS volumes (for example, during read/write operations), it's also encrypted. This ensures that data remains protected while it's being transferred over the network.

In summary, Amazon EBS encryption helps to ensure the security and confidentiality of your data both while it's stored and while it's being moved between your EC2 instances and EBS volumes.

Instance Store



Amazon EC2 Instance Store

No external network hop, (present in the same local network)
attached to VM , pricing is included in VM cost (storage + compute)

- An instance store provides temporary block-level storage for your instance.
- This storage is located on disks that are physically attached to the host computer.
- Instance store is ideal for temporary storage of information that changes frequently, such as buffers, caches, scratch data, and other temporary content, or for data that is replicated across a fleet of instances, such as a load-balanced pool of web servers.

Instance Store Use Cases

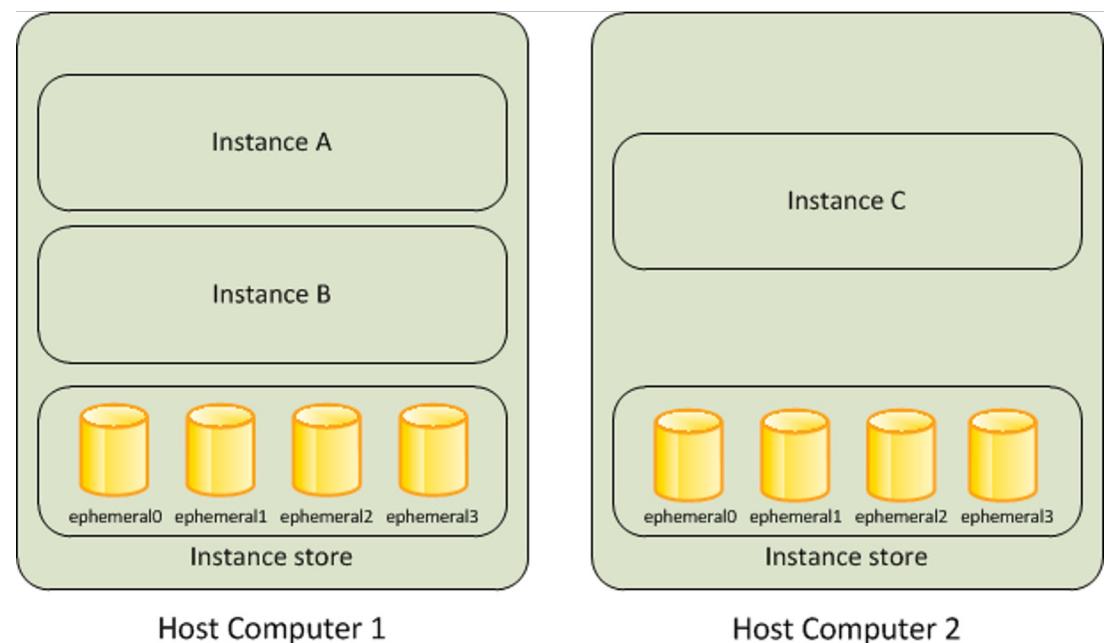
- Caching
- Temporary processing, or
- Applications that replicate data to several servers as some databases do

Instance Store Volumes

- An instance store consists of one or more instance store volumes exposed as block devices.
- The size of an Instance store varies by instance type.
- The virtual devices for instance store volumes are **ephemeral[0-23]**.
- Instance types that support one instance store volume have **ephemeral0**.
- Instance types that support two instance store volumes have **ephemeral0** and **ephemeral1**, and so on.

Root disk has OS (goes over local network)
No automated snapshots
No failover provided by cloud provider

* Diff btw Storage options - persistent & instance will come for midterm



Instance Store Lifetime

- The data in an instance store persists only during the lifetime of its associated instance. VM
- If an instance reboots (intentionally or unintentionally), data in the instance store persists.
- However, data in the instance store is lost under the following circumstances:
 1. The underlying disk drive fails
 2. The instance stops
 3. The instance terminates

Instance Store Cost

No additional charge for instance store as instance store is only available with certain instances and instance store charges are included in the EC2 instance price.

Encryption

- Instance store does not have out of box solution for encrypting data at rest.
- However there are ways to encrypt data using custom solution such as <https://aws.amazon.com/blogs/security/how-to-protect-data-at-rest-with-amazon-ec2-instance-store-encryption/>

You can use LUKS (Linux Unified Key Setup) in the file System to perform data encryption. Encryption is done using AWS KMS (Key management System) that internally uses encrypted password from AWS S3

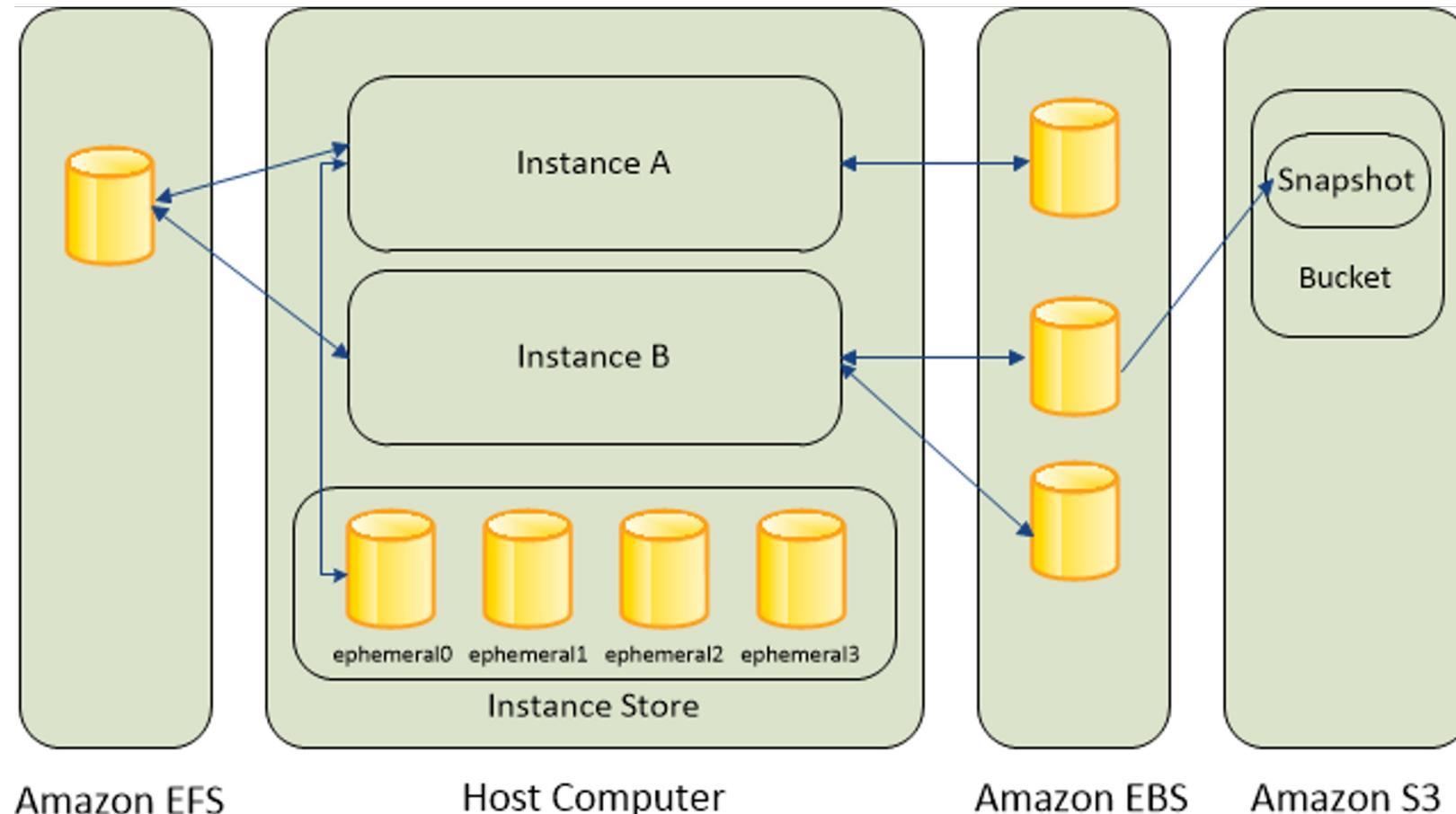
This encryption process occurs at the block level, ensuring that all files and directories within the file system are protected.

The Linux dm-crypt infrastructure is utilized to encrypt the file system transparently, ensuring that data written to disk is encrypted using an AES-256 encryption algorithm.

Final Thoughts on Instance Store

- Instance store is included in the virtual server and cannot exist without the virtual server unlike EBS.
- Don't use an instance store for data that must not be lost. temp storage
- Instance store helps avoid noisy neighbor problem as the EC2 servers do not have to go over network to access data.
- Instance store backed volume cannot be “stopped”. They can only be rebooted or terminated.

Amazon Elastic File System (Amazon EFS)



virtual storage

Amazon Elastic File System (Amazon EFS)

EFS-> disk automatically shrinks & grows in size (scalable), when created starts with 0B

- Amazon Elastic File System (Amazon EFS) provides simple, scalable file storage for use with Amazon EC2 instances in the AWS Cloud.
- It supports Network File System versions 4 (NFSv4) and 4.1 (NFSv4.1), which makes it easy to migrate enterprise applications to AWS or build new ones. nfs -> network file storage protocol (application layer protocol)
- It is also highly available and highly durable because it stores data and metadata across multiple Availability Zones in a Region. diff zones, same region

Accessing Amazon EFS

It is a secondary disk, not root (primary disk)
secondary cannot be used for holding OS software
Multiple secondary disks can be attached to a VM
Cannot access from Cloud provider console, must go into VM console to test it

- Amazon EFS must be mounted to Amazon EC2 instance. Once mounted, Amazon EFS provides a standard file system interface and file system access semantics.
- Multiple Amazon EC2 instances can access an Amazon EFS at the same time allowing EFS to provide a common data source for workloads and applications running on more than one Amazon EC2 instance.
- Amazon EFS is designed to meet the needs of multi-threaded applications and applications that concurrently access data from multiple EC2 instances and that require substantial levels of aggregate throughput and input/output operations per second (IOPS).

Scalability and Elasticity

- Amazon EFS automatically scales your file system storage capacity up or down as you add or remove files without disrupting your applications, giving you just the storage you need, when you need it, and while eliminating time-consuming administration tasks associated with traditional storage management (such as planning, buying, provisioning, and monitoring).
- Your EFS file system can grow from an empty file system to multiple petabytes automatically, and there is no provisioning, allocating, or administration.

Final Thoughts on EFS

- EFS cannot be used as root volume for your EC2 instance.
- Data in EFS is replicated across multiple Availability Zones (AZs).

EFS Use Case

- Big Data and analytics
- Media processing workflows
- Content management
- Web serving
- Home directories

Object Storage Service

Amazon S3

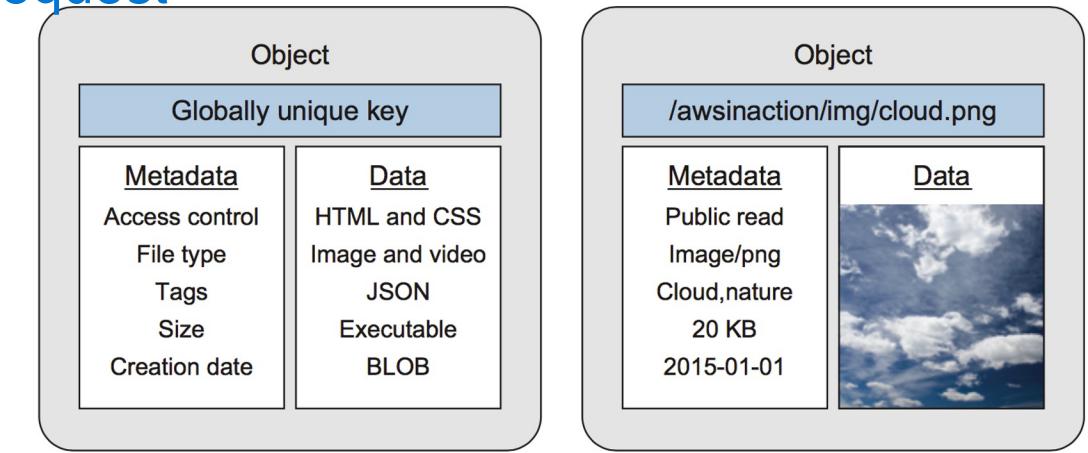
What is Object & Object Store?

- In an object store, data is stored as objects.
- Each object consists of a globally unique identifier, some metadata, and the data itself.
- The separation of metadata and data allows clients to work only with the metadata for managing and querying data.
- You only have to load the data if you really need it.
- Metadata is also used to store access-control information and for other management tasks.

how to access object store > foes via network
cannot control bandwidth

meta data & actual data : VM

Head http method : get only metada of the request



Amazon Simple Storage Service (S3)

- S3 is one of the oldest AWS service.
- S3 offers unlimited storage space.
- S3 is highly available and durable.
- An object in S3 can be as large as 5TB.
- S3 is accessed using RESTful APIs.

Amazon S3: Data retrieval by HTTP methods
(REST based)

Cannot be mounted on VM, need not have a
VM to have S3 buckets & objects

Reliability : File System > S3

S3 need not be a file, its an object: S3 can hold: .txt, .mp3, .vid, etc

Azure: Blob Storage

Amazon S3 Use Cases

Amazon S3 is an excellent choice for a large variety of use cases ranging from a simple storage repository for backup & recovery to primary storage for some of the most cutting-edge cloud-native applications in the market today and everything in between.

Data Consistency Model for S3

- All S3 GET, PUT, and LIST operations, as well as operations that change object tags, ACLs, or metadata, are **strongly consistent**.

Previous:

S3 had eventual consistency (were immutable, was updated with another object)
Versioning was not charged

Now:

S3 objects are immutable
Updates are made with new objects, older objects are deleted
We have to track versions -> more cost for versioning

S3 - Storage Tiers/Classes

latest / newest access -> more availability (hot content) ->
more num of requests -> highest storage tier

older access -> lesser availability (takes more time to load) ->
less num of requests -> lower storage tier

- **S3 (Standard)** - 99.99% availability, 99.99999999 durability, stored redundantly.
- **S3 - IA (Infrequently Accessed)** For data that is accessed less frequently but requires rapid access when needed. Costs less than S3 but you pay for retrieval.
- **Reduced Redundancy Storage** - 99.99% durability and 99.99% availability of objects over a given year.
- **Glacier** - Very cheap but used for archival only. It takes 3-5 hours to retrieve data from glacier.

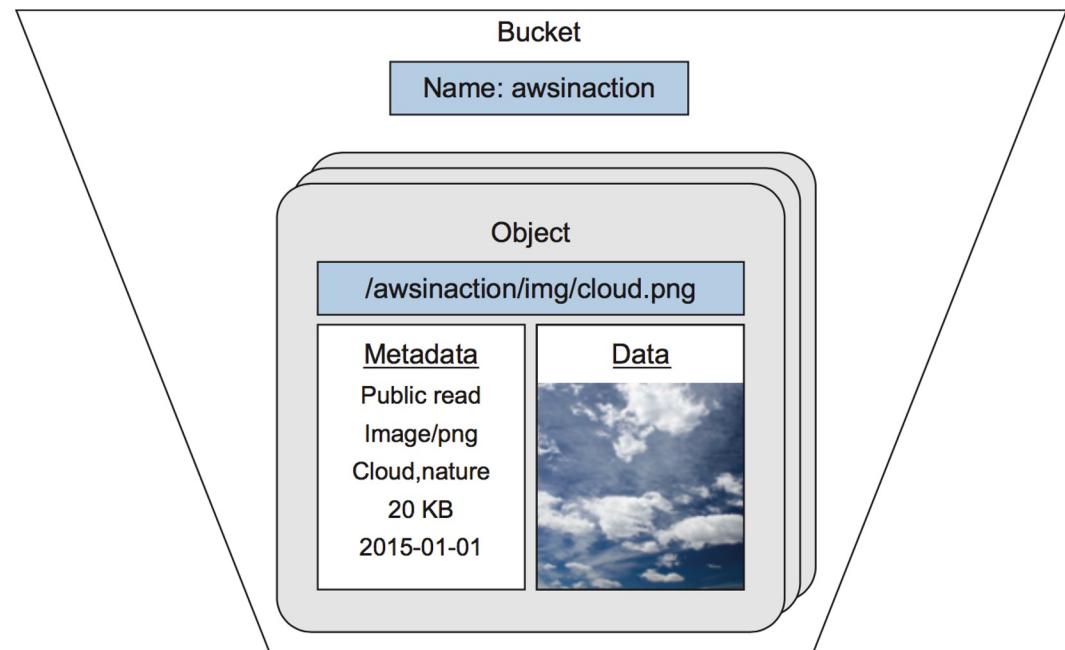
S3 - Storage Tiers/Classes

	Standard	Standard - IA	Amazon Glacier
Designed for Durability	99.99999999%	99.99999999%	99.99999999%
Designed for Availability	99.99%	99.9%	N/A
Availability SLA	99.9%	99%	N/A
Minimum Object Size	N/A	128KB*	N/A
Minimum Storage Duration	N/A	30 days	90 days
Retrieval Fee	N/A	per GB retrieved	per GB retrieved**
First Byte Latency	milliseconds	milliseconds	select minutes or hours***
Storage Class	object level	object level	object level
Lifecycle Transitions	yes	yes	yes

S3 Buckets

S3 is rest based, cannot use same endpoint across regions
All buckets have diff names across regions

- Every object you store in Amazon S3 resides in a bucket.
- Buckets may be used to group related objects in the same way that you use a directory to group files in a file system.
- Buckets have properties, such as access permissions and versioning status, and you can specify the region where you want them to reside.
- Buckets must have globally unique name.
- No other AWS customer in any other region can have same bucket name as yours.



Bucket Logging

- Logging provides a way to get detailed access logs delivered to a bucket you choose.
- An access log record contains details about the request, such as the request type, the resources specified in the request worked, and the time and date the request was processed.

Prod env: everything is logged, all data is preserved 1+ years

Since we want to know what happened: 500s, security incidents, etc.

Enable logging option for VM -> is chargeable

Bucket Event Notifications

Amazon S3 buckets can be configured to send a notification message to a destination whenever one of the following event occur:

1. An object created event
2. An object removed event
3. A Reduced Redundancy Storage (RRS) object lost event

Can store derivates of lower resolution from an asynchronous process (maybe through Kafka notifications)

Ex: Image or Video resolution

Why do we require diff resolutions, diff devices are connected to diff network, diff devices which are capable to load only certain versions

Bucket Versioning

- A versioning-enabled bucket can have multiple versions of objects in the bucket.
- By default, S3 versioning is disabled for every bucket.
- Suppose you use the following steps to upload two objects:
 - Add an object with key A and data 1.
 - Add an object with key A and data 2.
- If you download, also known as get, the object with key A, you'll download data 2. The old data 1 doesn't exist anymore.
- You can change this behavior by turning on versioning for a bucket.

Data Lifecycle Management

- Lifecycle configuration rules allow you to define actions you want Amazon S3 to take during an object's lifetime such as following:
 - Transition objects to another storage class
 - Archive objects
 - Delete objects after a specified period of time
- You can define a rule for all objects or a subset of objects in the bucket (by specifying the key name prefix). You can temporarily disable a rule.

Object Tagging

- S3 Object Tags are key-value pairs applied to S3 objects which can be created, updated or deleted at any time during the lifetime of the object.
- With these, you'll have the ability to create Identity and Access Management (IAM) policies, setup S3 Lifecycle policies, and customize storage metrics.
- These object-level tags can then manage transitions between storage classes and expire objects in the background.

Encryption

- SSE with Amazon S3 Key Management (SSE-S3)
 - With SSE-S3, Amazon S3 will encrypt your data at rest and manage the encryption keys for you.
- SSE with Customer-Provided Keys (SSE-C)
 - With SSE-C, Amazon S3 will encrypt your data at rest using the custom encryption keys that you provide.
- SSE with AWS KMS (SSE-KMS)
 - With SSE-KMS, Amazon S3 will encrypt your data at rest using keys that you manage in the AWS Key Management Service (KMS).

Cross-Region Replication

- Cross-region replication is the automatic, asynchronous copying of objects across buckets in different AWS regions.
- By activating cross-region replication, Amazon S3 will replicate newly created objects, object updates, and object deletions from a source bucket into a destination bucket in a different region.
- See <http://docs.aws.amazon.com/AmazonS3/latest/dev/crr-what-is-not-replicated.html> for limitations of Cross-Region Replication.

Querying S3

- Amazon S3 doesn't offer query capabilities to retrieve specific objects.
- When you use Amazon S3 you need to know the exact bucket name and key for the files you want to retrieve from the service.
- Amazon S3 can't be used as a database or search engine by itself.
- Instead, you can pair Amazon S3 with Amazon DynamoDB, Amazon CloudSearch, or Amazon Relational Database Service (Amazon RDS) to index and query metadata about Amazon S3 buckets and objects.

S3 is not suited for....

File System - Amazon S3 uses a flat namespace and isn't meant to serve as a standalone, POSIX-compliant file system. Instead, consider using Amazon EFS as a file system.

Relational Databases

Amazon Relational Database Service (RDS)

What is a relational database?

- Relational databases are what most of us are all used to. They have been around since late 70s.
- The term "relational database" was invented by E. F. Codd at IBM in 1970.
- An RDBMS at minimum
 - Presents the data to the user as relations (a presentation in tabular form, i.e. as a collection of tables with each table consisting of a set of rows and columns);
 - Provides relational operators to manipulate the data in tabular form.

Popular RDBMS

- Oracle
- MySQL
- Microsoft SQL Server
- PostgreSQL [Why do all startups prefer Postgres - cuz its cost efficient](#)
- IBM DB2
- Microsoft Access
- SQLite
- Amazon Aurora
- MariaDB

Transactions

- In order for a database management system (DBMS) to operate efficiently and accurately, it must support ACID transactions.
- *ACID* is short for *Atomicity, Consistency, Isolation, Durability*.

Atomicity

- Atomicity requires that each transaction be "all or nothing": if one part of the transaction fails, then the entire transaction fails, and the database state is left unchanged.
- An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes.
- To the outside world, a committed transaction appears (by its effects on the database) to be indivisible ("atomic"), and an aborted transaction does not happen.

Consistency

The consistency property ensures that any transaction will bring the database from one valid state to another.

Isolation

The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed sequentially, i.e., one after the other.

Varies by diff storage engines:

Isolations:

1. Lock on db
2. Lock on Table
3. Lock on row

More granular operations -> more durability -> more parallel computations

Durability

- The durability property ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.
- In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter).

write ahead logs happens asynchronously this ensures durability

Back up write ahead logs with back of DB
when DB crashes, on startup it first reloads & checks consistency with current DB state

Automated Backups of AWS RDS

There are two different types of backups for AWS.

1. Automated Backups
2. Database Snapshots

[write ahead logs can provide point in time recovery](#)

[optional toggle feature -> chargeable](#)

Advantages of AWS RDS

- Easy to Administer
- Highly Scalable
- Available and Durable
- Fast
- Secure

Automated Backups

- Recover database to any point in time within the retention period.
- Automated backups take a full daily snapshot and will also store transaction logs throughout the day.
- Automated Backups are enabled by default.
- Backup is stored in S3 but it is free.
- You do not pay for S3.
- During backup window, you might experience higher latency as storage I/O may be suspended.
- Automated backups are deleted when you terminate your RDS instance.

Database Snapshots

- Snapshots are done manually.
- They are stored even after you delete the original RDS instance.

Restoring Backups

Whenever you restore either an **Automatic Backup** or a manual **Database Snapshot** backup, the restored version of the database will be a new RDS instance with a new end point.

Upgrading dbs:

Standby is promoted to active 1

Active is promoted to standby 2 -> then 2 is upgraded

After that 1 is upgraded

2 becomes active, 1 becomes standby

Encryption at Rest and in Transit

- Amazon RDS allows you to encrypt your databases using keys you manage through AWS Key Management Service (KMS).
- On a database instance running with Amazon RDS encryption, data stored at rest in the underlying storage is encrypted, as are its automated backups, read replicas, and snapshots.
- Amazon RDS supports the use of SSL to secure data in transit.

Amazon RDS Multi-AZ Deployments

- Amazon RDS Multi-AZ deployments provide enhanced availability and durability for Database (DB) Instances.
- When you provision a Multi-AZ DB Instance, Amazon RDS automatically creates a primary DB Instance and synchronously replicates the data to a standby instance in a different Availability Zone (AZ).
- Each AZ runs on its own physically distinct, independent infrastructure, and is engineered to be highly reliable.
- In case of an infrastructure failure Amazon RDS performs an automatic failover to the standby, so that you can resume database operations as soon as the failover is complete.
- Since the endpoint for your DB Instance remains the same after a failover, application can resume database operation without the need for manual administrative intervention.

Amazon RDS Read Replicas

only 1 instance has write capability
rest are read only replicas

read replicas will always have lag,
it is eventual consistency

- Amazon RDS Read Replicas provide enhanced performance and durability for database (DB) instances.
- This replication feature makes it easy to elastically scale out beyond the capacity constraints of a single DB Instance for read-heavy database workloads.
- You can create one or more replicas of a given source DB Instance and serve high-volume application read traffic from multiple copies of your data, thereby increasing aggregate read throughput.
- Read replicas can also be promoted when needed to become standalone DB instances.

NoSQL Databases

Value of Relation Database

- Store large amounts of persistent data
- Concurrency
- Standard Model (Relational Algebra & SQL)
- ACID

Not so Good side of RDBMS

- Impedance mismatch - The difference between relational model and in-memory data structure.
- Mediocre horizontal scaling support

What is NoSQL?

- The term NoSQL may refer to "non SQL", "non relational" or "not only SQL".
- A NoSQL database provides a mechanism for storage and retrieval of data which is modeled in means other than the tabular relations used in relational databases.

Why is it called "NoSQL"?

- Johan Oskarsson, then a developer at Last.fm, wanted a name for meetup in 2009.
- Something that would make a good hashtag: short, memorable, and without too many Google hits so that search of the term would quickly find the meetup.
- The name attempted to label the emergence of an increasing number of non-relational, distributed data stores.

Characteristics of NoSQL Database

- Schema-less (most of them, not all)
- Does not use relational model (most of them, not all)
- Runs well on cluster

NoSQL is more than rows

NoSQL systems store and retrieve data from many formats such as

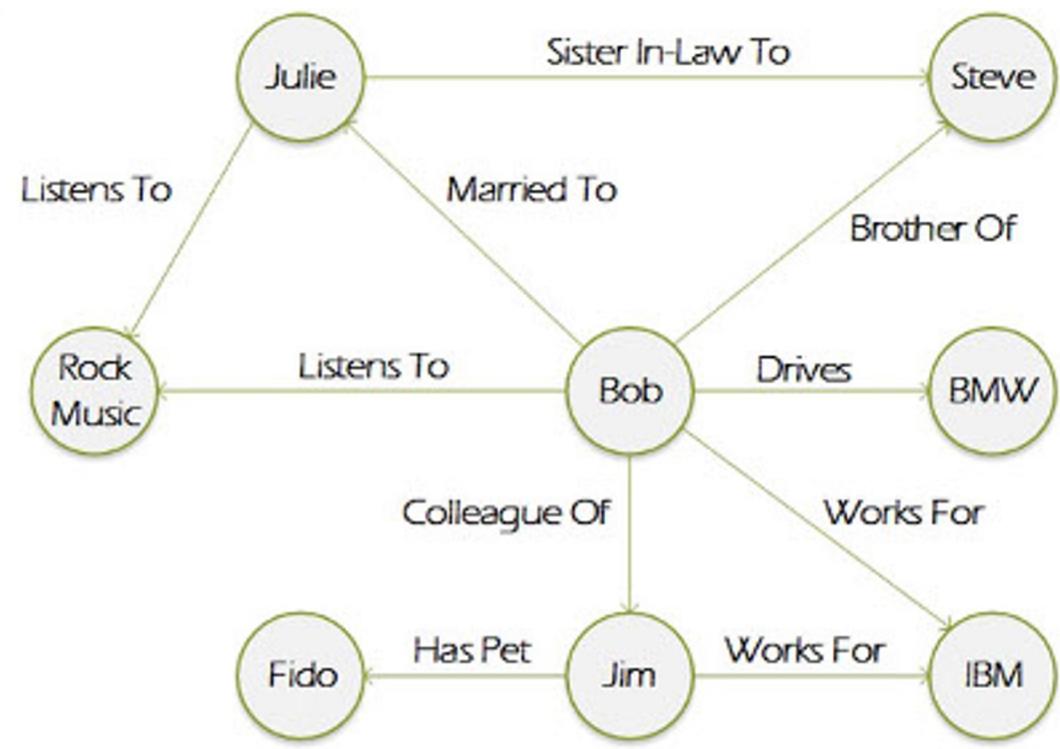
- Key-Value (DynamoDB)
- Graph
- Column-family (Cassandra)
- Document (MongoDB)

What is key-value store?

- A key-value store is a simple database that when presented with a simple string (the key) returns an arbitrary large BLOB of data (the value).
- Key-value stores usually have no query language.
- In a key-value store, key is of String data type and value can be of any type.
- One does not have to specify the type for value.
- Example: DynamoDB & Redis

Graph Databases

- This kind of database is designed for data whose relations are well represented as a graph consisting of elements interconnected with a finite number of relations between them.
- The type of data could be social relations, public transport links, road maps or network topologies.



Column Family (Bigtable) Stores

- Bigtable maps two arbitrary string values (row key and column key) and timestamp (hence three-dimensional mapping) into an associated arbitrary byte array.
- Think of spreadsheets. ([don't](#))
- You can use combination of row number and column letter as an address to “look up” the value of any cell.
- Examples: Google Bigtable & Apache Cassandra

Document Database

- Document Databases differ from other NoSQL databases we have looked at in the sense that they Key may be simple id which is never used or seen but you can get almost any item out of document store by querying any value or content within the document.
- The central concept of a document-oriented database is the notion of a document.
- While each document-oriented database implementation differs on the details of this definition, in general, they all assume documents encapsulate and encode data (or information) in some standard formats or encodings such as JSON, BSON, XML, etc.

NoSQL is free of Joins

NoSQL systems allow you to extract your data using simple interfaces. It does not support “joins”.

NoSQL databases are Schema Free

- A NoSQL system can handle data in various formats.
- You do not have to tell NoSQL system in advance about the format of data.

Scaling NoSQL

- NoSQL systems scale linearly.
- More processors you add, more performant the system gets.

CAP Theorem

(in presence of a network connection)

- The CAP theorem states that it is impossible for a distributed computer system to simultaneously provide more than two out of three of the following guarantees:
 - Consistency - Every read receives the most recent write or an error
 - Availability - Every request receives a (non-error) response – without guarantee that it contains the most recent write *ex: social media applications, e-commerce display*
 - Partition tolerance - The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes *Split brain problem ?*
- In other words, the CAP Theorem states that in the presence of a network partition, one has to choose between consistency and availability.
- Note that consistency as defined in the CAP Theorem is quite different from the consistency guaranteed in ACID database transactions.

Transactions in NoSQL

CAP Most NoSQL stores lack true ACID transactions.

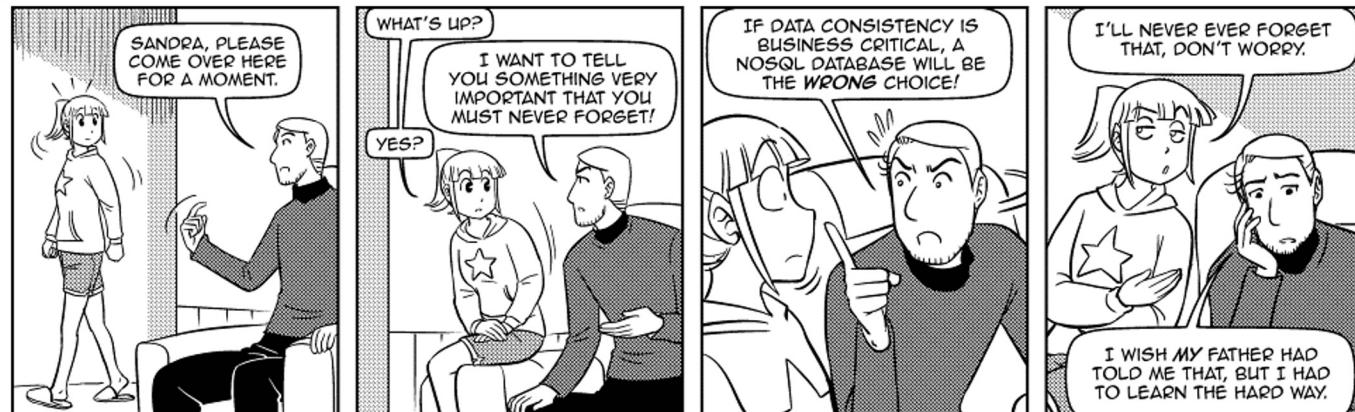
1. ****Consistency****: Refers to all nodes in a distributed system having the same data at the same time, even in the presence of concurrent updates. In the CAP theorem, consistency means that all nodes see the same version of data at the same time, regardless of which node they query.
2. ****Partition tolerance****: Refers to the system's ability to continue operating despite communication breakdowns (partitions) between nodes. Partition tolerance acknowledges that network partitions can occur, and the system should still be able to function and serve requests.
3. ****Availability****: Refers to the system's ability to continue operating and respond to requests, even in the presence of failures. Availability ensures that every request receives a response, whether it's successful or not.

ACID

1. ****Atomicity****: Ensures that a transaction is treated as a single unit of work, either all of its operations are completed successfully, or none of them are. There is no halfway point; transactions are atomic and indivisible.
2. ****Consistency****: In the context of ACID, consistency refers to the database being in a valid state before and after the transaction. It ensures that all constraints, triggers, and relationships are maintained, preserving data integrity.
3. ****Isolation****: Ensures that the intermediate state of a transaction is invisible to other transactions until it's committed. Isolation prevents interference or conflicts between concurrent transactions, maintaining data integrity and correctness.
4. ****Durability****: Guarantees that once a transaction is committed, its changes are permanent and will not be lost, even in the event of system failures or crashes. Durability ensures that committed transactions survive system failures and are persisted to durable storage.

NoSQL Databases & Consistency

NoSQL databases offer a concept of "eventual consistency" in which database changes are propagated to all nodes "eventually" (typically within milliseconds) so queries for data might not return updated data immediately or might result in reading data that is not accurate, a problem known as stale reads.



Sandra and Woo by Oliver Knörzer (writer) and Powree (artist) – www.sandraandwoo.com

Normalization & Denormalization

- **Database normalization** is a technique for designing relational database schemas that ensures that the data is optimal for ad-hoc querying and that modifications such as deletion or insertion of data does not lead to data inconsistency.
- **Database denormalization** is the process of optimizing your database for reads by creating redundant data. A consequence of denormalization is that insertions or deletions could cause data inconsistency if not uniformly applied to all redundant copies of the data within the database.

Polyglot Persistence

- Polyglot Persistence, like polyglot programming, is all about choosing the right persistence option for the task at hand.
- NoSQL has resulted in widespread use of Polyglot Persistence.

"Polyglot persistence" refers to the practice of using multiple types of data storage technologies (often referred to as databases) within a single application or system. This approach recognizes that different types of data may be best suited for different storage technologies, and thus employs a variety of databases to efficiently manage and retrieve data

Concurrency Control Mechanisms

- **Optimistic** - Delay the checking of whether a transaction meets the isolation and other integrity rules (e.g., serializability and recoverability) until its end, without blocking any of its (read, write) operations ("...and be optimistic about the rules being met..."), and then abort a transaction to prevent the violation, if the desired rules are to be violated upon its commit. An aborted transaction is immediately restarted and re-executed, which incurs an obvious overhead (versus executing it to the end only once). If not too many transactions are aborted, then being optimistic is usually a good strategy.
- **Pessimistic** - Block an operation of a transaction, if it may cause violation of the rules, until the possibility of violation disappears. Blocking operations is typically involved with performance reduction.

Write-Write Conflict

- Two users updating same data item with different values.
- Optimistic Approaches:
 - Conditional Update: See if value has changed since last read before update
 - Process both updates and record that there is a conflict

Read-Write Conflict

- Alice and Bob are using a website to book tickets for a specific show. Only one ticket is left for the specific show. Alice signs on first to see that only one ticket is left, and finds it expensive. Alice takes time to decide. Bob signs on and also finds one ticket left, and orders it instantly. Bob purchases and logs off. Alice decides to buy a ticket, to find there are no tickets. This is a typical read-write conflict situation.

Version Stamps

- A field that changes every time the underlying data in the record changes.
- When you read the data you keep note of the version stamp, so that when you write data you can check to see if the version has changed.

Postgres - mvcc, you have records that have been updated , vacuum / garbage collector you can run out of free transaction ids, more you vacuum, more harder to manage/expensive

MySQL - updates 1 copy of data

Single Server (Distribution Model)

- One machine handles all reads and writes to the database. only on dev, not on production
- Simplest distribution model as there is no cluster.
- Suitable for development setup.

Sharding (Distribution Model)

- Store different parts of data on different servers.
- Spread the load across the cluster as different users ideally will be accessing data from different nodes.
- Sharding can improve both read and write performance.
- Note: If the data is sharded using wrong key, you can end up with highly unbalanced cluster and all traffic will end up going to same server nodes.

Primary/Secondary Nodes Replication (Distribution Model)

- In this distribution model you replicate data across multiple nodes.
- One node is designated as “primary” and others are designated as “secondary.”
- This “primary” node is authoritative source of data and is usually responsible for processing any updates to that data.
- primary node can be a bottleneck as all writes must go to it.
- If primary node dies, secondary node can still answer read queries while another node is elected primary either automatically or manually.
- primary node can be single point of failure.

Peer-to-Peer Replication (Distribution Model)

- There is no master node in peer-to-peer replication.
- All replicas have equal weight, they can all accept writes, and loss of any of them doesn't prevent access to data store.

Combining Sharding & Replication (Distribution Model)

Peer-to-peer replication combined with sharding is common strategy used by column-family databases.

**3 DATABASE ADMINS
WALKED INTO
A NOSQL BAR...**

**A LITTLE WHILE LATER
THEY WALKED OUT BECAUSE
THEY COULDN'T FIND A TABLE**

Additional Resources

See Lecture Page

Email Service

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University

The need for email service

- alerting

Alerts can not only need to be a communication to end users, it can be:

Alert for other internal service

Alerts are stored for legal & compliance purposes

New compliance for email: ensure to have predefined structure, headers, company links, unsubscribe and other validations so that, new mail services - Google & Yahoo do not mark it as spam

- **SEND TRANSACTIONAL MESSAGES** - Keep your customers up-to-date by sending automated emails, such as purchase confirmations, shipping notifications, order status updates, and policy change notices.
- **SEND NOTIFICATIONS** - Keep your users informed by sending timely information, including system health reports, application alerts, and workflow status updates.
- **RECEIVE INCOMING EMAIL** - Close the loop on your email program to receive email

Understand Email Delivery Issues



Understand Email Delivery Issues - Bounce

- In most cases, your messages are delivered successfully to recipients who expect them. In some cases, however, a delivery might fail, or a recipient might not want to receive the mail that you are sending. Bounces, complaints, and the suppression list are related to these delivery issues.
- **Bounce** - If your recipient's receiver (for example, an ISP) fails to deliver your message to the recipient, the receiver bounces the message back to Amazon SES. Amazon SES then notifies you of the bounced email through email or through Amazon Simple Notification Service (Amazon SNS), depending on how you have your system set up. There are hard bounces and soft bounces, as follows:
 - **Hard bounce** – A persistent email delivery failure. For example, the mailbox does not exist. Amazon SES does not retry hard bounces, with the exception of DNS lookup failures. You should not make repeated delivery attempts to email addresses that hard bounce.
 - **Soft bounce** – A temporary email delivery failure. For example, the mailbox is full, there are too many connections (also called throttling), or the connection times out. Amazon SES retries soft bounces multiple times. If the email still cannot be delivered, then Amazon SES stops retrying it.
- Bounces can also be synchronous or asynchronous.
 - A **synchronous** bounce occurs while the email servers of the sender and receiver are actively communicating.
 - An **asynchronous** bounce occurs when a receiver initially accepts an email message for delivery and then subsequently fails to deliver it to the recipient.

Soft bounce - retry for 72 hours automatically by the server -> then it becomes a hard bounce

Hard bounce -> no retry (complete failure)

Understand Email Delivery Issues - Complaint

Spam emails can be silently blocked by mail servers

- Most email client programs provide a button labeled "Mark as Spam," or similar, which moves the message to a spam folder, and forwards it to the ISP.
- Additionally, most ISPs maintain an abuse address (e.g., abuse@example.net), where users can forward unwanted email messages and request that the ISP take action to prevent them.
- In both of these cases, the recipient is making a complaint. If the ISP concludes that you are a spammer, and Amazon SES has a feedback loop set up with the ISP, then the ISP will send the complaint back to Amazon SES.
- When Amazon SES receives such a complaint, it forwards the complaint to you either by email or by using an Amazon SNS notification, depending on how you have your system set up.
- It is not recommended to make repeated delivery attempts to email addresses that generate complaints.

Understand Email Delivery Issues - Suppression List

manual removal, auto clears after a few days / years

- The suppression list is a list of recipient email addresses that have recently caused a hard bounce for any Amazon SES customer.
- If you try to send an to an address that is on the suppression list, the call may succeed, but the email may be treated as a hard bounce instead of attempting to send it.
- Like any hard bounce, suppression list bounces count towards your sending quota and your bounce rate.
- An email address can remain on the suppression list for up to 14 days.
- If you are sure that the email address that you're trying to send to is valid, you can submit a suppression list removal request.

Be Proactive

- One of the biggest issues with email on the Internet is unsolicited bulk email, or spam.
- ISPs take considerable measures to prevent their customers from receiving spam.

Best practices:

Don't send spam

Address the complaints

Keep checking ip address & domain

build a reputation for emails: > 99% & monitor it

Be Proactive - Verification

- Unfortunately, it's possible for a spammer to falsify an email header and spoof the originating email address so that it appears as though the email originated from a different source.
- To maintain trust between ISPs and email service, email service needs to ensure that its senders are who they say they are.
- You can also verify entire domains.

Be Proactive - Authentication

- Authentication is another way that you can indicate to ISPs that you are who you say you are.
- When you authenticate an email, you provide evidence that you are the owner of the account and that your emails have not been modified in transit.
- In some cases, ISPs refuse to forward email that is not authenticated.
- Amazon SES supports two methods of authentication
 - Sender Policy Framework (SPF)
 - DomainKeys Identified Mail (DKIM)

Sender Policy Framework (SPF)

- An SPF record indicates to ISPs that you have authorized a service to send mail for your domain.

SPF - deprecated, but still in use, authorizes whether a domain can send emails

First SPF record, then DKIM record -> DNS receive this, checks the sign & then checks the signature

DomainKeys Identified Mail (DKIM)

- DomainKeys Identified Mail (DKIM) is a standard that allows senders to sign their email messages with a cryptographic key.
- Email providers then use these signatures to verify that the messages weren't modified by a third party while in transit.
- An email message that is sent using DKIM includes a DKIM-Signature header field that contains a cryptographically signed representation of the message.
- A provider that receives the message can use a public key, which is published in the sender's DNS record, to decode the signature.
- Email providers then use this information to determine whether messages are authentic.
- To learn more about DKIM, see <http://dkim.org>.

Be Proactive - Reputation

- When it comes to email sending, reputation—a measure of confidence that an IP address, email address, or sending domain is not the source of spam—is important.
- Amazon SES maintains a strong reputation with ISPs so that ISPs deliver your emails to your recipients' inboxes.
- Similarly, you need to maintain a trusted reputation with Amazon SES. You build your reputation with Amazon SES by sending high-quality content.
- When you send high-quality content, your reputation becomes more trusted over time and Amazon SES increases your sending limits.
- Excessive bounces and complaints negatively impact your reputation and can cause Amazon SES to lower your sending limits or terminate your Amazon SES account.

Additional Resources

See Lecture Page

Domain Name System (DNS)

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University

Why is DNS used:

IP lookup

Block external service lookup to block ads

Resolving sites for internal services or intranet (intranet is private -> so needs a private dns)
for better performance & have your own logic for lookups - Cache additional ip addresses

What is Domain Name System (DNS)?

- DNS, or the Domain Name System is the networking system in place that allows us to resolve human-friendly names to unique IP addresses.
- DNS can be thought of as the phonebook of the Internet.
 - Humans access information online through domain names, like nytimes.com or espn.com.
 - Web browsers interact through Internet Protocol (IP) addresses.
 - DNS translates domain names to IP addresses so browsers can load Internet resources.
- DNS is a globally distributed, stateless, scalable, reliable database.

Why DNS?

- Human-friendly
 - Is this easy to remember - <https://216.58.219.196> ?
 - How about this - <https://www.google.com> ?
- De-centralized Administration

Anyone can create a dns server except for residential ip addresses, i.e. use cloud provider or company registered internal dns server: ex: northeastern has its own dns servers

Global Distribution

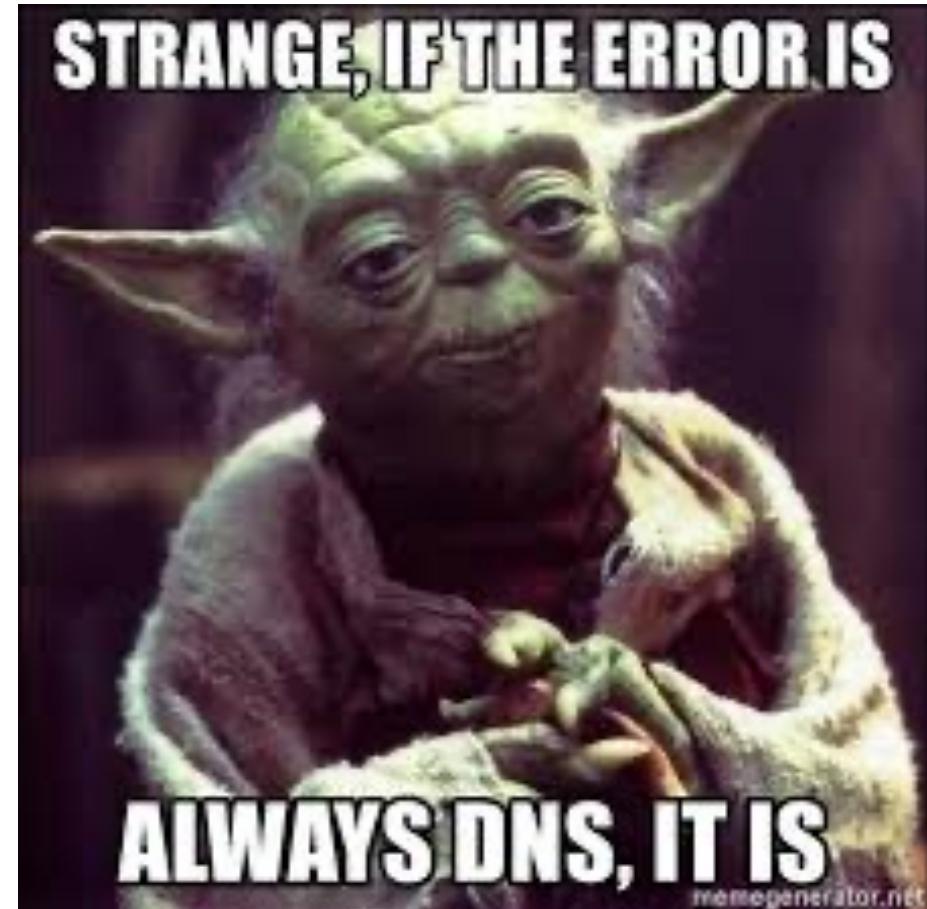
- Data is maintained locally, but retrievable globally
 - No single server has all the DNS data
- Remote DNS data is locally cacheable to improve performance

Globally distributed - there is no master dns server that gives the correct translation, sometimes some servers can store stale records (based on zone ttl) - there is no broadcast that transmits a value has expired

Loose Coherency

- Each version of a subnet of the database (a zone) has a serial number which is incremented on each database change
- Changes to the master copy of the database are propagated to replicas
- Cached data expires according to the timeout set (TTL)

Advice/best practice: drop the ttl to down to the minute (inc load & cost) - no ensure cache is not maintained



Scalability

- No limit on the size of the database
- No limit to the number of queries
- Queries distributed among primary, secondary, and caches

Reliability

- Data is replicated
- Client can query master or any of the secondary servers
- Client will typically query local caches

Dynamicity

- Database on master can be updated dynamically
- Modification of the master database triggers replication

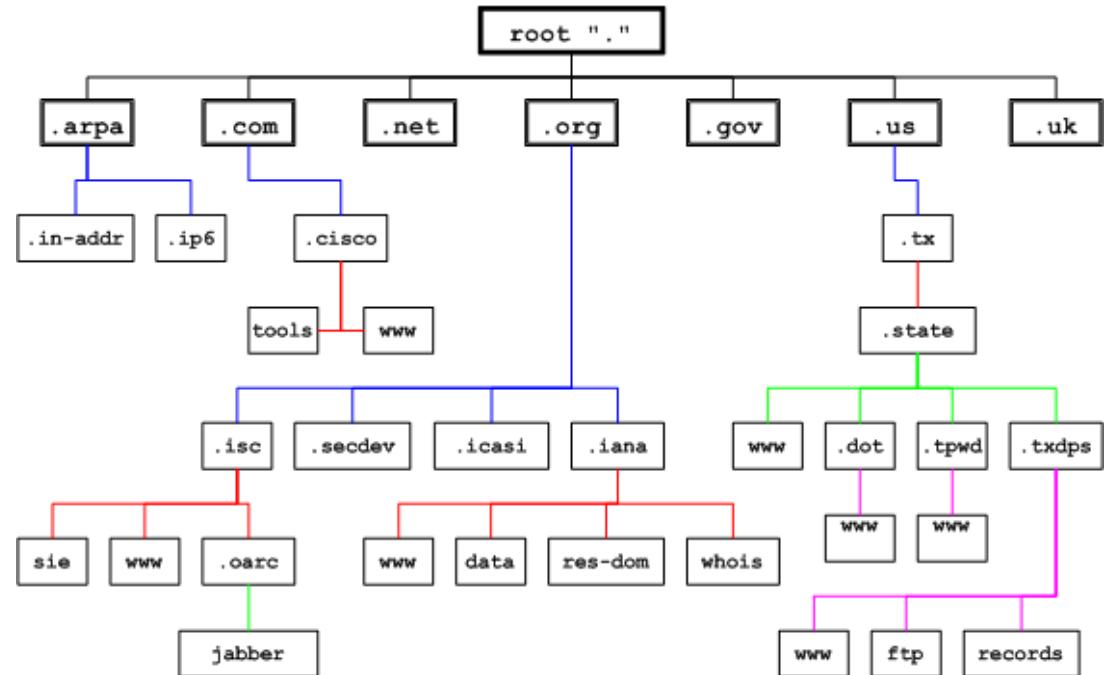
DNS Components

DNS is comprised of three components

- Name space (TLDs)
- Name Servers
- Resolvers

The Name Space

- The **name space** is the structure of the DNS database
 - An inverted tree with the root node at the top
- Each node has a label
- The root node has a **null** label written as "."



.arpa: primarily used for address to host mappings

.com, .net, .org, .org: are generic TLDs (gTLD)

.us, .uk: are country code TLDs (ccTLD)

Labels

- Each node in the tree must have label
- A label is a string of up to 63 bytes
- RFCs 852 and 1123 define legal character for hostname
- A-Z, 0-9 and "-" are the only valid characters. A-Z and a-z are treated the same. Hostnames are case-insensitive
- Sibling node must have unique label
- The **null** label is reserved for the root node

Domain Names

- A **domain name** is the sequence of labels from a node to the root, separated by dots ("."s) read left to right
- Domain names are limited to 255 characters in length
- Top-Level Domains (TLDs)
 - gTLDs: Generic top-level domain
 - ccTLDs: Country code top-level domain
- N-level domains - The name space has a maximum depth of 127 levels
- Subdomains



"It's our first. Don't know where to begin.
Haven't even picked out a domain name."

Top-Level Domain

- A top-level domain, or TLD, is the most general part of the domain.
- The top-level domain is the furthest portion to the right (as separated by a dot).
- Common top-level domains are “com”, “net”, “org”, “gov”, “edu”, and “io”.
- Top-level domains are at the top of the hierarchy in terms of domain names.
 - .com is not a tld but sub level domain, the single node at the top is “.”
 - only node at the top that has null name
 - Browsers always adds a “.” for every request, some services may not
 - Ex: google.com.
 - Tld must have unique labels, ex: only one northeastern.edu for .edu tld
 - cctlds : 2 characters - ex: us, in, uk
 - gtlds: .ai -> very expensive

Country code top-level domain

- A country code top-level domain (ccTLD) is an Internet top-level domain generally used or reserved for a country, sovereign state, or dependent territory identified with a country code.
- All ASCII ccTLD identifiers are two letters long, and all two-letter top-level domains are ccTLDs.
- There are 312 ccTLDs in active use totally.
 - The .cn, .tk, .de and .uk ccTLDs contain the highest number of domains.

Subdomains

- DNS works in a hierarchy.
- TLDs can have many domains under them.
 - For instance, the “com” TLD has both “google.com” and “ubuntu.com” underneath it.
- A **subdomain** is a domain that is part of a larger domain; the only domain that is not also a subdomain is the root domain.
 - For example, **west.example.com** and **east.example.com** are subdomains of the **example.com** domain, which in turn is a subdomain of the **com** top-level domain (TLD).

Subdomains are carved out for every sub-institution for their own mail servers (say marketing, separate IT team management, zone management etc.)
Ex: coe, cos, cps are have a separate subdomain

Fully Qualified Domain Name

Fully Qualified Domain name, proper fully qualified domain name: has root at the end “.”

- A fully qualified domain name, often called FQDN, is what we call an absolute domain name.
- Domains in the DNS system can be given relative to one another, and as such, can be somewhat ambiguous.
- A FQDN is an absolute name that specifies its location in relation to the absolute root of the domain name system.
- A proper FQDN ends with a dot, indicating the root of the DNS hierarchy. An example of a FQDN is **mail.google.com**.

Name Servers

- Run the software (BIND, BIND 9, NSD) which receive and respond to DNS queries
- Name servers store information about the name space in units called "zone"
- Usually, more than one name server are authoritative for the same zone ensuring redundancy and load balancing
- A single name server may be authoritative for many zones

Why should you use custom name servers??

Ns can have 2 diff dns servers

Can set up your own dns using pi whole using raspberry pi

Types of Name Servers

- Two main types of name servers
 - **Authoritative** – maintains the data
 - Primary – where the data is edited
 - Secondary – where the data is replicated to
 - **Caching** – stores data obtained from authoritative server
- No special hardware is needed to run name servers

Zones

- The DNS is broken up into many different zones.
- These zones differentiate between distinctly managed areas in the DNS namespace.
- A DNS zone is a portion of the DNS namespace that is managed by a specific organization or administrator.
- A DNS zone is an administrative space which allows for more granular control of DNS components, such as authoritative nameservers.
- The domain name space is a hierarchical tree, with the DNS root domain at the top.
- A DNS zone starts at a domain within the tree and can also extend down into subdomains so that multiple subdomains can be managed by one entity.

Zone File Example

```
; SOA Record
example.com. 3600 IN SOA ns69.domaincontrol.com. dns.jomax.net (
2016122100
28800
7200
604800
600
)

; A Records
@ 600 IN A 154.45.18.26

; CNAME Records
www 3600 IN CNAME @
email 3600 IN CNAME email.secureserver.net

; MX Records
@ 3600 IN MX 10 mailstore1.secureserver.net
@ 3600 IN MX 0 smtp.secureserver.net

; TXT Records
@ 3600 IN TXT "site-verification-1213fasd12312414asda"

; NS Records
@ 3600 IN NS ns69.domaincontrol.com
@ 3600 IN NS ns70.domaincontrol.com
```

DNS Resource Record Types

Ns records cannot be changed
Best practice soa record should have lower ttl, change the default negative cache record for something that works??

- **A** - The value for an A record is an IPv4 address in dotted decimal notation
- **AAAA** - The value for a AAAA record is an IPv6 address in colon-separated hexadecimal format
- **CAA** - A CAA record lets you specify which certificate authorities (CAs) are allowed to issue certificates for a domain or subdomain.
- **CNAME** - A CNAME Value element is the same format as a domain name
- **MX** - A mail exchanger record (MX record) is a type of resource record in the Domain Name System that specifies a mail server responsible for accepting email messages on behalf of a recipient's domain
- **NS** - An NS record identifies the name servers for the hosted zone
- **SOA** - A start of authority (SOA) record provides information about a domain and the corresponding hosted zone
- **SPF** - SPF records were formerly used to verify the identity of the sender of email messages.
- **TXT** - A TXT record contains a space-separated list of double-quoted strings
- and many more types

SOA record

- The DNS "start of authority" (SOA) record stores important information about a domain or zone such as the email address of the administrator, when the domain was last updated, and how long the server should wait between refreshes.

```
domain.com. IN SOA ns1.domain.com. admin.domain.com. (
    12083 ; serial number
    3h    ; refresh interval
    30m   ; retry interval
    3w    ; expiry period
    1h    ; negative TTL
)
```

A Record

- The ‘A’ stands for ‘address’ and this is the most fundamental type of DNS record: it indicates the IP address of a given domain.
- A records hold IPv4 addresses.
- The most common usage of A records is IP address lookups: matching a domain name to an IPv4 address.

AAAA Record

- AAAA records hold IPv6 addresses.

CNAME Record

- The ‘canonical name’ (CNAME) record is used in lieu of an A record, when a domain or subdomain is an alias of another domain.
- All CNAME records must point to a domain, never to an IP address.

MX Record

Example of an MX record:

example.com	record type:	priority:	value:	TTL
@	MX	10	mailhost1.example.com	45000
@	MX	20	mailhost2.example.com	45000

- A DNS 'mail exchange' (MX) record directs email to a mail server.
- The MX record indicates how email messages should be routed in accordance with the Simple Mail Transfer Protocol (SMTP, the standard protocol for all email).
- Like CNAME records, an MX record must always point to another domain.
- The 'priority' numbers before the domains for these MX records indicate preference; the lower 'priority' value is preferred.
 - The server will always try mailhost1 first because 10 is lower than 20.
 - In the result of a message send failure, the server will default to mailhost2.

TXT Record

- The DNS ‘text’ (TXT) record lets a domain administrator enter text into the Domain Name System (DNS).
- One domain can have many TXT records.
- Today, two of the most important uses for DNS TXT records are email spam prevention and domain ownership verification, although TXT records were not designed for these uses originally.

NS Record

- NS stands for ‘nameserver,’ and the nameserver record indicates which DNS server is authoritative for that domain (i.e. which server contains the actual DNS records).
- NS records tell the Internet where to go to find out a domain's IP address.
- A domain often has multiple NS records which can indicate primary and backup nameservers for that domain.
- Without properly configured NS records, users will be unable to load a website or application.

CAA Record

- This is the ‘certification authority authorization’ record, it allows domain owners state which certificate authorities can issue certificates for that domain.
- If no CAA record exists, then anyone can issue a certificate for the domain.
- These records are also inherited by subdomains.

TTL (Time to Live)

- The amount of time, in seconds, that you want DNS recursive resolvers to cache information about this resource record set.
- If you specify a longer value (for example, 172800 seconds, or two days) it often takes longer for changes to the resource record set (for example, a new IP address) to take effect because recursive resolvers use the values in their cache for longer periods instead of querying.

*I'll tell you a DNS joke but
be advised, it could take up
to 24 hours for everyone to
get it.*

Name Resolution

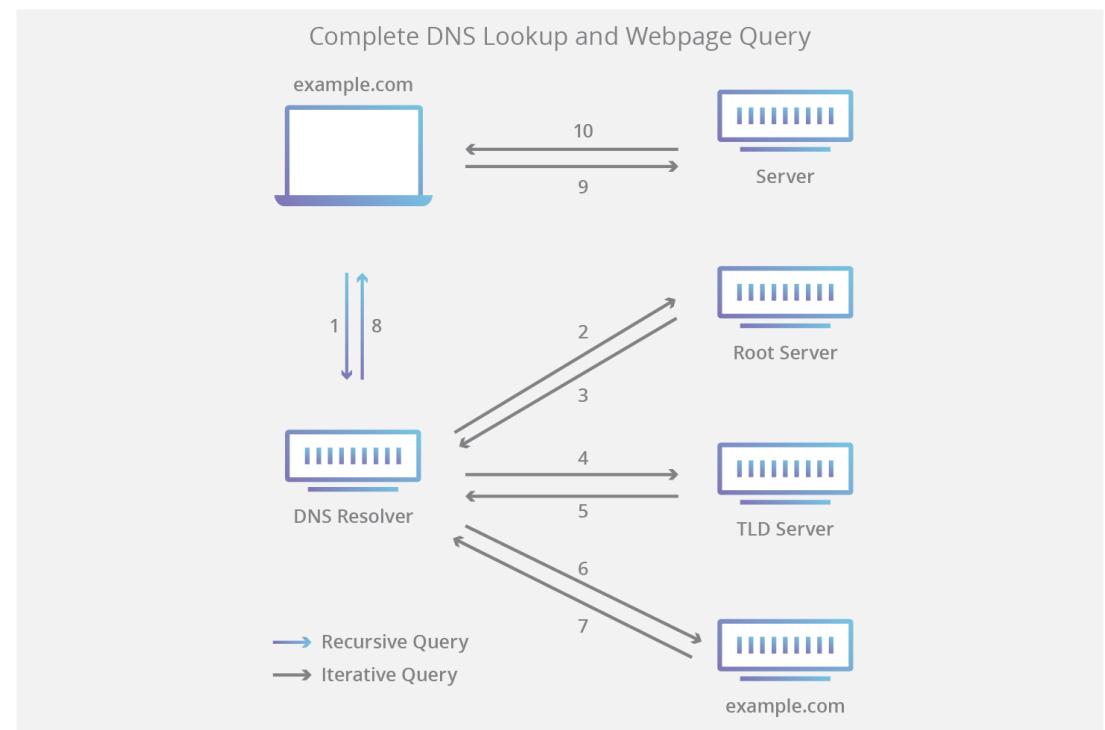
- Name resolution is the process by which resolvers and name space servers cooperate to find data in the name space
- A DNS query has three parameters:
 - A domain name (e.g., www.example.com)
 - A class (e.g., IN), and
 - A type (e.g., A)
- Upon receiving a query from a resolver, a name server will
 1. look for answer in its authoritative data and its cache
 2. if step 1 fails, the answer must be looked up

The 8 steps in a DNS lookup

1. A user types 'example.com' into a web browser and the query travels into the Internet and is received by a DNS recursive resolver.
2. The resolver then queries a DNS root nameserver (.).
3. The root server then responds to the resolver with the address of a Top-Level Domain (TLD) DNS server (such as .com or .net), which stores the information for its domains. When searching for example.com, our request is pointed toward the .com TLD.
4. The resolver then makes a request to the .com TLD.
5. The TLD server then responds with the IP address of the domain's nameserver, example.com.
6. Lastly, the recursive resolver sends a query to the domain's nameserver.
7. The IP address for example.com is then returned to the resolver from the nameserver.
8. The DNS resolver then responds to the web browser with the IP address of the domain requested initially.

Once the 8 steps of the DNS lookup have returned the IP address for example.com, the browser is able to make the request for the web page:

- The browser makes a HTTP request to the IP address.
- The server at that IP returns the webpage to be rendered in the browser (step 10).



DNS Query Resolution Process Example

```
sh-3.2# dig +trace @8.8.8.8 www.northeastern.edu
; <>> DiG 9.8.3-P1 <>> +trace @8.8.8.8 www.northeastern.edu
;(1 server found)
;; global options: +cmd
.
.          160738  IN  NS  c.root-servers.net.
.
.          160738  IN  NS  l.root-servers.net.
.
.          160738  IN  NS  m.root-servers.net.
.
.          160738  IN  NS  b.root-servers.net. [highlight]
.
.          160738  IN  NS  f.root-servers.net.
.
.          160738  IN  NS  i.root-servers.net.
.
.          160738  IN  NS  a.root-servers.net.
.
.          160738  IN  NS  h.root-servers.net.
.
.          160738  IN  NS  k.root-servers.net.
.
.          160738  IN  NS  g.root-servers.net.
.
.          160738  IN  NS  d.root-servers.net.
.
.          160738  IN  NS  j.root-servers.net.
.
.          160738  IN  NS  e.root-servers.net.
;
;; Received 228 bytes from 8.8.8.8#53(8.8.8.8) in 38 ms

edu.      172800  IN  NS  c.edu-servers.net. [highlight]
edu.      172800  IN  NS  l.edu-servers.net.
edu.      172800  IN  NS  d.edu-servers.net.
edu.      172800  IN  NS  f.edu-servers.net.
edu.      172800  IN  NS  g.edu-servers.net.
edu.      172800  IN  NS  a.edu-servers.net.
;
;; Received 273 bytes from 192.228.79.201#53(192.228.79.201) in 89 ms

northeastern.edu. 172800  IN  NS  ns20.customer.level3.net.
northeastern.edu. 172800  IN  NS  ns29.customer.level3.net.
northeastern.edu. 172800  IN  NS  nb4276.neu.edu.
northeastern.edu. 172800  IN  NS  nb4277.neu.edu.
;
;; Received 205 bytes from 192.26.92.30#53(192.26.92.30) in 15 ms

www.northeastern.edu. 600  IN  A   155.33.17.68
northeastern.edu. 3600  IN  NS  ns20.customer.level3.net.
northeastern.edu. 3600  IN  NS  ns29.customer.level3.net.
northeastern.edu. 3600  IN  NS  nb4276.neu.edu.
northeastern.edu. 3600  IN  NS  nb4277.neu.edu.
```

```
www.northeastern.edu. 600  IN  A   155.33.17.68
northeastern.edu. 3600  IN  NS  ns20.customer.level3.net.
northeastern.edu. 3600  IN  NS  ns29.customer.level3.net.
northeastern.edu. 3600  IN  NS  nb4276.neu.edu. [highlight]
northeastern.edu. 3600  IN  NS  nb4277.neu.edu.
;
;; Received 189 bytes from 155.33.16.201#53(155.33.16.201) in 9 ms
```

Internet Corporation for Assigned Names and Numbers (ICANN)

- Not-for-profit corporation
- Manages the IP namespace
- Controls the Top-level names
- Manages/oversees the root servers

The Root Name Servers

- The root zone file lists the name and IP addresses of the authoritative DNS servers for all top-level domains (TLDs).
- The root zone file is published on 13 root servers.
 - However, as there are an incredible number of names to resolve every minute, each of these servers is mirrored.
 - The interesting thing about this set up is that each of the mirrors for a single root server share the same IP address.
 - When requests are made for a certain root server, the request will be routed to the nearest mirror of that root server.
- The Root name server operations is currently provided by volunteer efforts by a very diverse set of organizations.

Registries, Registrars, and Registrants

- A Registrant's domain gets registered by a Registrar who is accredited by a Registry.
- Registries - gTLD and ccTLD Registries.
- Registrars - GoDaddy, 1&1, Namecheap, etc.

Risks of ccTLD

<https://eurid.eu/en/register-a-eu-domain/brexit-notice/>

Brexit notice

On 28 March 2018 the European Commission issued a notice to stakeholders concerning the .eu domain names registered by UK residents. The notice reads:

"Subject to any transitional arrangement that may be contained in a possible withdrawal agreement, the EU regulatory framework for the .eu Top Level Domain will no longer apply to the United Kingdom as from the withdrawal date. [...]

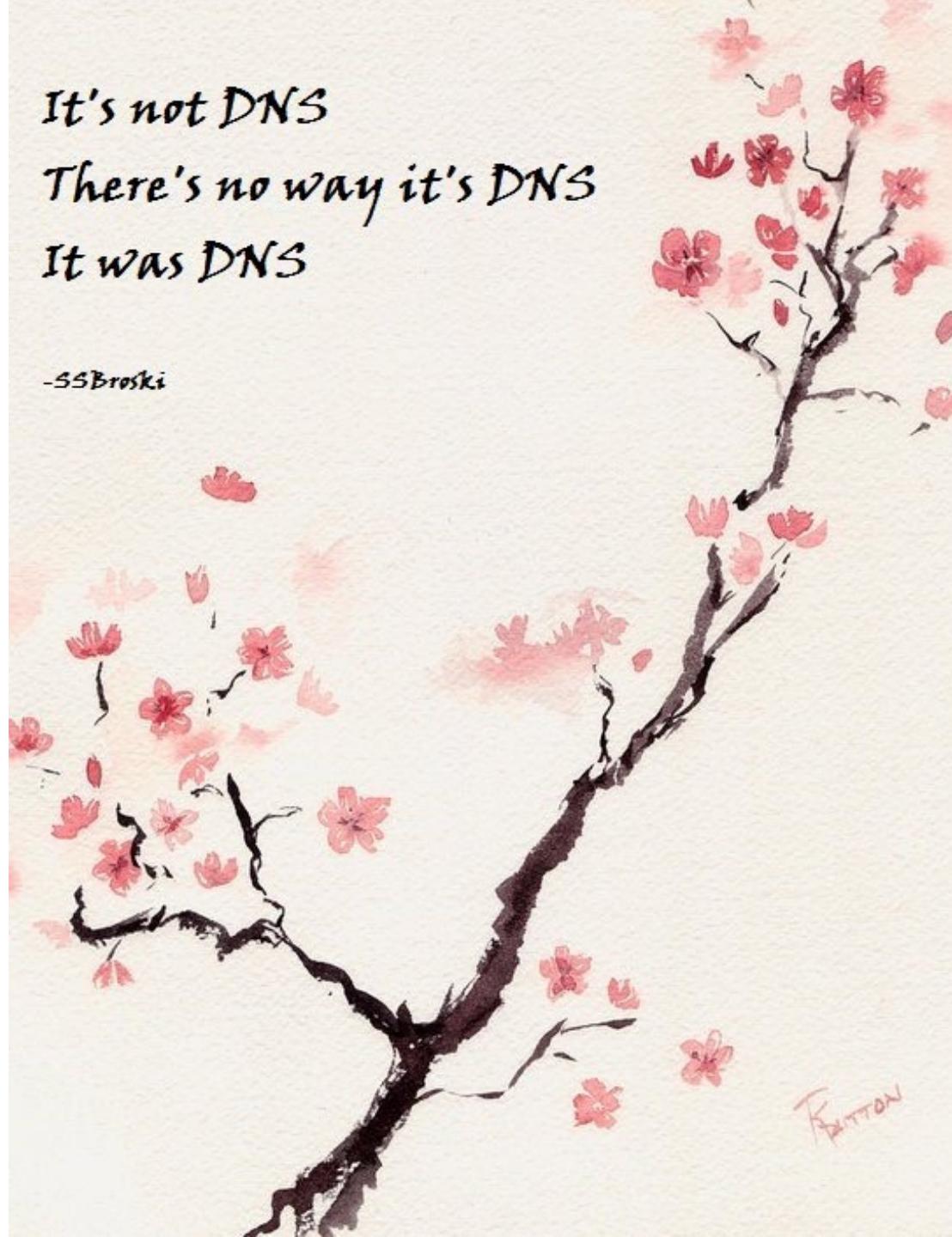
"As of the withdrawal date, undertakings and organisations that are established in the United Kingdom but not in the EU and natural persons who reside in the United Kingdom will no longer be eligible to register .eu domain names or, if they are .eu registrants, to renew .eu domain names registered before the withdrawal date. Accredited .eu Registrars will not be entitled to process any request for the registration of or for renewing registrations of .eu domain names by those undertakings, organisations and persons."

As reported above, the [full communication](#) highlights the fact that this information is subject to any transitional arrangement that may be contained in a possible withdrawal agreement, which is an ongoing negotiation between the United Kingdom and European Commission.

*It's not DNS
There's no way it's DNS
It was DNS*

-SSBroski

FATTON

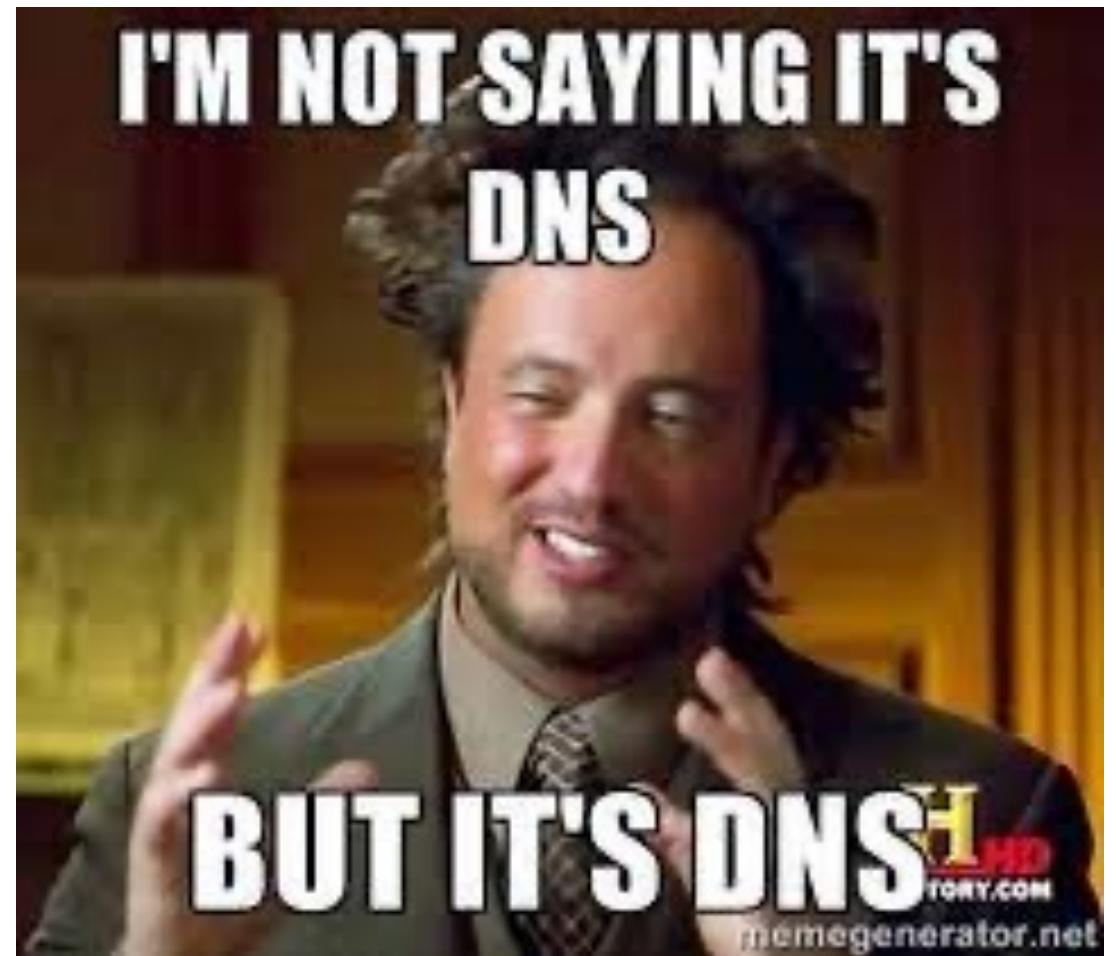


References

- <https://www.cloudflare.com/learning/dns/what-is-dns/>
- <https://www.digitalocean.com/community/tutorials/an-introduction-to-dns-terminology-components-and-concepts>
- <https://www.ibm.com/cloud/learn/dns>
- <https://github.com/ahupowerdns/hello-dns>

DNS Related Outages

- <https://www.cnet.com/news/how-pakistan-knocked-youtube-offline-and-how-to-make-sure-it-never-happens-again/>
- <https://www.wired.com/2016/10/internet-outage-ddos-dns-dyn/>
- <http://www.zdnet.com/article/global-dns-outage-hits-microsoft-azure-customers/>
- <http://www.businessinsider.com/google-is-down-2015-3>
- <https://www.digitalocean.com/company/blog/update-on-the-march-24-2016-dns-outage/>
- <https://www.wired.com/2012/09/godaddy-goes-down/>
- <https://nakedsecurity.sophos.com/2019/02/01/dns-outage-turns-tables-on-azure-database-users/>



Additional Resources

See Lecture Page

Site Reliability Engineering (SRE)

Tejas Parikh t.parikh@northeastern.edu

CSYE 6225
Northeastern University



Cost of System

- Software engineering has this in common with having children: the labor *before* the birth is painful and difficult, but the labor *after* the birth is where you spend most of your effort.
- Software engineering as a discipline spends much more time talking about the first period as opposed to second, despite estimates that 40-90% of the total costs of a system are incurred after birth.
- The popular industry model that conceives of deployed, operational software as being "stabilized" in production, and therefore needing much less attention from software engineers, is wrong.

“ *Hope is not a strategy.*”

Traditional SRE saying

Traditional Systems Administrator Role

- This systems administrator, or sysadmin, approach involves assembling existing software components and deploying them to work together to produce a service.
- Sysadmins are then tasked with running the service and responding to events and updates as they occur.

The Conflict

- The development teams want to launch new features and see them adopted by users as quickly as possible.
- The operations teams want to make sure the service doesn't break while they are holding the pager. Because most outages are caused by some kind of change—a new configuration, a new feature launch, or a new type of user traffic
- the two teams' goals are fundamentally in tension.

**MY COWORKERS
WATCHING ME DEPLOY A
"SMALL FIX" ON A FRIDAY**



What is Site Reliability Engineering?

- **Site reliability engineering (SRE)** is a discipline that incorporates aspects of software engineering and applies that to operations whose goals are to create ultra-scalable and highly reliable software systems.
- Ben Treynor, founder of Google's Site Reliability Team defines SRE as "*what happens when a software engineer is tasked with what used to be called operations.*"

DevOps or SRE?

- The term “DevOps” emerged in industry in late 2008 and is still in a state of flux.
- Its core principles—involve ment of the IT function in each phase of a system’s design and development, heavy reliance on automation versus human effort, the application of engineering practices and tools to operations tasks—are consistent with many of SRE’s principles and practices.
- One could view DevOps as a generalization of several core SRE principles to a wider range of organizations, management structures, and personnel.
- One could equivalently view SRE as a specific implementation of DevOps with some idiosyncratic extensions.

Tenets of SRE

In general, an SRE team for a service is responsible for the following:

- Availability
- Latency
- Performance
- Efficiency
- Change management
- Monitoring
- Emergency response
- Capacity planning

Eliminating Toil

“ *If a human operator needs to touch your system during normal operations, you have a bug. The definition of normal changes as your systems grow.* **”**

Carla Geisser. Google SRE

Toil is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows.

Error Budget

- The error budget stems from the observation that 100% is the wrong reliability target for basically everything (pacemakers and anti-lock brakes being notable exceptions).
- In general, for any software service or system, 100% is not the right reliability target because no user can tell the difference between a system being 100% available and 99.999% available.
- There are many other systems in the path between user and service (their laptop, their home WiFi, their ISP, the power grid...) and those systems collectively are far less than 99.999% available.
- Thus, the marginal difference between 99.999% and 100% gets lost in the noise of other unavailability, and the user receives no benefit from the enormous effort required to add that last 0.001% of availability.

Error Budget (contd.)

- If 100% is the wrong reliability target for a system, what, then, is the right reliability target for the system? This actually isn't a technical question at all—it's a product question, which should take the following considerations into account:
 1. What level of availability will the users be happy with, given how they use the product?
 2. What alternatives are available to users who are dissatisfied with the product's availability?
 3. What happens to users' usage of the product at different availability levels?
- The business or the product must establish the system's availability target. Once that target is established, the error budget is one minus the availability target. A service that's 99.99% available is 0.01% unavailable. That permitted 0.01% unavailability is the service's **error budget**.

Embracing Risk

- Reliability isn't linear in cost. It can easily cost 100x more to get one additional increment of reliability.
 - Cost associated with redundant equipment.
 - Cost of building out features for reliability as opposed to “normal” features.
 - Goal: make systems reliable enough, but not too reliable!
- Example: if a user is on a smartphone with 99% reliability, they can't tell the difference between 99.99% and 99.999% reliability

Measuring Service Risk

- For most services, the most straightforward way of representing risk tolerance is in terms of the acceptable level of unplanned downtime.
- Unplanned downtime is captured by the desired level of service availability, usually expressed in terms of the number of "nines" we would like to provide: **99.9%**, **99.99%**, or **99.999%** availability.
- Each additional nine corresponds to an order of magnitude improvement toward **100%** availability.

Time-based availability

- Using this formula over the period of a year, we can calculate the acceptable number of minutes of downtime to reach a given number of nines of availability.
- A time-based metric for availability is usually not meaningful when we are looking across globally distributed services.

$$\text{availability} = \frac{\text{uptime}}{(\text{uptime} + \text{downtime})}$$

Aggregate Availability

- Define availability in terms of the request success rate.
- For example, a system that serves 2.5M requests in a day with a daily availability target of 99.99% can serve up to 250 errors and still hit its target for that given day.
- *Not all requests are equal: failing a new user sign-up request is different from failing a request polling for new email in the background.*

$$\text{availability} = \frac{\text{successful requests}}{\text{total requests}}$$

Risk Tolerance of Services

- SREs work with product owners to translate business objectives into explicit objectives.

Service Level Objectives - Terminology

- Indicators - An SLI is a **service level indicator**—a carefully defined quantitative measure of some aspect of the level of service that is provided. Example, Consider request latency—how long it takes to return a response to a request—as a key SLI. Other common SLIs include the error rate, often expressed as a fraction of all requests received, and system throughput, typically measured in requests per second.
- Objectives - An SLO is a **service level objective**: a target value or range of values for a service level that is measured by an SLI. A natural structure for SLOs is thus **SLI \leq target**, or **lower bound \leq SLI \leq upper bound**. For example, we might decide that we will return Shakespeare search results "quickly," adopting an SLO that our average search request latency should be less than 100 milliseconds.
- Agreements - SLAs are service level agreements: an explicit or implicit contract with your users that includes consequences of meeting (or missing) the SLOs they contain. The consequences are most easily recognized when they are financial—a rebate or a penalty—but they can take other forms. An easy way to tell the difference between an SLO and an SLA is to ask "what happens if the SLOs aren't met?": if there is no explicit consequence, then you are almost certainly looking at an SLO.¹⁶

Collecting Indicators

- Many indicator metrics are most naturally gathered on the server side, using a monitoring system.
- Some indicators must be collected on client side.

“ *May the queries flow, and the pager stay silent.*”

Traditional SRE blessing

Practical Alerting from Time-Series Data

- Monitoring, is fundamental to running a stable service.
- Monitoring enables service owners to make rational decisions about the impact of changes to the service, apply the scientific method to incident response, and of course ensure their reason for existence: to measure the service's alignment with business goals.

Emergency Response

- Things break; that's life.
- Few of us naturally respond well during an emergency.
- A proper response takes preparation and periodic, pertinent, hands-on training.
- The way you respond to a situation is almost as important as the remedy itself.
- [BA038 777 Crash ATC Recording](#)

Postmortem: Learning from Failure

- Learn from the Past. Don't Repeat It.
- A postmortem is a written record of an incident, its impact, the actions taken to mitigate or resolve it, the root cause(s), and the follow-up actions to prevent the incident from recurring.
- Avoid Blame and Keep It Constructive
- Collaborate and Share Knowledge
- No Postmortem Left Unreviewed

Additional Resources

See Lecture Page

Observability

(Logging, Metrics, Monitoring & Alerting)

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225

Northeastern University

Logging

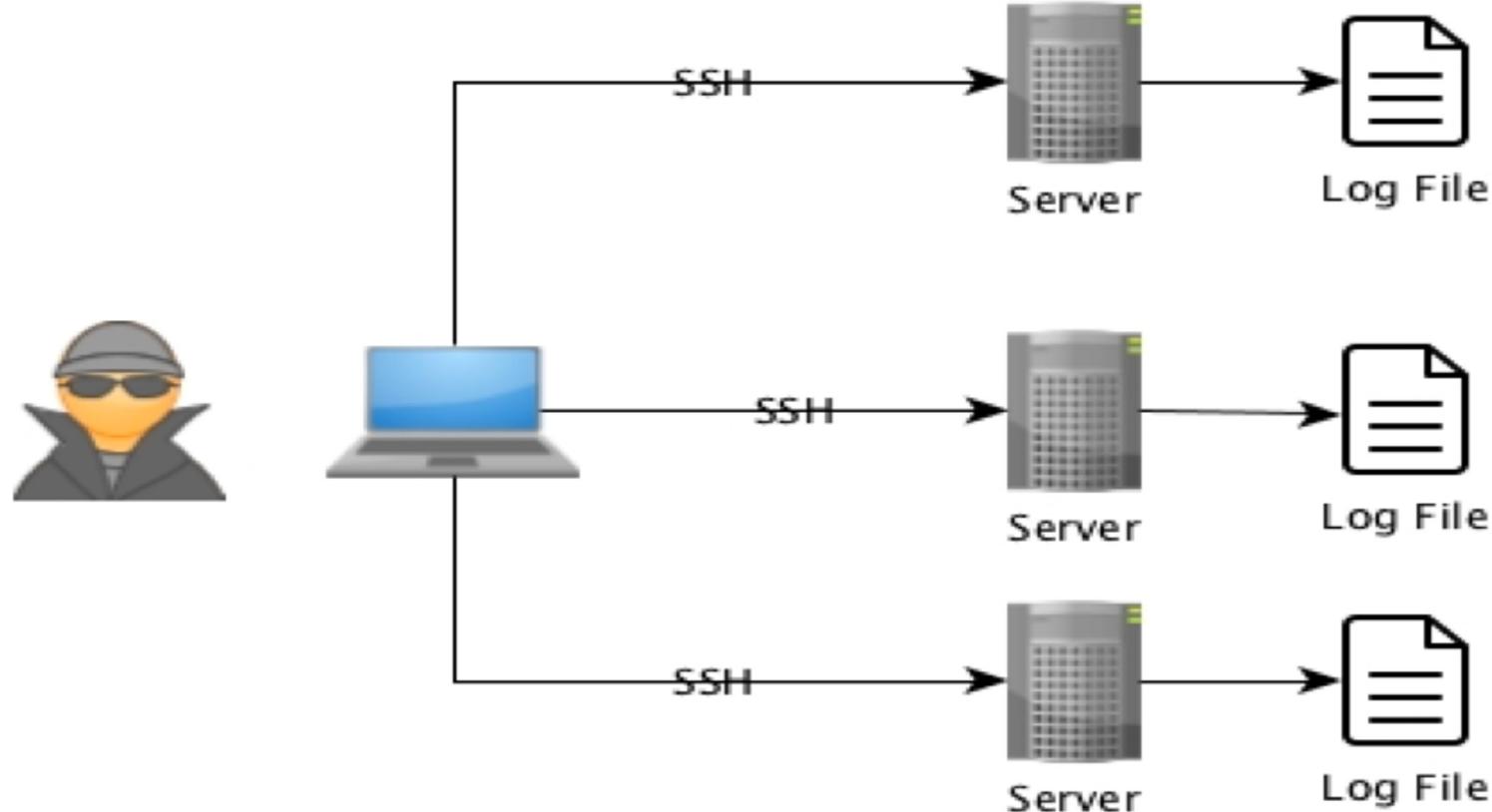
Understanding the Concept of Logging

- Application logs are one of the most import piece of analytical data in cloud-based service infrastructure.
- Logs are critical to successful debugging, troubleshooting, monitoring, and error reporting in an application.

Old Ways to Look at Logs

- A lot of times these logs are written to flat file on the local disk. Whenever an issue is reported a developer or sysadmin connects to the system via SSH and looks at the log file to find out what happened. The tools available in situation like these are grep or a text editor with search functionality.
- This becomes cumbersome when the application is hosted on multiple servers, as you now have to scan thru 100's of log file spread across 10's of servers without any good tools. There is no way to look at all the logs from various servers aggregated based on criteria such as timestamp, log source, etc.
- A common approach to this problem is to use centralized logging where logs from various sources can be aggregated in a centralized location.

Old Ways to Look at Logs



AWS CloudWatch Features

- CloudWatch Logs lets you monitor and troubleshoot your systems and applications using your existing system, application, and custom log files.
- With CloudWatch Logs, you can monitor your logs, in near real-time, for specific phrases, values or patterns (metrics).
- For example, you could set an alarm on the number of errors that occur in your system logs or view graphs of web request latencies from your application logs.
- You can view the original log data to see the source of the problem if needed.
- Log data can be stored and accessed for as long as you need using highly durable, low-cost storage so you don't have to worry about filling up hard drives.

Logging

Best Practices for Logging

When to Log

- Logs are not generated automatically by your application.
- The application developer i.e. you must deliberately create them to provide insight.
- First and foremost, log data that will help you troubleshoot an issue without a debugger.
- Debugging an application is not the only reason to log though.
- Log data when important events occur such as errors, configuration changes, component startup or shutdown and more.
- You may also want to simply monitor your application to make sure it is working and undergoing activity of some sort.
- **Don't muddle the logs by logging everything you can.**

What to Log

- You can probably think of lot of information you want to put in a log file and the possibilities are technically endless.
- From a high level, the best logs messages tell you exactly what happened, and when, where and how.

Should provide sequences of events to understand what was the chain of process

Exceptions, some warnings, Errors, Config changes, Things being flaky, retries should be logged

Be mindful about both quality & quantity of log messages

Logging must be as asynchronous as possible

Exceptions and Errors

- Both caught and uncaught exceptions should be logged.
- If a certain exception happens too often it is good to know.
- You should log any information you have about the error (type, message, stack trace, input parameters, and what the application was doing when the error occurred).

Log level inheritance, say we log upto info, debug and trace would not be logged, this can be configured in your service

Warnings

- Sometimes issue arise that aren't quite errors and don't stop action from continuing but should be brought to somebody's attention at a later time. These issues indicate potential issues in the application.

Informational Messages

- To investigate problems reported by users of your service, it's helpful to store all input and output of the service.
- Investigating customer issues otherwise can be very difficult.
- If your services call another service, it may be helpful to log input and output of that service as well. This will help identifying issues at the right level (your service vs. their service).

Debug Messages

- Debug log messages aid you when fixing defect and/or following the flow of the code execution.
- You can log entry and exit of methods, input and output values and anything else that will help your team identify issues in your services.
- **Debug logging should NOT be enabled in production.**

Logging

How to Log

Use The Right Logging Levels

It is important to ensure those log messages are appropriate in content and severity.

Log Level	Description
ALL	All log levels including custom levels.
TRACE	Designates finer-grained informational events than the DEBUG.
DEBUG	Designates fine-grained informational events that are most useful to debug an application.
INFO	Designates informational messages that highlight the progress of the application at coarse-grained level.
WARN	Designates potentially harmful situations.
ERROR	Designates error events that might still allow the application to continue running.
FATAL	Designates very severe error events that will presumably lead the application to abort.
OFF	The highest possible rank and is intended to turn off logging.

Log Level Inheritance

- A log request of level p in a logger with level q, is enabled if $p \geq q$.
- For the standard levels, we have ALL < TRACE < DEBUG < INFO < WARN < ERROR < FATAL < OFF.

Production Log Level

By default the message priority should be no lower than INFO. That is, by default DEBUG message should NOT be seen in the logs.

Timestamp in Log Events

- The correct time is critical to understanding the proper sequence of events.
- Timestamps are critical for debugging, analytics, and deriving transactions.
- All log events should have the timestamp in GMT/UTC time zone to avoid confusion when looking at logs from servers hosted in various time zones.
- Logging framework will automatically add a timestamp field to the log message.

Use developer-friendly formats

- Write logs in developer-friendly formats like JavaScript Object Notation (JSON), which is also readable by humans.
- You also avoid the hassle of dealing with new line characters in messages such as exception stack trace, which can end up as multiple log messages instead of one.

Be mindful about both quality & quantity of log messages

Avoid Multi-Line Events If Log Format Is Not JSON

- Multi-line events generate a lot of segments, which can affect how data is indexed.
- Consider breaking multi-line events into separate events or use JSON format so that message stays as 1 event.

Log Locally To Files

If you log to a local file, it provides a local buffer and you aren't blocked if the network goes down.

Log Message Encoding

- Log in TEXT format only.
- Binary information cannot be processed by logging stack and cannot be searched and thus will be of no help.
- Log messages should be in ENGLISH only.

Override `toString()` and `hashCode()` method

- Always override `toString()` to give a good representation of the object.
- Too often you get output that looks like `SomeFancyObject@15ba6d5`, which is useless.

Guarded Logging

Always check if logging for particular log level (such as DEBUG) is enabled before assembling log message. This will save CPU cycles and improve performance.

```
// When logging is set to ERROR following message does not get logged but log4j ends up spending  
time to create the log message  
  
log.debug("fancy" + "log"+ "message");
```

```
//Optimal way to write debug log events  
  
if(log.isDebugEnabled()) {  
  
log.debug("fancy" + "log"+ "message");  
  
}
```

What Not to Log

Do not log sensitive data such as passwords, credit card information, social security number, or any other Personally Identifiable Information.

Java Logging Framework Libraries

- Apache log4j
- Apache Extras for Apache log4j
- Apache Commons Lang
- Json Smart
- log4j-jsonevent-layout

Where Does System.out & System.err Log Messages Go?

- System.out and System.err are both redirected to CATALINA_BASE/logs/catalina.out when using Tomcat's startup scripts (bin/startup.sh/.bat or bin/catalina.sh/.bat).
- Any code that writes to System.out or System.err will end up writing to that file.
- If your webapp uses System.out and/or System.err a lot, you can suppress this via the 'swallowOutput' attribute in your configuration element and send to different log files (configured elsewhere: see the documentation for configuring logging).

Metrics

Measure Anything, Measure Everything

- Application metrics usually provide the most important data point yet they are the hardest to collect.

Metrics can answer questions like

- How slow is this query?
- How many times is this page accessed?
- What does my JVM memory usage look like during normal operations?

Data Source Types

- **Counters** - A counter is a numeric value that gets incremented when some event, such as a client connecting to a server, occurs. When reported to monitoring systems, a counter will typically be converted to a rate of change by comparing two samples. Counter values should be monotonically increasing.
- **Gauge** - Gauges store an arbitrary value. Simply put, it takes any number and gets flushed to the backend.
- **Timers** - Timers collect numbers. E.g. Response time for an API. They are not necessarily limited to store values of time.

Popular Client Libraries

- Statsd
- Netflix Servo

Popular Backend Datastores

- Graphite
- InfluxDB
- OpenTSDB
- Prometheus

Benefits of Metrics Data

- Metrics data helps with analysis of long term trend.
- Compare performance of application across different versions.
- Data can be used for Alerting.
- Data can be helpful in Debugging.

Alerting

Need for Alerting

- Automated alerts are essential to monitoring.
- They allow you to spot problems anywhere in your infrastructure, so that you can rapidly identify their causes and minimize service degradation and disruption.

Level of Alerting

- **Low Severity** - for record only.
- **Moderate Severity** - Usually via email notification. Requires human intervention but not right away.
- **High Severity** - The most urgent alerts that require human intervention right away. These are the alerts that can impact customer facing SLAs.

Additional Resources

See Lecture Page

CDN

Tejas Parikh (t.parikh@northeastern.edu)
CSYE 6225
Northeastern University

Content Delivery Network (CDN)

What is CDN?

- A **content delivery network or content distribution network** (CDN) is a geographically distributed network of proxy servers and their data centers.
- **CDN** is an umbrella term spanning different types of content delivery services: video streaming, software downloads, web and mobile content acceleration, licensed/managed CDN, transparent caching, and services to measure CDN performance, load balancing, multi-CDN switching and analytics and cloud intelligence.
- A CDN allows for the quick transfer of assets needed for loading Internet content including HTML pages, JavaScript files, stylesheets, images, and videos. *CDN caches data, so if we are not able to hit the backend servers, CDN can send the cached data to the client*
- The popularity of CDN services continues to grow, and today the majority of web traffic is served through CDNs, including traffic from major sites like Facebook, Netflix, and Amazon.

CDN can perform authorization with backend to retrieve the assets

What are the benefits of using a CDN?

- **Improving website load times** - By distributing content closer to website visitors by using a nearby CDN server (among other optimizations), visitors experience faster page loading times. As visitors are more inclined to click away from a slow-loading site, a CDN can reduce bounce rates and increase the amount of time that people spend on the site. In other words, a faster website means more visitors will stay and stick around longer.
- **Reducing bandwidth costs** - Bandwidth consumption costs for website hosting is a primary expense for websites. Through caching and other optimizations, CDNs are able to reduce the amount of data an origin server must provide, thus reducing hosting costs for website owners.
- **Increasing content availability and redundancy** - Large amounts of traffic or hardware failures can interrupt normal website function. Thanks to their distributed nature, a CDN can handle more traffic and withstand hardware failure better than many origin servers.
- **Improving website security** - A CDN may improve security by providing DDoS mitigation, improvements to security certificates, and other optimizations.

CDN Usecases

- Static Asset Caching
- Live and On-Demand Video Streaming
- Security and DDoS Protection
- API Acceleration
- Software Distribution

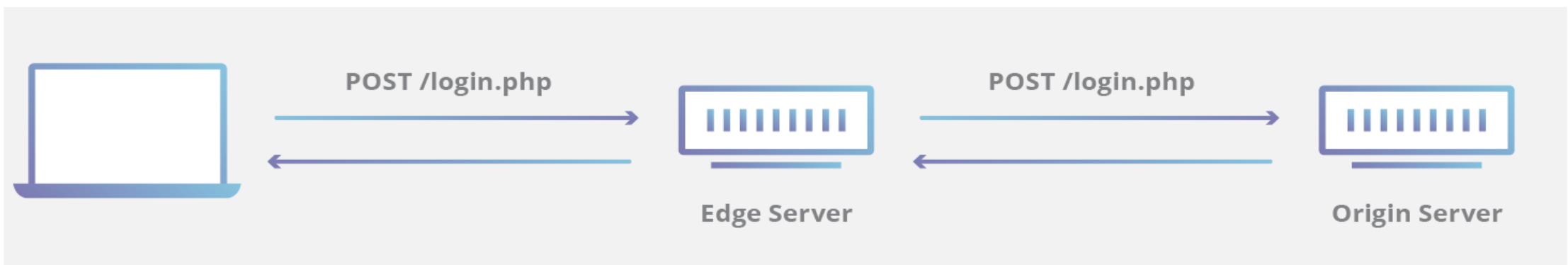
What is an Origin Server?

- The purpose of an **origin server** is to process and respond to incoming Internet requests from Internet clients.
- The concept of an origin server is typically used in conjunction with the concept of an edge server or caching server.
- At its core, an origin server is a computer running one or more programs that are designed to listen for and process incoming Internet requests.
- An origin server can take on all the responsibility of serving up the content for an Internet property such as a website, provided that the traffic does not extend beyond what the server is capable of processing and latency is not a primary concern.

We can now deploy firewalls on edge server with additional rules other than original backend server

What is a CDN edge server?

- A CDN **edge server** is a computer that exists at the logical extreme or “edge” of a network. An edge server often serves as the connection between separate networks.
- A primary purpose of a CDN edge server is to store content as close as possible to a requesting client machine, thereby reducing latency and improving page load times.



What is Anycast?

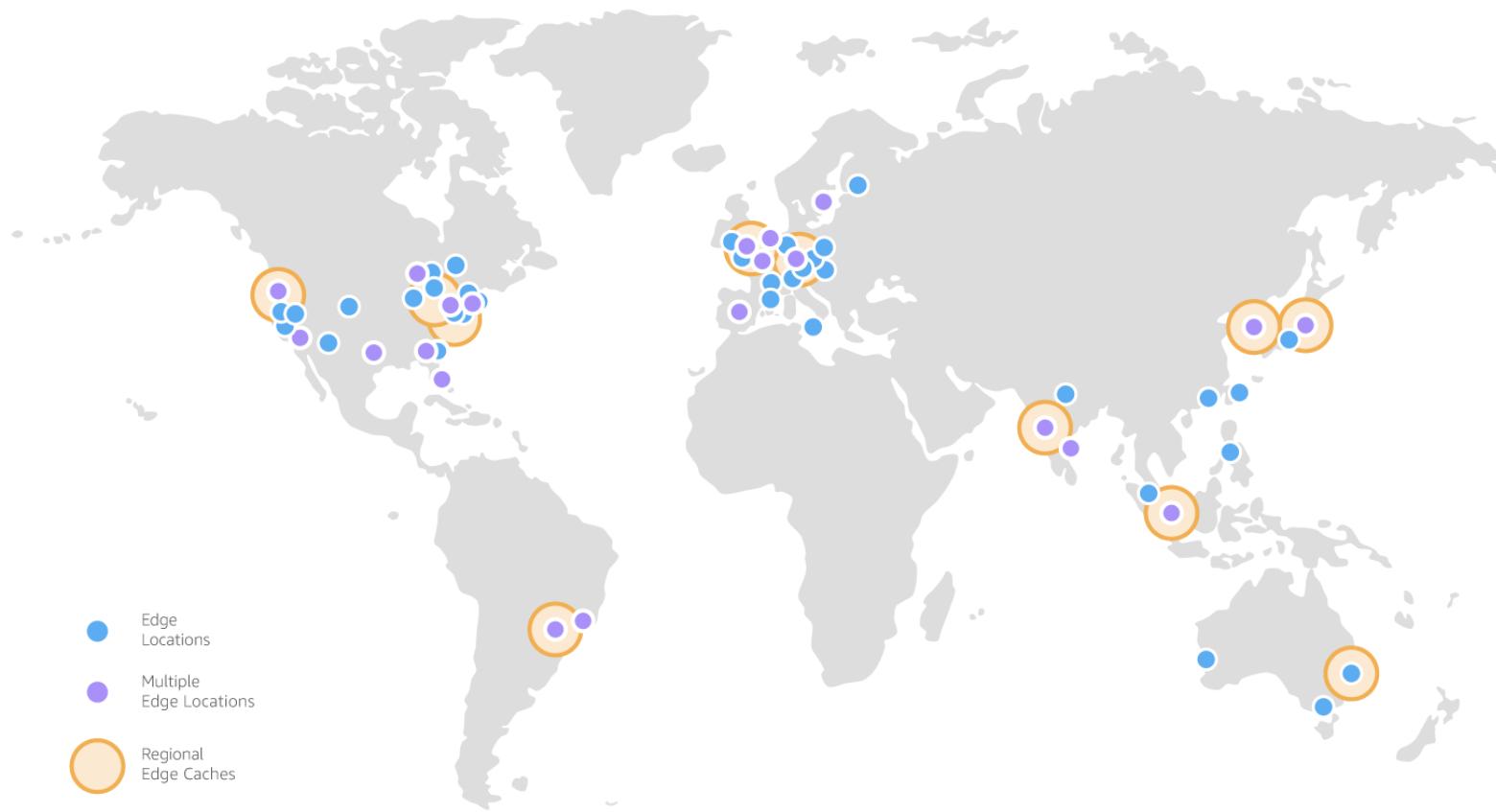
- Anycast is a network addressing and routing method in which incoming requests can be routed to a variety of different locations or “nodes.”
- In the context of a CDN, Anycast typically routes incoming traffic to the nearest data center with the capacity to process the request efficiently.
- Selective routing allows an Anycast network to be resilient in the face of high traffic volume, network congestion, and DDoS attacks.



How CDN Works

- At its core, a CDN is a network of servers linked together with the goal of delivering content as quickly, cheaply, reliably, and securely as possible.
- In order to improve speed and connectivity, a CDN will place servers at the exchange points between different networks.
- These Internet exchange points (IXPs) are the primary locations where different Internet providers connect in order to provide each other access to traffic originating on their different networks.
- By having a connection to these high speed and highly interconnected locations, a CDN provider is able to reduce costs and transit times in high speed data delivery.

The Amazon CloudFront Global Edge Network



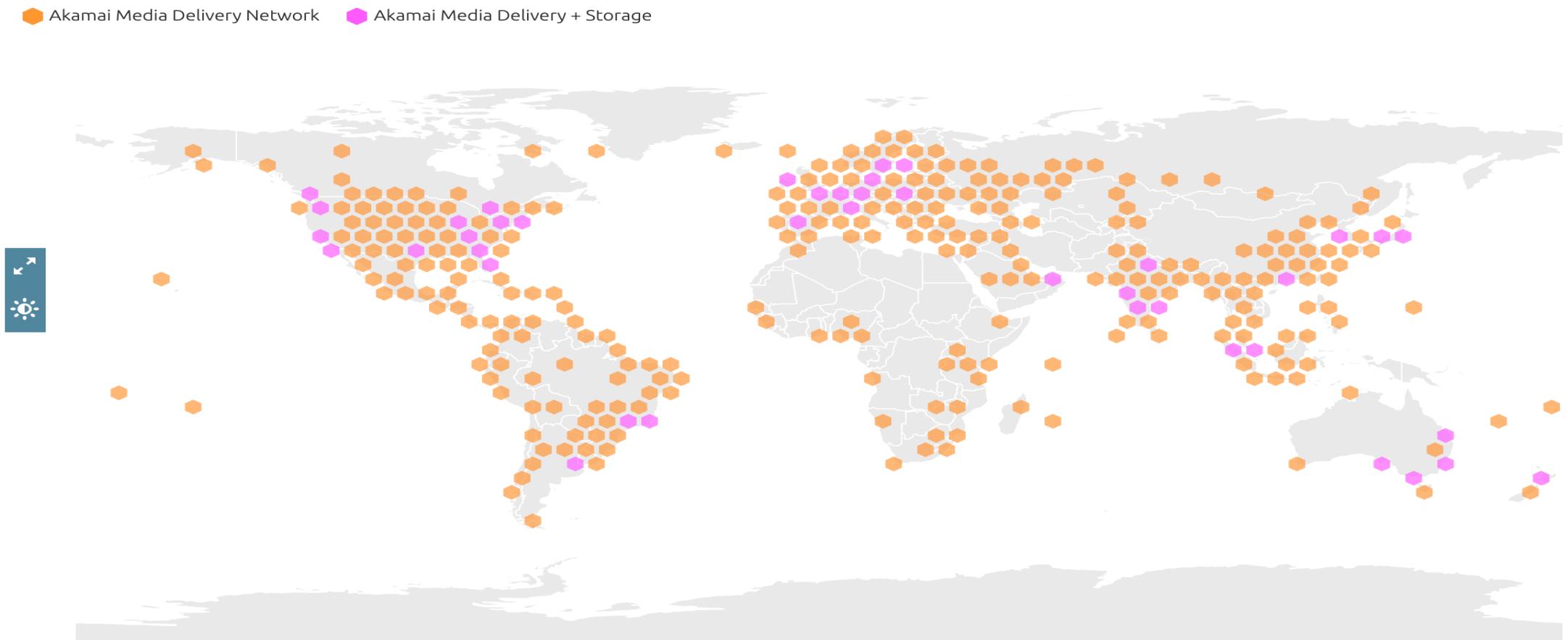
The Cloudflare Global Anycast Network

The Cloudflare Global Anycast Network

The Cloudflare network spans over 200 cities in more than 100 countries. [View system status >](#)



Akamai Media Delivery Network Map



Latency - How does a CDN improve website load times?

When it comes to websites loading content, users drop off quickly as a site slows down. CDN services can help to reduce load times in the following ways:

- The globally distributed nature of a CDN means reduce distance between users and website resources. Instead of having to connect to wherever a website's origin server may live, a CDN lets users connect to a geographically closer data center. Less travel time means faster service.
- CDNs can reduce the amount of data that's transferred by reducing file sizes using tactics such as minification and file compression. Smaller file sizes mean quicker load times.
- CDNs can also speed up sites which use TLS/SSL certificates by optimizing connection reuse and enabling TLS false start.

Reliability and Redundancy - How does a CDN keep a website always online?

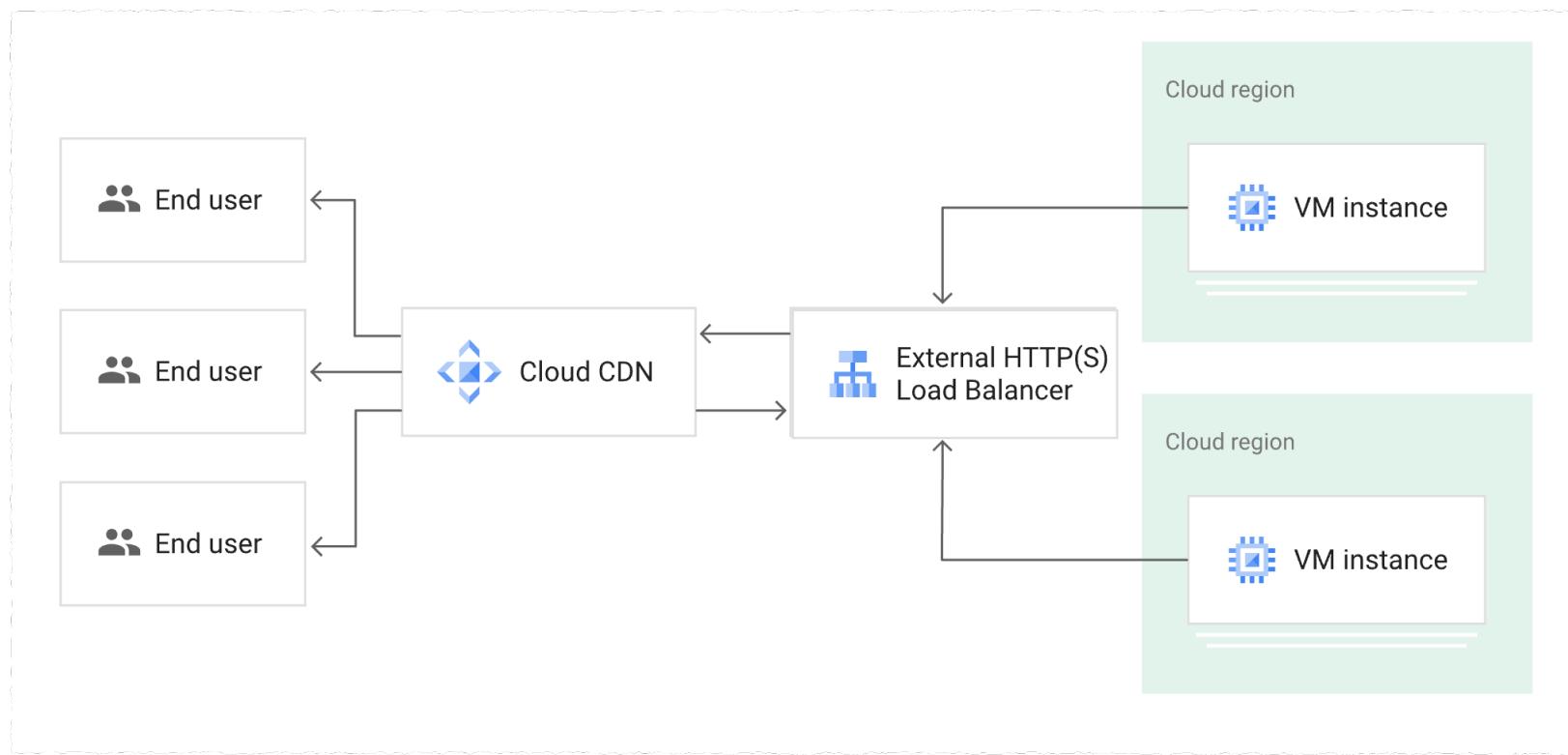
Uptime is a critical component for anyone with an Internet property. Hardware failures and spikes in traffic, as a result of either malicious attacks or just a boost in popularity, have the potential to bring down a web server and prevent users from accessing a site or service. A well-rounded CDN has several features that will minimize downtime:

- Load balancing distributes network traffic evenly across several servers, making it easier to scale rapid boosts in traffic.
- Intelligent failover provides uninterrupted service even if one or more of the CDN servers go offline due to hardware malfunction; the failover can redistribute the traffic to the other operational servers.
- In the event that an entire data center is having technical issues, Anycast routing transfers the traffic to another available data center, ensuring that no users lose access to the website.

Will (might) come for final Read about Eviction & Use signed cookies - Cloud CDN GCP

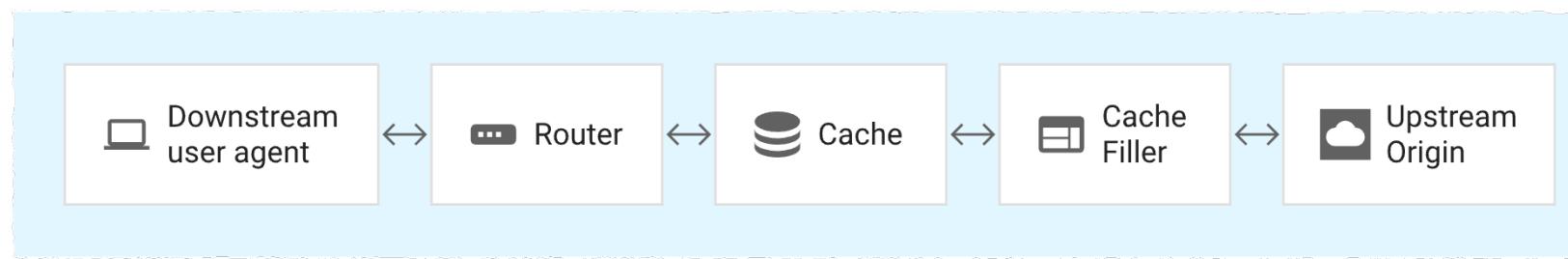
Cloud CDN - GCP

- <https://cloud.google.com/cdn/docs/overview>



Media CDN - GCP

- <https://cloud.google.com/media-cdn/docs/overview>



Negative Caching

- Negative caching lets you set a different TTL for each status code.

Status codes and default TTLs

Negative caching applies to specific status codes, which are listed in the following table.

★ Note: The TTL values for HTTP success codes (HTTP 2xx) are controlled by the values of [default TTL](#) and [max TTL](#).

Cloud CDN applies the following default TTLs to these status codes:

Status code	Meaning	TTL
HTTP 300	Multiple choices	10 minutes
HTTP 301 and 308	Permanent redirects	10 minutes
HTTP 302 and 307	Temporary redirects	Not cached by default
HTTP 404	Not found	120 seconds
HTTP 405	Method not found	60 seconds
HTTP 410	Gone	120 seconds
HTTP 451	Unavailable for legal reasons	120 seconds
HTTP 501	Not implemented	60 seconds

You can override these default values by using negative caching to set a cache TTL for the specified HTTP status code.

Additional Resources

See Lecture Page

Event-driven Architecture

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University

What is Pub/Sub?

- Pub/Sub is an asynchronous and scalable messaging service that decouples services producing messages from services processing those messages.
- Pub/Sub allows services to communicate asynchronously.
- Pub/Sub lets you create systems of event producers and consumers, called **publishers** and **subscribers**.
- Publishers communicate with subscribers asynchronously by broadcasting events, rather than by synchronous remote procedure calls (RPCs).

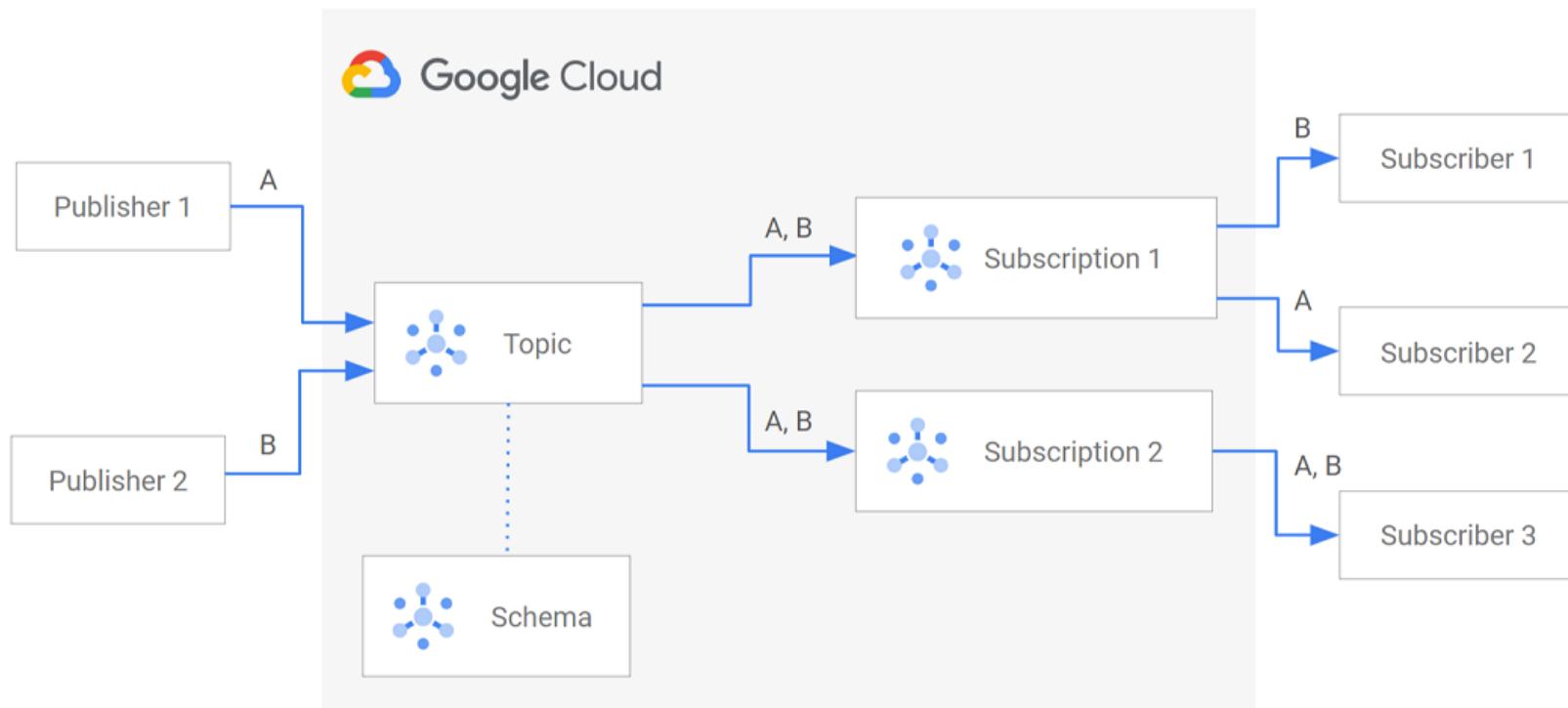
Common use cases

- **Ingesting user interaction and server events.** To use user interaction events from end-user apps or server events from your system, you might forward them to Pub/Sub. You can then use a stream processing tool, such as Dataflow, which delivers the events to databases. Examples of such databases are BigQuery, Bigtable, and Cloud Storage. Pub/Sub lets you gather events from many clients simultaneously.
- **Real-time event distribution.** Events, raw or processed, may be made available to multiple applications across your team and organization for real-time processing. Pub/Sub supports an "enterprise event bus" and event-driven application design patterns. Pub/Sub lets you integrate with many systems that export events to Pub/Sub.
- **Replicating data among databases.** Pub/Sub is commonly used to distribute change events from databases. These events can be used to construct a view of the database state and state history in BigQuery and other data storage systems.
- **Parallel processing and workflows.** You can efficiently distribute many tasks among multiple workers by using Pub/Sub messages to communicate with the workers. Examples of such tasks are compressing text files, sending email notifications, evaluating AI models, and reformatting images.
- **Enterprise event bus.** You can create an enterprise-wide real-time data sharing bus, distributing business events, database updates, and analytics events across your organization.
- **Data streaming from applications, services, or IoT devices.** For example, a SaaS application can publish a real-time feed of events. Or, a residential sensor can stream data to Pub/Sub for use in other Google Cloud products through a data-processing pipeline.
- **Refreshing distributed caches.** For example, an application can publish invalidation events to update the IDs of objects that have changed.
- **Load balancing for reliability.** For example, instances of a service may be deployed on Compute Engine in multiple zones but subscribe to a common topic. When the service fails in any zone, the others can pick up the load automatically.

Types of Pub/Sub services

- **Pub/Sub service.** This messaging service is the default choice for most users and applications. It offers the highest reliability and largest set of integrations, along with automatic capacity management. Pub/Sub supports synchronous replication of all data to at least two zones and best-effort replication to a third additional zone.
- **Pub/Sub Lite service.** A separate but similar messaging service built for lower cost. It offers lower reliability compared to Pub/Sub. It offers either zonal or regional topic storage. Zonal Lite topics are stored in only one zone. Regional Lite topics replicate data to a second zone asynchronously. Also, Pub/Sub Lite requires you to provision and manage storage and throughput capacity. Consider Pub/Sub Lite only for applications where achieving a low cost justifies some additional operational work and lower reliability.

Overview of the Pub/Sub service



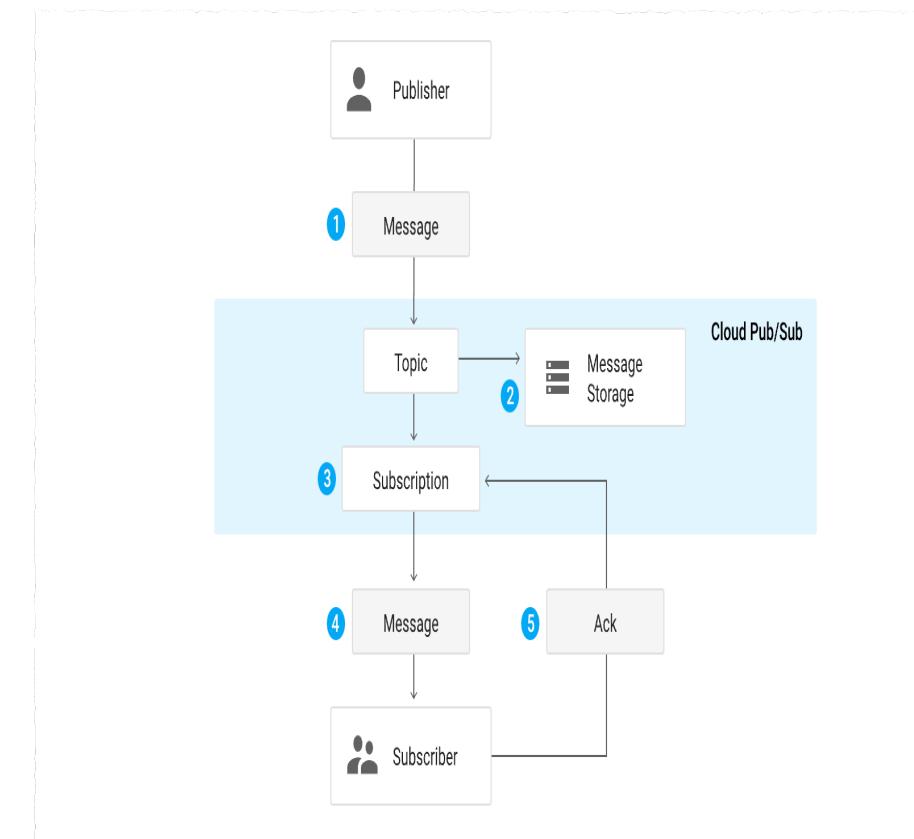
Components of a Pub/Sub service

- **Publisher** (also called a producer): creates messages and sends (publishes) them to the messaging service on a specified topic.
- **Message**: the data that moves through the service.
- **Topic**: a named entity that represents a feed of messages.
- **Schema**: a named entity that governs the data format of a Pub/Sub message.
- **Subscription**: a named entity that represents an interest in receiving messages on a particular topic.
- **Subscriber** (also called a consumer): receives messages on a specified subscription.

Lifecycle of a message in Pub/Sub

The following steps describe how a message flows in Pub/Sub:

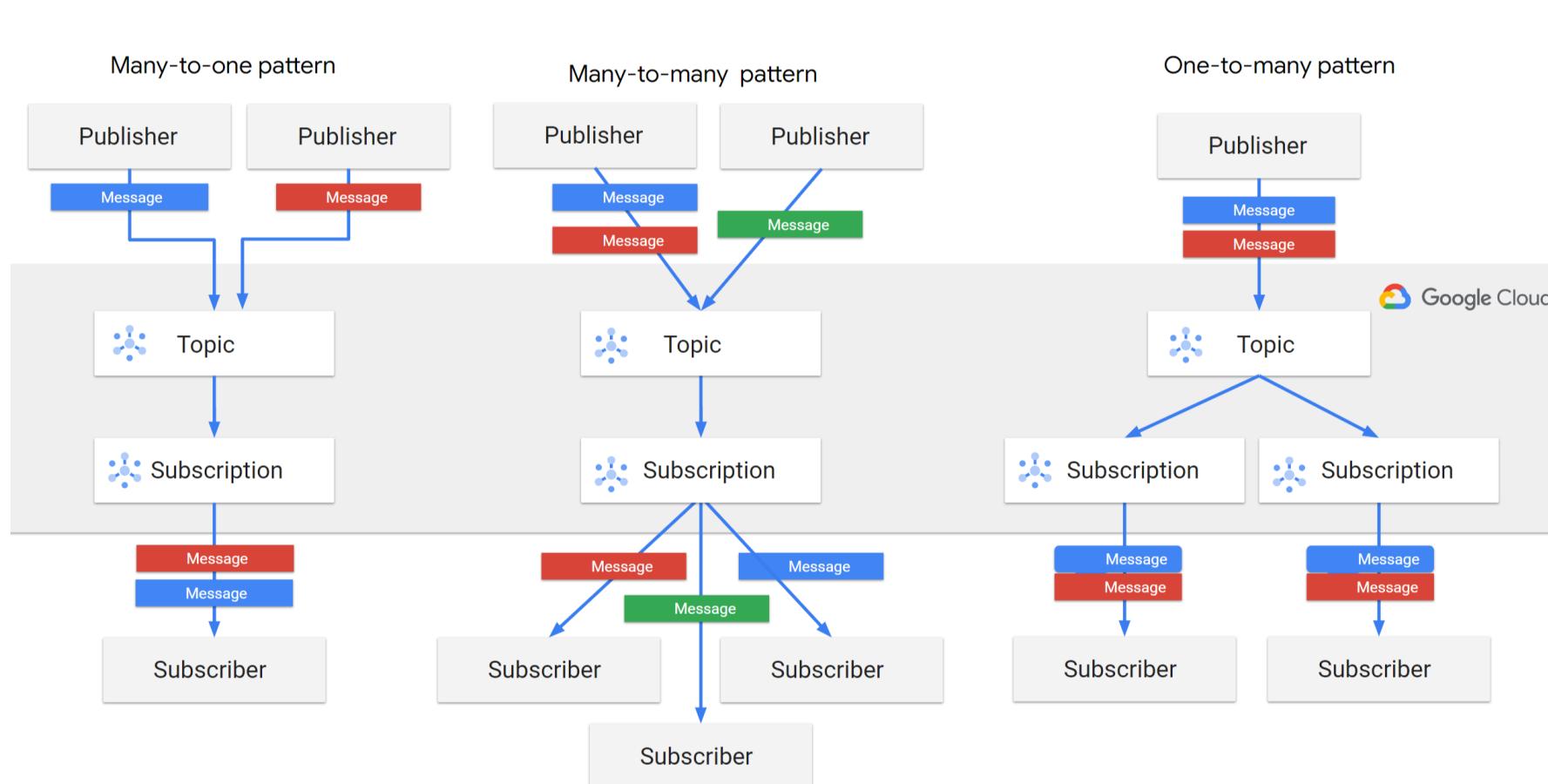
1. A publisher application sends a message to a Pub/Sub topic.
2. The message is written to storage.
3. Along with writing the message to storage, Pub/Sub delivers the message to all the attached subscriptions of the topic.
4. In this example, it's a single subscription.
5. The subscription sends the message to an attached subscriber application.
6. The subscriber sends an acknowledgment to Pub/Sub that they have processed the message.
7. After at least one subscriber for each subscription has acknowledged the message, Pub/Sub deletes the message from storage.



Status of a message in Pub/Sub

- There can be three states for a message in a Pub/Sub service:
- **Acknowledged messages (acked)**. After a subscriber application processes a message sent from a topic to a subscription, it sends an acknowledgment back to Pub/Sub. If all the subscriptions on a topic have acknowledged the message, the message is asynchronously deleted from the publish message source and from storage.
- **Unacknowledged messages (unacked)**. If Pub/Sub doesn't receive an acknowledgment within the acknowledgment deadline, a message might be delivered more than once. For example, the subscriber might send an acknowledgment after the deadline expires or the acknowledgment might be lost due to transient network issues. An unacknowledged message is continued to be delivered until the message retention duration expires since the message was published. At this point, the message expires.
- **Negatively acknowledged messages (nacked)**. Nacking a message by a subscriber causes Pub/Sub to redeliver it immediately. When a subscriber nacks messages that are invalid or when it cannot process the messages, the subscriber helps ensure that these messages are not lost and that they are eventually processed successfully. You can use [modifyAckDeadline](#) with a value of 0 to nack a message.

Pub/Sub publish and subscribe pattern



Pub/Sub publish and subscribe pattern

- **Fan in (many-to-one).** In this example, multiple publisher applications publish messages to a single topic. This single topic is attached to a single subscription. The subscription is, in turn, connected to a single subscriber application that gets all the published messages from the topic.
- **Load balanced (many-to-many).** In this example, a single or multiple publisher applications publish messages to a single topic. This single topic is attached to a single subscription that is, in turn, connected to multiple subscriber applications. Each of the subscriber applications gets a subset of the published messages, and no two subscriber applications get the same subset of messages. In this load balancing case, you use multiple subscribers to process messages at scale. If more messages need to be supported, you add more subscribers to receive messages from the same subscription.
- **Fan out (one-to-many).** In this example, a single or multiple publisher applications publish messages to a single topic. This single topic is attached to multiple subscriptions. Each subscription is connected to a single subscriber application. Each of the subscriber applications gets the same set of published messages from the topic. When a topic has multiple subscriptions, then every message has to be sent to a subscriber receiving messages on behalf of each subscription. If you need to perform different data operations on the same set of messages, fan out is a good option. You can also attach multiple subscribers to each subscription and get a load-balanced subset of messages for each subscriber.

Additional Resources

See Lecture Page

Serverless Computing aka Function as a Service (FaaS)

Tejas Parikh
[\(t.parikh@northeastern.edu\)](mailto:t.parikh@northeastern.edu)
CSYE 6225
Northeastern University



What is Serverless Computing?

Serverless only runs on an event - reactive compute design

- Serverless computing also known as function as a service (FaaS) refers to a model where the existence of servers is simply hidden from developers. I.e. that even though servers still exist developers are relieved from the need to care about their operation.
- Developers are relieved from the need to worry about low-level infrastructural and operational details such as scalability, high-availability, infrastructure-security, and so forth.
- Serverless computing is essentially about reducing maintenance efforts to allow developers to quickly focus on developing value-adding code.
- Serverless computing encourages and simplifies developing microservices oriented solutions in order to decompose complex applications into small and independent modules that can be easily exchanged.

Units of Scale

- Virtual Machines
 - Machine as the unit of scale
 - Abstracts the hardware
- Containers
 - Application as the unit of scale
 - Abstracts the OS
- Serverless
 - Functions as the unit of scale
 - Abstracts the language runtime

Reactive Computing Design

Code is triggered by events or called by APIs.

Example triggers:

- PUT in Amazon S3 bucket
- Updates to DynamoDB tables
- Message on Kinesis queue
- etc.

Benefits of Serverless Computing

- No servers to manage
- Continuous Scaling
- Never pay for idle servers
- Reduced Operational Costs

Drawbacks

- Loss of Server optimizations
- No in-server state for Serverless FaaS
- Startup Latency
- Vendor Lockin

Use Cases

- Event driven programming
- On-demand Lambda function invocation over HTTPS
- On-demand Lambda function invocation
- Scheduled events

Will (might) come for final Read about Eviction & Use signed cookies - Cloud Functions

Additional Resources

See Lecture Page

Microservices

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University

What Are Microservices?

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of loosely coupled services.

Microservices are...

- Small, and focused on doing one thing well
- autonomous

Microservices are small, and focused on doing one thing well

- Codebases grow as we write code to add new features.
- Over time, it can be difficult to know where a change needs to be made because the codebase is so large.
- Code related to similar functions starts to become spread all over, making fixing bugs or implementations more difficult.
- Microservices focus our service boundaries on business boundaries, making it obvious where code lives for a given piece of functionality.

Microservices are autonomous

- Microservices are a separate entity.
- Microservices are deployed as isolated service.
- All communication between the services themselves are via network calls, to enforce separation between the services and avoid the perils of tight coupling.
- Microservices need to be able to change independently of each other, and be deployed by themselves without requiring consumers to change.
- Services expose an API and other services communicate with it via those APIs.

Benefits of Microservices

- Technology Heterogeneity – Pick right tool (programming language, framework, operating system, database, etc.) for the job.
- Resiliency – A service failure does not cascade into total system failure.
- Scaling – Scale on the services that need to be scaled.
- Ease of Deployment – Service deployments are independent of the rest of the components.
- Composability – Microservices can be designed to allow its functionality be consumed in many different ways for different purposes.
- Optimizing for Replaceability – Easy to kill a service when not needed.

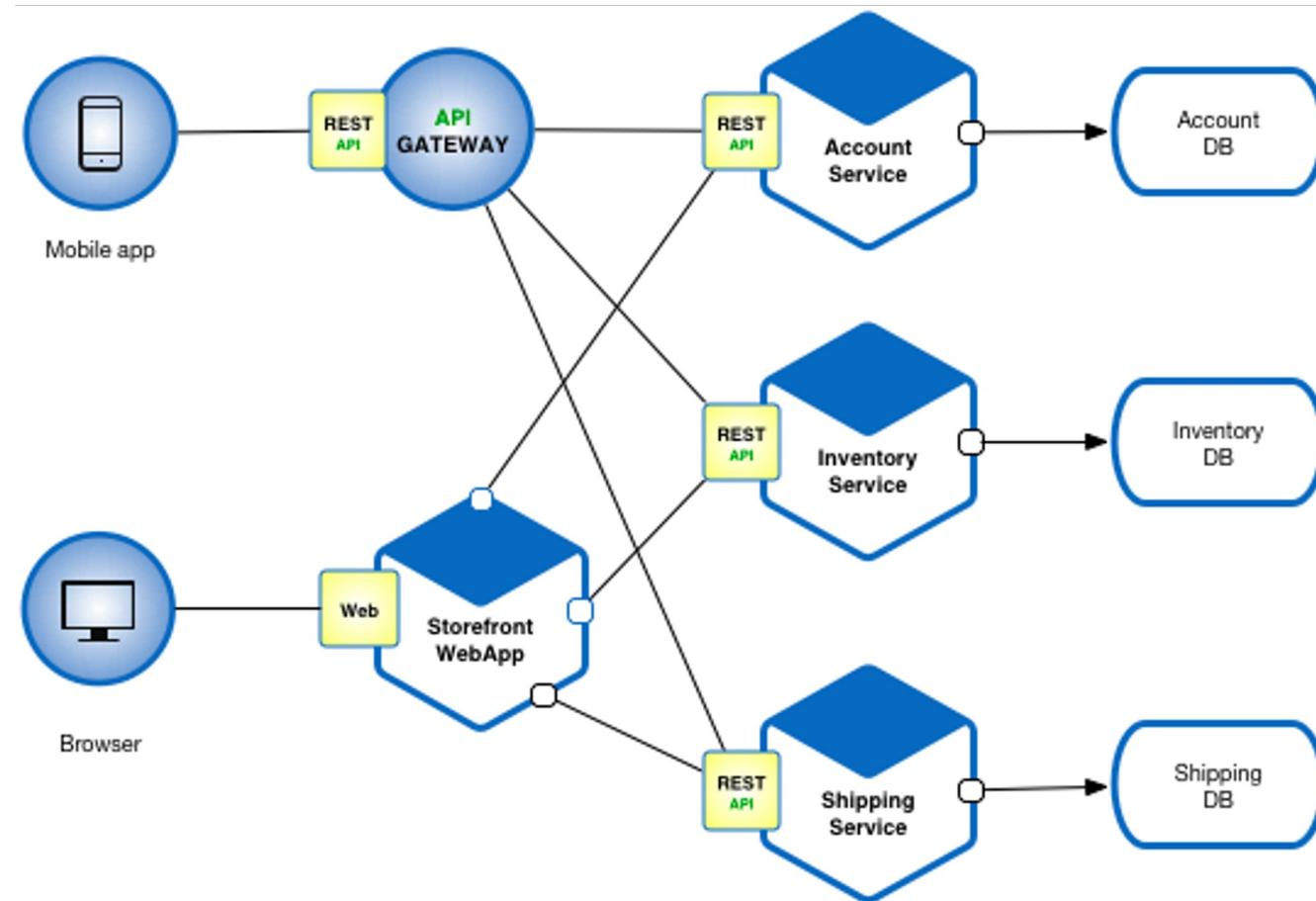
What Makes a Good Service?

- Loose Coupling – When services are loosely coupled, a change to one service should not require change to another.
- High Cohesion – Related behavior should sit together and unrelated behavior to sit elsewhere.

Integration

- Shared Database?
- Synchronous Versus Asynchronous
- Remote Procedure Calls **Protobuf & GRPCs**
- REST **when should we use it & when should we not?**
- Message Queues
- Reactive Design

Fictitious e-commerce application



Drawbacks of Microservices

- Services can be too small.
- Microservices architecture introduces additional complexity and new problems to deal with such as network latency, message formats, load balancing and fault tolerance.
- Calls between services over a network have higher cost in term of network latency and processing time compared to in-process calls in monolithic services.

Additional Resources

See Lecture Page

Load Balancing, Auto Scaling & Availability Zones

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225

Northeastern University

GCP

Partition of servers in the same building (single building multiple zones) - if there is a problem in the building, say generator or AC or hardware problem, both zones would be affected

May have same network interfaces - Low Network Latency

Zonal resources: Compute Engine, Storage Disk attached to VM

Resources are zonal, regional & global

Amazon

Partition of servers in different building - (1 - 20 mi) (one building one zone) - if there is problem with one building, say generator or AC or hardware problem, only one zone would be affected

Definitely have diff network interface - Network latency is higher

Zonal resources: EC2, Storage Disk attached to VM

Resources are zonal & regional

Regions & Availability Zone

AVAILABLE IN

40

REGIONS

121

ZONES

187

NETWORK EDGE LOCATIONS

200+

COUNTRIES AND TERRITORIES

COMING SOON! Google Cloud will continue expanding into the following regions: Mexico, Malaysia, Thailand, New Zealand, Greece, Norway, Austria and Sweden.



Regions & Zones

- A region is a specific geographical location where you can host your resources. Regions are collections of zones.
- A zone is a deployment area within a region. A single datacenter may contain multiple zones.
- Zones have high-bandwidth, low-latency network connections to other zones in the same region.
- In order to deploy fault-tolerant applications that have high availability, Google recommends deploying applications across multiple zones and multiple regions. This helps protect against unexpected failures of components, up to and including a single zone or region.

Warning about Zones

- **Definition of Zones is different across different Cloud Providers.**
- AWS Regions are separate geographic areas.
- AWS Regions consist of multiple, physically separated and isolated Availability Zones that are connected with low latency, high throughput, highly redundant networking.
- AWS Availability Zones consist of one or more discrete data centers, each with redundant power, networking, and connectivity, and housed in separate facilities

Auto Scaling

What is Autoscaling

- Autoscaling works by adding more VMs to your MIG when there is more load (scaling out), and deleting VMs when the need for VMs is lowered (scaling in).

Need for Auto Scaling

Saving cost

Availability?

Managed instance groups

- Autoscaling is a feature of **managed instance groups (MIGs)**.
- A managed instance group is a collection of virtual machine (VM) instances that are created from a common instance template.
- An autoscaler adds or deletes instances from a managed instance group based on the group's autoscaling policy.

Autoscaling policy

- When you define an autoscaling policy for your group, you specify one or more signals that the autoscaler uses to scale the group.
- When you set multiple signals in a policy, the autoscaler calculates the recommended number of VMs for each signal and sets your group's recommended size to the largest number.
- An autoscaling policy must always have at least one scaling signal.
- When you turn on autoscaling in a MIG, by default, the autoscaler adds a CPU utilization signal.

Target utilization metrics

Should not react to short Burst of high CPU util - say 90% util for only 2 min
Should scale up only if burst is of more freq & more time - sustained high util

- You can autoscale based on one or more of the following metrics that reflect the load of the instance group:
 - Average CPU utilization
 - HTTP load balancing serving capacity
 - Cloud Monitoring metrics
- The autoscaler continuously collects usage information based on the selected utilization metric, compares actual utilization to your desired target utilization, and uses this information to determine whether the group needs to remove instances (scale in) or add instances (scale out).
- The target utilization level is the level at which you want to maintain your virtual machine (VM) instances. For example, if you scale based on CPU utilization, you can set your target utilization level at 75% and the autoscaler will maintain the CPU utilization of the specified group of instances at or close to 75%.
- The utilization level for each metric is interpreted differently based on the autoscaling policy.

Schedules

- You can use schedule-based autoscaling to allocate capacity for anticipated loads. You can have up to 128 scaling schedules per instance group. For each scaling schedule, specify the following:
 - **Capacity:** minimum required VM instances
 - **Schedule:** start time, duration, and recurrence (for example, once, daily, weekly, or monthly)
- Each scaling schedule is active from its start time and for the configured duration. During this time, autoscaler scales the group to have at least as many instances as defined by the scaling schedule.

Initialization period (Cool down Period)

- The initialization period, formerly known as cool down period, is the duration it takes for applications to initialize on your VM instances. While an application is initializing on an instance, the instance's usage data might not reflect normal circumstances. So the autoscaler uses the initialization period for scaling decisions in the following ways:
 - For scale-in decisions, the autoscaler considers usage data from all instances, even an instance that is still within its initialization period. The autoscaler recommends to remove instances if the average utilization from all instances is less than the target utilization.
 - For scale-out decisions, the autoscaler ignores usage data from instances that are still in their initialization period.
- Actual initialization times vary because of numerous factors. We recommend that you test how long your application takes to initialize.

Stabilization period

AWS - stabilization time is based on how long it takes to spin up a vm- decide the next step - scaling up or down

Stabilization period - scaling down

- Autoscaling signals like CPU utilization are not very stable and can change rapidly.
- As the load goes up and down, the autoscaler needs to stabilize the signal to avoid continuous VM deletion and creation.
- The autoscaler stabilizes a signal by keeping sufficient VM capacity in order to serve the peak load that is observed during the *stabilization period*.
- The stabilization period is equal to 10 minutes or to the initialization period that you set, whichever is longer.
- **The stabilization period is used only for scale-in decisions when the autoscaler has to delete VMs.**
- When the load goes down, the autoscaler does not delete VMs immediately.
- The autoscaler keeps monitoring capacity needed for the duration of the stabilization period and deletes VMs only when there is sufficient capacity to meet the peak load.
- This might appear as a delay in scaling in, but it is a built-in feature of autoscaling.

Scale-in controls

- **Maximum allowed reduction**
 - The number of VM instances that your workload can afford to lose (from its peak size) within the specified trailing time window.
 - Use this parameter to limit how much your group can be scaled in so that you can still serve a likely load spike until more instances start serving.
 - The smaller you set the maximum allowed reduction, the longer it takes for your group to scale in.
- **Trailing time window**
 - The history within which the autoscaler monitors the peak size required by your workload.
 - The autoscaler will not resize below the maximum allowed reduction subtracted from the peak size observed in this period.
 - You can use this parameter to define how long the autoscaler should wait before removing instances, as defined by the maximum allowed reduction.
 - With a longer trailing time window, the autoscaler considers more historical peaks, making scale-in more conservative and stable.

What happens during autohealing

- Autoscaling works independently from autohealing.
- If you configure autohealing for your group and an instance fails the health check, the MIG attempts to recreate the instance.
- While an instance is being recreated by the MIG, the number of running instances in the group might be lower than the minimum number of instances specified for the group

Load Balancing



Can you check the load balancer?

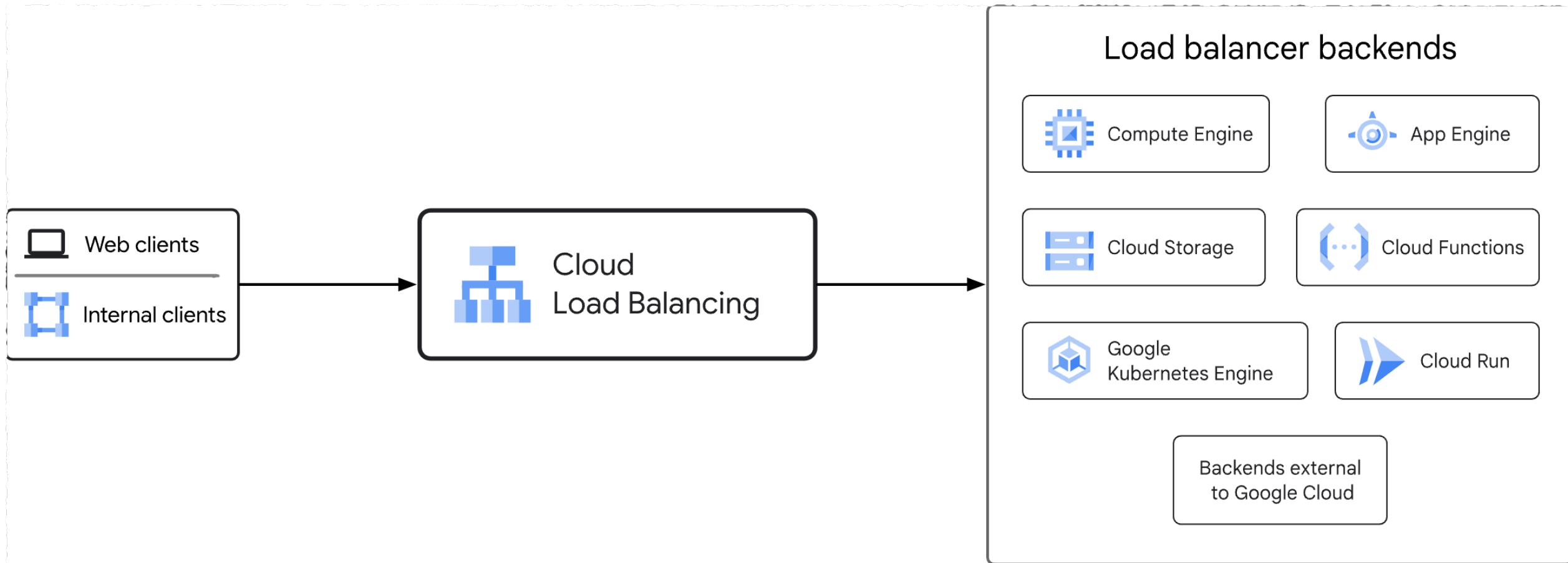
Server 1

Server 2



imgflip.com

Cloud Load Balancing



What Does A Load Balancer Do?

- A load balancer distributes user traffic across multiple instances of your applications.
- By spreading the load, load balancing reduces the risk that your applications experience performance issues.

Cloud Load Balancer Features

- **Single anycast IP address**
 - With Cloud Load Balancing, a single anycast IP address is the frontend for all of your backend instances in regions around the world.
 - It provides cross-region load balancing, including automatic multi-region failover, which moves traffic to failover backends if your primary backends become unhealthy.
 - Cloud Load Balancing reacts instantaneously to changes in users, traffic, network, backend health, and other related conditions.
- **Seamless autoscaling**
 - Cloud Load Balancing can scale as your users and traffic grow, including easily handling huge, unexpected, and instantaneous spikes by diverting traffic to other regions in the world that can take traffic.
 - Autoscaling does not require pre-warming: you can scale from zero to full traffic in a matter of seconds.
- **Software-defined load balancing**
 - Cloud Load Balancing is a fully distributed, software-defined, managed service for all your traffic.
 - It is not an instance-based or device-based solution, so you won't be locked into a physical load-balancing infrastructure or face the high availability, scale, and management challenges inherent in instance-based load balancers.

Cloud Load Balancer Features (contd.)

- **Layer 4 and Layer 7 load balancing**
 - Use Layer 4-based load balancing to direct traffic based on data from network and transport layer protocols such as TCP, UDP, ESP, GRE, ICMP, and ICMPv6 .
 - Use Layer 7-based load balancing to add request routing decisions based on attributes, such as the HTTP header and the uniform resource identifier.
- **External and internal load balancing**
 - You can use external load balancing when your users reach your applications from the internet.
 - You can use internal load balancing when your clients are inside of Google Cloud.
- **Global and regional load balancing**
 - You can distribute your load-balanced resources in single or multiple regions to terminate connections close to your users and to meet your high availability requirements.

Types of load balancers

Cloud Load Balancing offers two types of load balancers:
Application Load Balancers and Network Load Balancers.

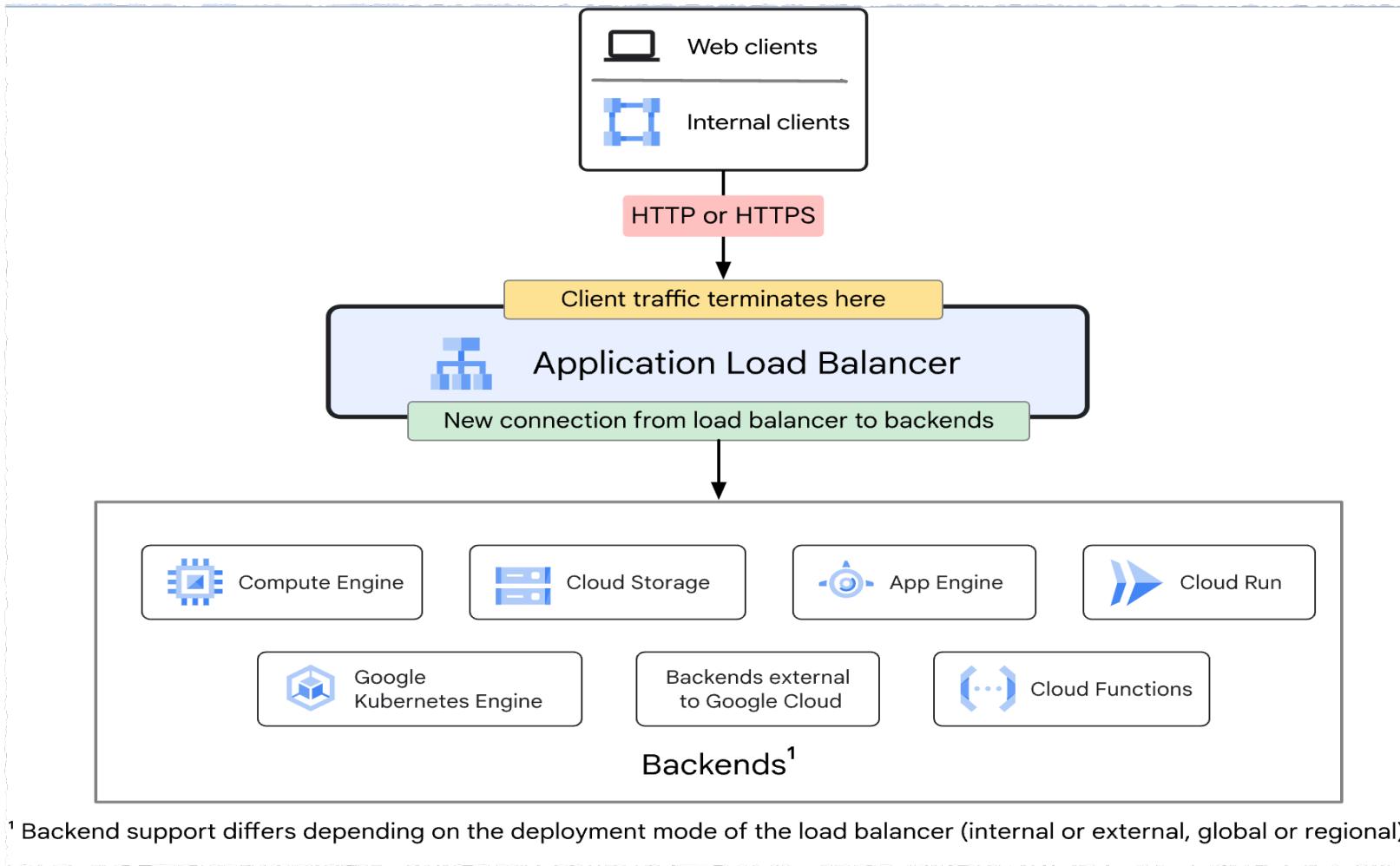
You'd choose an Application Load Balancer when you need a Layer 7 load balancer for your applications with HTTP(S) traffic.

You'd choose a Network Load Balancer when you need a Layer 4 load balancer that supports TLS offloading (with a proxy load balancer) or you need support for IP protocols such as UDP, ESP, and ICMP (with a passthrough load balancer).

Application Load Balancers

- Application Load Balancers are proxy-based Layer 7 load balancers that enable you to run and scale your services behind an anycast IP address.
- The Application Load Balancer distributes HTTP and HTTPS traffic to backends hosted on a variety of Google Cloud platforms—such as Compute Engine and Google Kubernetes Engine (GKE)—as well as external backends outside Google Cloud.

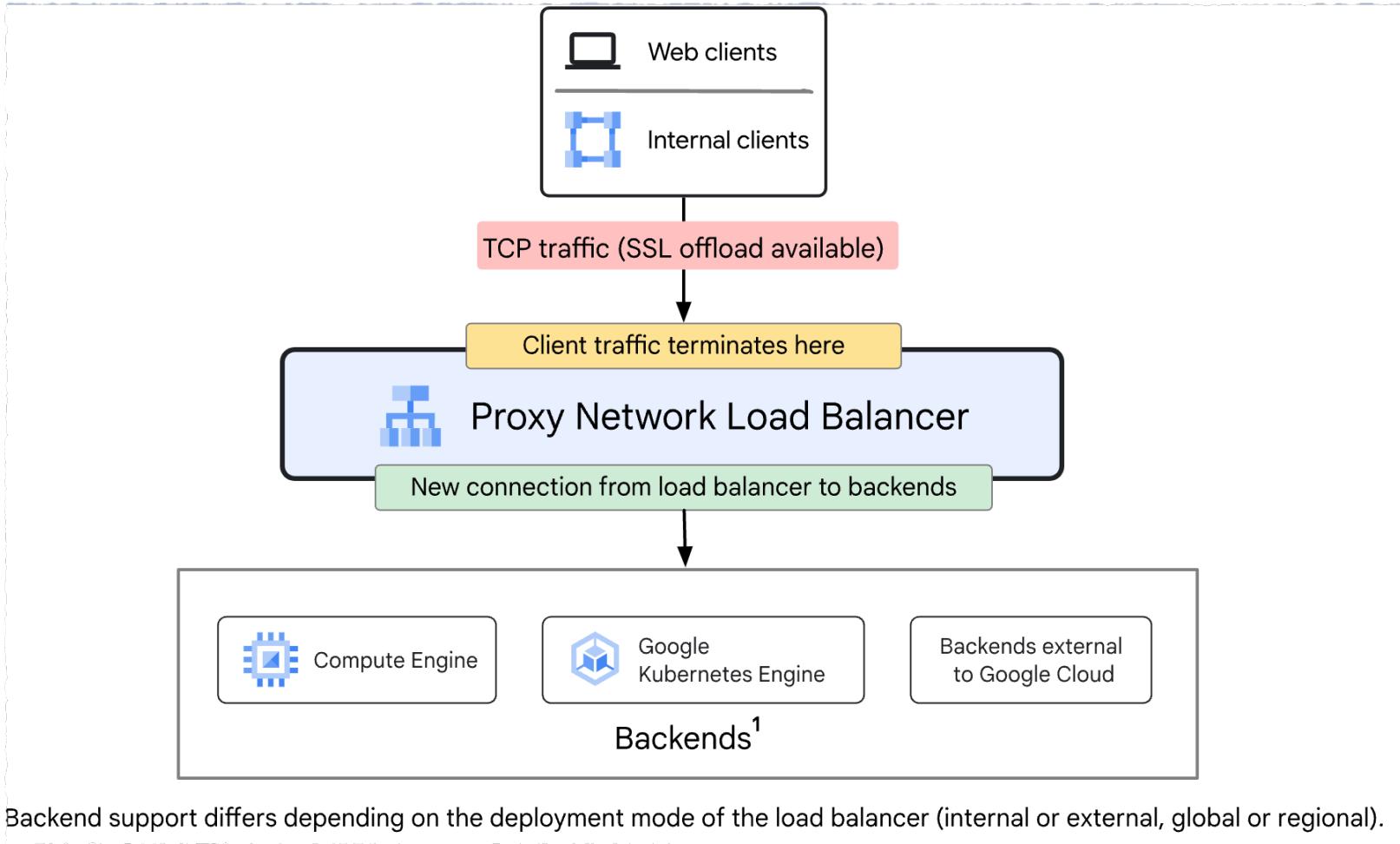
Application Load Balancer Architecture



Network Load Balancers

- Network Load Balancers are Layer 4 load balancers that can handle TCP, UDP, or other IP protocol traffic.
- These load balancers are available as either proxy load balancers or passthrough load balancers.
- Choose a proxy Network Load Balancer if you want to configure a reverse proxy load balancer with support for advanced traffic controls and backends on-premises and in other cloud environments.
- Choose a passthrough Network Load Balancer if you want to preserve the source IP address of the client packets, you prefer direct server return for responses, or you want to handle a variety of IP protocols such as TCP, UDP, ESP, GRE, ICMP, and ICMPv6 .

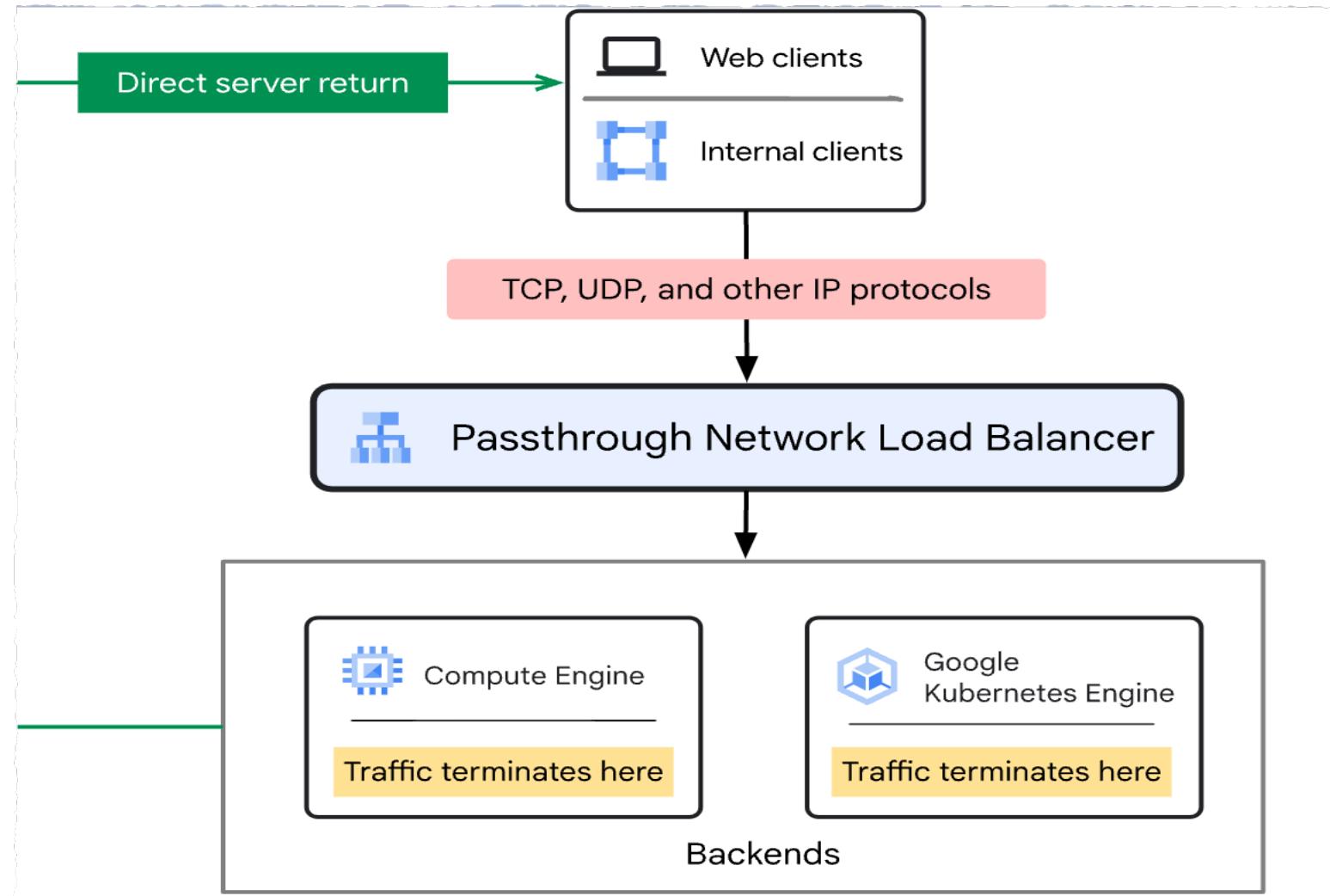
Proxy Network Load Balancers



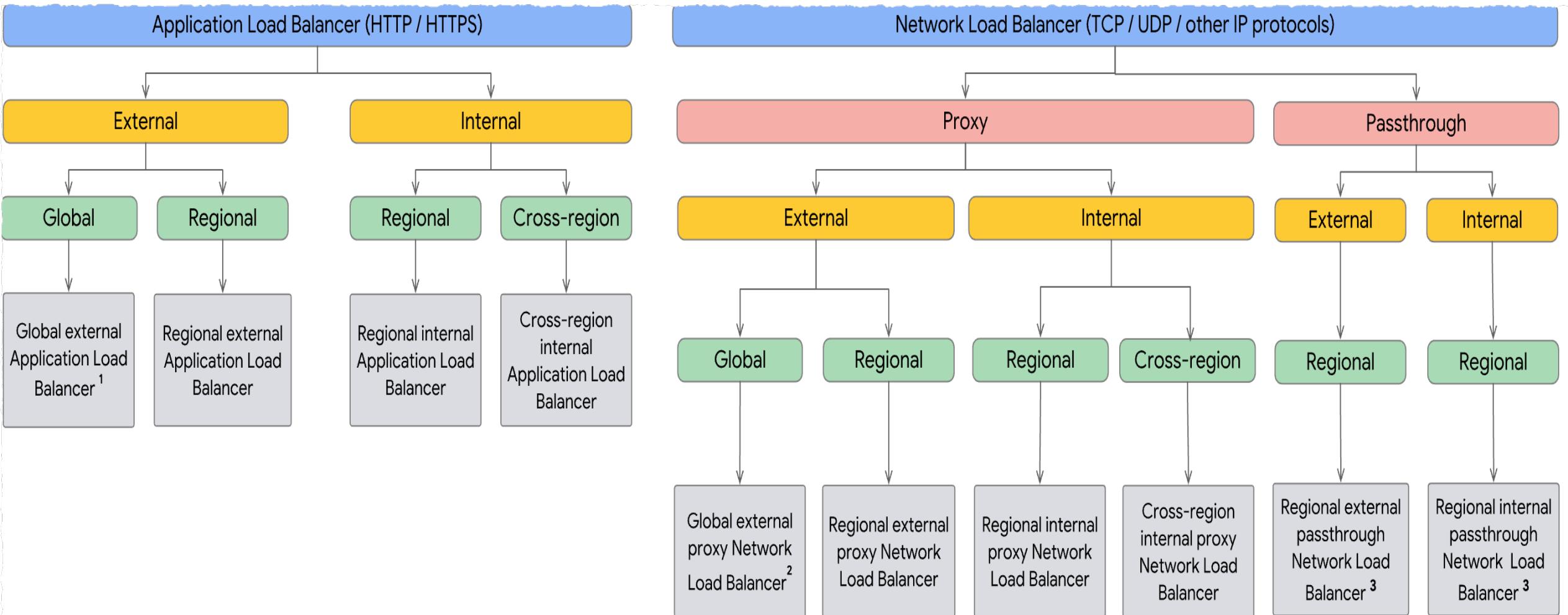


WHERE IS THE REAL IP?

Passthrough Network Load Balancers



Choose a load balancer



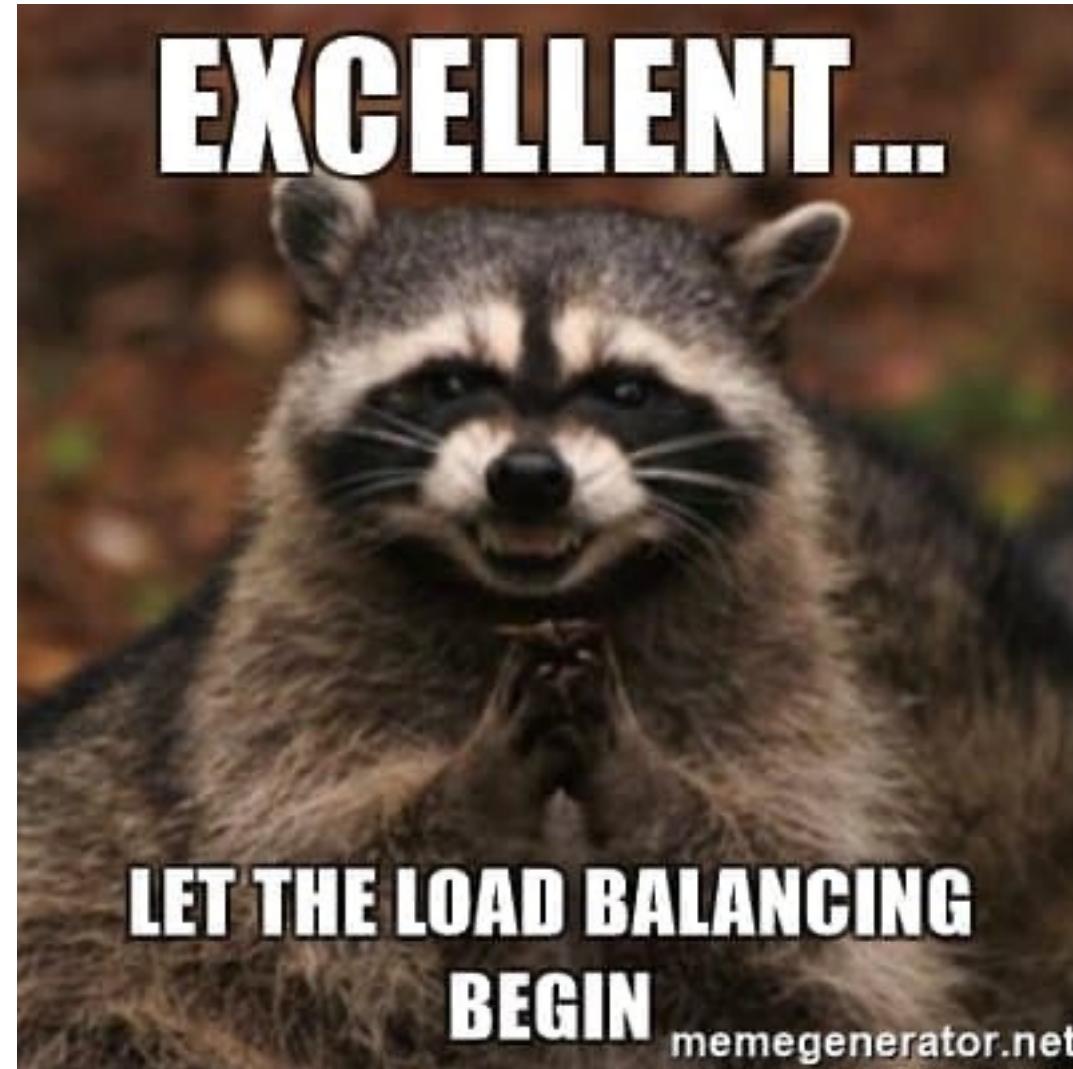
Summary of types of Google Cloud load balancers

Load balancer	Deployment mode	Traffic type	Network service tier	Load-balancing scheme
Application Load Balancers	Global external	HTTP or HTTPS	Premium Tier	EXTERNAL_MANAGED
	Regional external	HTTP or HTTPS	Premium or Standard Tier	EXTERNAL_MANAGED
	Classic	HTTP or HTTPS	Global in Premium Tier Regional in Standard Tier	EXTERNAL
	Regional internal	HTTP or HTTPS	Premium Tier	INTERNAL_MANAGED
	Cross-region internal	HTTP or HTTPS	Premium Tier	INTERNAL_MANAGED
Proxy Network Load Balancers	Global external	TCP with optional SSL offload	Premium Tier	EXTERNAL_MANAGED
	Regional external	TCP	Premium or Standard Tier	EXTERNAL_MANAGED
	Classic	TCP with optional SSL offload	Global in Premium Tier Regional in Standard Tier	EXTERNAL
	Regional internal	TCP without SSL offload	Premium Tier	INTERNAL_MANAGED
	Cross-region internal	TCP without SSL offload	Premium Tier	INTERNAL_MANAGED
Passthrough Network Load Balancers	External Always regional	TCP, UDP, ESP, GRE, ICMP, and ICMPv6	Premium or Standard Tier	EXTERNAL
	Internal Always regional	TCP, UDP, ICMP, ICMPv6, SCTP, ESP, AH, and GRE	Premium Tier	INTERNAL

Connection Draining

- Connection draining is a process that ensures that existing, in-progress requests are given time to complete when a VM is removed from an instance group or when an endpoint is removed from network endpoint groups (NEGs) that are zonal in scope.
- Connection draining begins whenever you do the following:
 - You manually remove a VM from an instance group.
 - You remove an instance from a managed instance group by performing a `resize()`, `deleteInstances()`, `recreateInstances()`, or `abandonInstances()` call.
 - An instance group is removed from a backend service. This isn't supported for internal passthrough Network Load Balancers.
 - Google Cloud deletes an instance as part of autoscaling.
 - You perform an update to the managed instance group using the Managed Instance Group Updater.

EXCELLENT...



**LET THE LOAD BALANCING
BEGIN**

memegenerator.net

Additional Resources

See Lecture Page

CI/CD

Tejas Parikh (t.parikh@northeastern.edu)

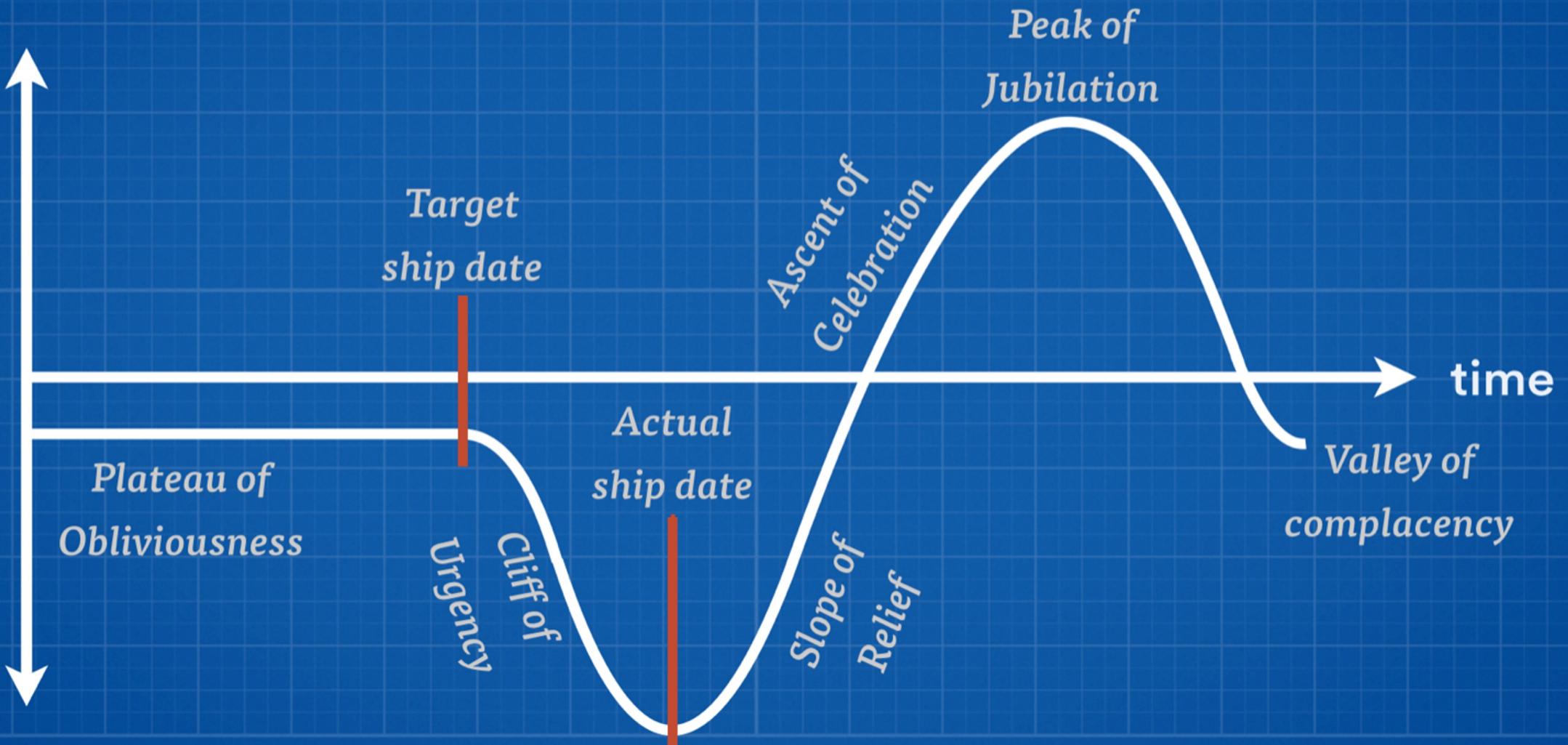
CSYE 6225

Northeastern University

Challenges with Traditional Release Method

- Servers must be set up by IT (sometimes manually)
- Third-party software such as application server, etc. must be installed.
- The software artifacts such as EAR or WAR must be copied to the production host.
- Application configuration must be copied or created.
- Finally any reference data needed must be copied over.
- As you can see there are lot of places where things can go wrong.
- With this process, the day of a software release tends to be a tense one.

Emotional cycle of manual delivery



What is Continuous Deployment?

- Continuous Deployment is a software development practice in which every code change goes through the entire pipeline and is put into production, automatically, resulting in many production deployments every day.
- With Continuous Delivery your software is always release-ready, yet the timing of when to push it into production is a business decision, and so the final deployment is a manual step.
- With Continuous Deployment, any updated working version of the application is automatically pushed to production.
- Continuous Deployment mandates Continuous Delivery, but the opposite is not required.

CONTINUOUS DELIVERY

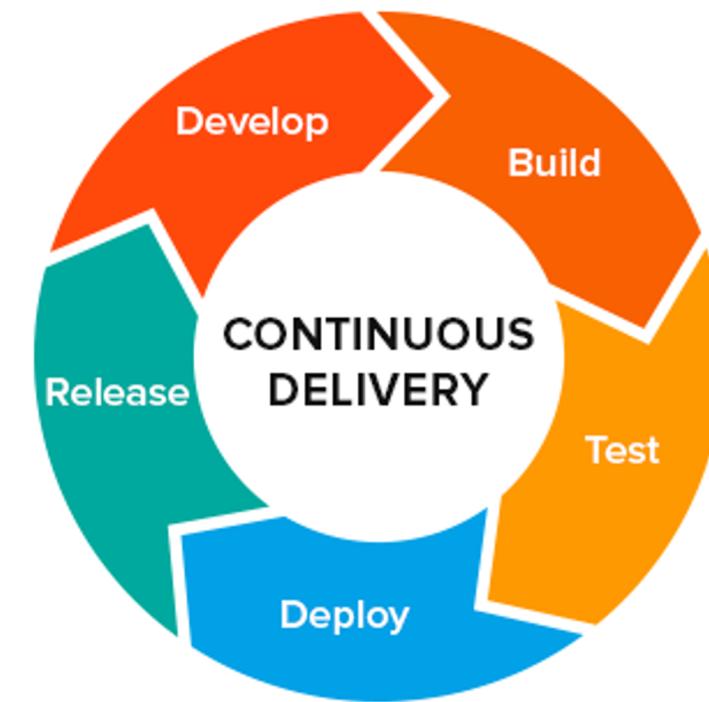


CONTINUOUS DEPLOYMENT



Continuous Delivery

- Continuous delivery is a DevOps software development practice where code changes are automatically built, tested, and prepared for a release to production.
- It expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage.
- When continuous delivery is implemented properly, developers will always have a deployment-ready build artifact that has passed through a standardized test process.



Difference between environments: Configuration

Continuous Delivery (contd.)

- With continuous delivery, every code change is built, tested, and then pushed to a non-production testing or staging environment.
- There can be multiple, parallel test stages before a production deployment.
- In the last step, the developer approves the update to production when they are ready.



(c) 2015 www.goerp.nl

Enabling Continuous Delivery

- Automate
 - The build, deploy, test, and release process must be automated so that it is repeatable.
- Frequent
 - Releases must be frequent.
 - The delta between releases will be small.
 - This significantly reduces the risk associated with releasing and makes it much easier to roll back.

Continuous Deployment

In continuous deployment push to production happens automatically without explicit approval.

Additional Resources

See Lecture Page

Cloud & Application (In)security

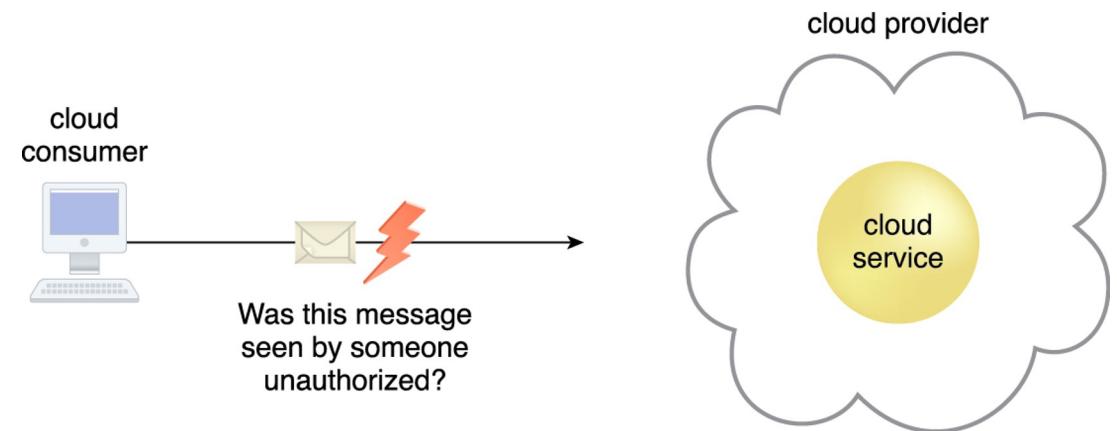
Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University

Basic Terms and Concepts

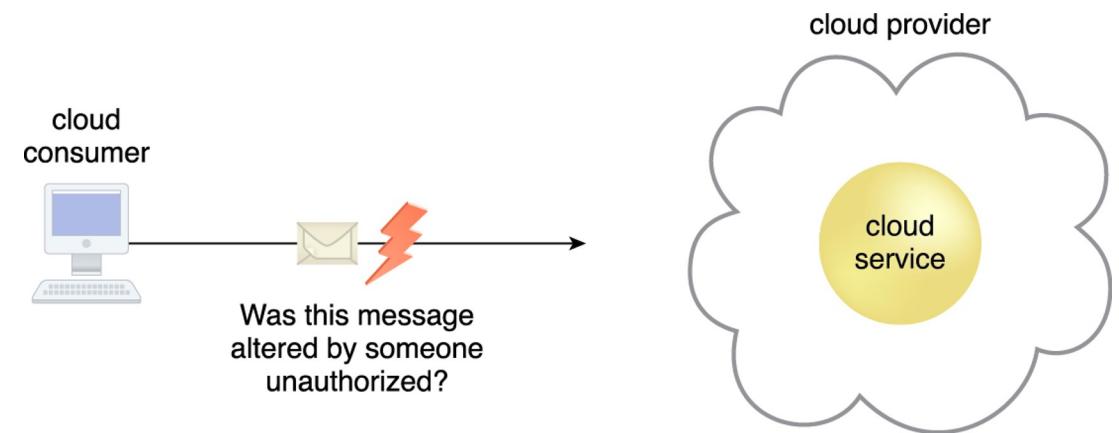
Confidentiality

Only authorized party can get access to data.



Integrity

Guarantee the user that the data it transmitted matches the data received by the cloud service.



Authenticity

Is the data coming from an authorized source?

Availability

Service should be accessible and usable during a specified time period.
For a cloud service, the time period is 24/7/365.

Threat

In computer security a threat is a possible danger that might exploit a vulnerability to breach security and therefore cause possible harm.

Vulnerability

A vulnerability is a weakness that can be exploited.

Attack Vectors

- An attack vector is a path or means by which a hacker can gain access to a computer or network server in order to deliver a payload or malicious outcome.
- Attack vectors enable hackers to exploit system vulnerabilities, including the human element.

Risk

Risk is the possibility of loss or harm arising from performing an activity

Security Controls

Security controls are countermeasures used to prevent or respond to security threats and to reduce or avoid risk.

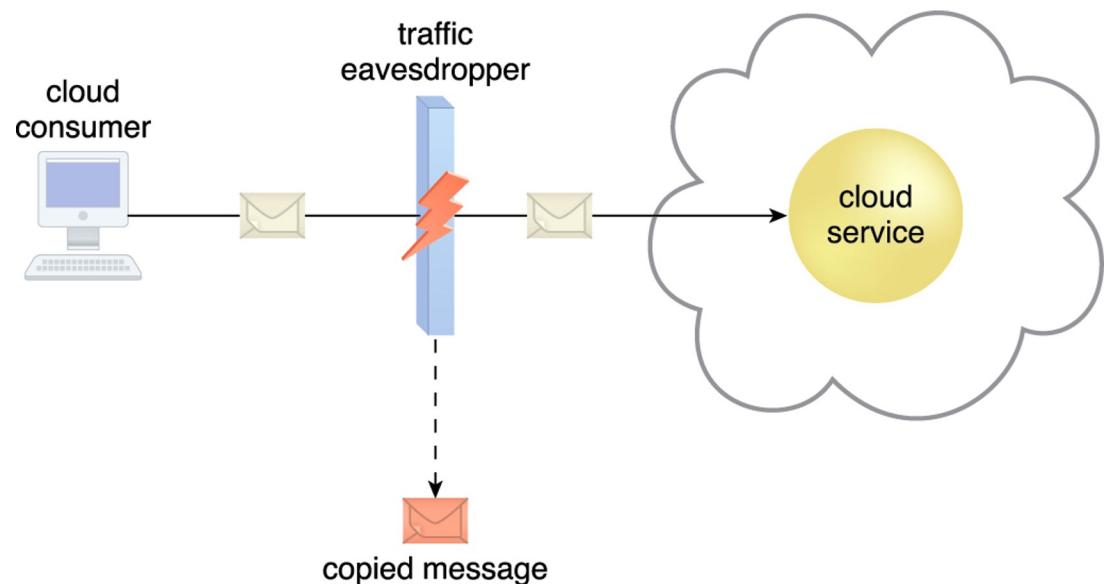
Threat Agent

A threat agent is an entity that poses a threat because it is capable of carrying out an attack.

Cloud Security Threats

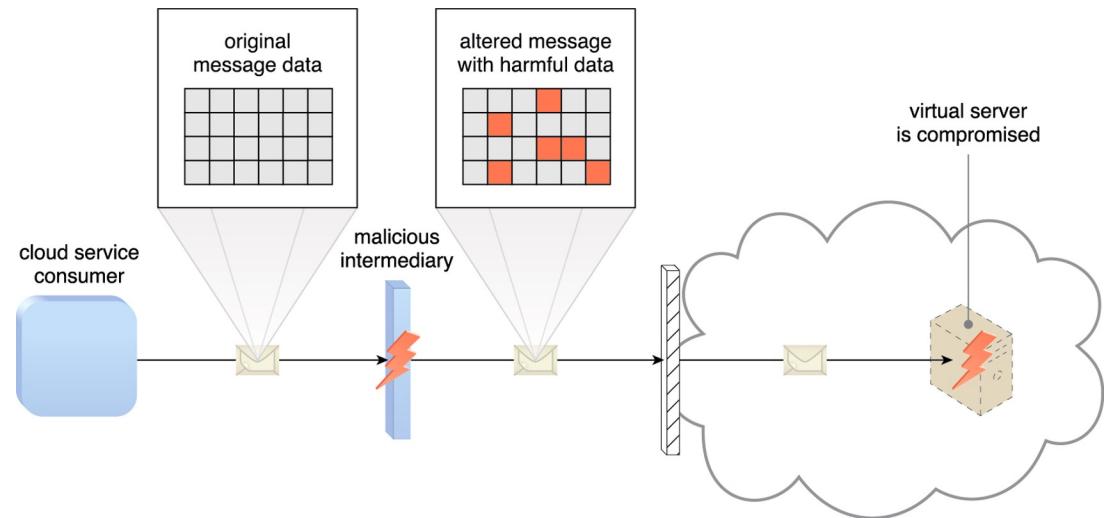
Traffic Eavesdropping

- Traffic eavesdropping occurs when data being transferred to or within a cloud service is passively intercepted for illegitimate information gathering purposes
- This attack's aim is to directly compromise the confidentiality of the data and, possibly, the confidentiality of the relationship between the cloud consumer and cloud provider.
- Because of the passive nature of the attack, it can more easily go undetected for extended periods of time.



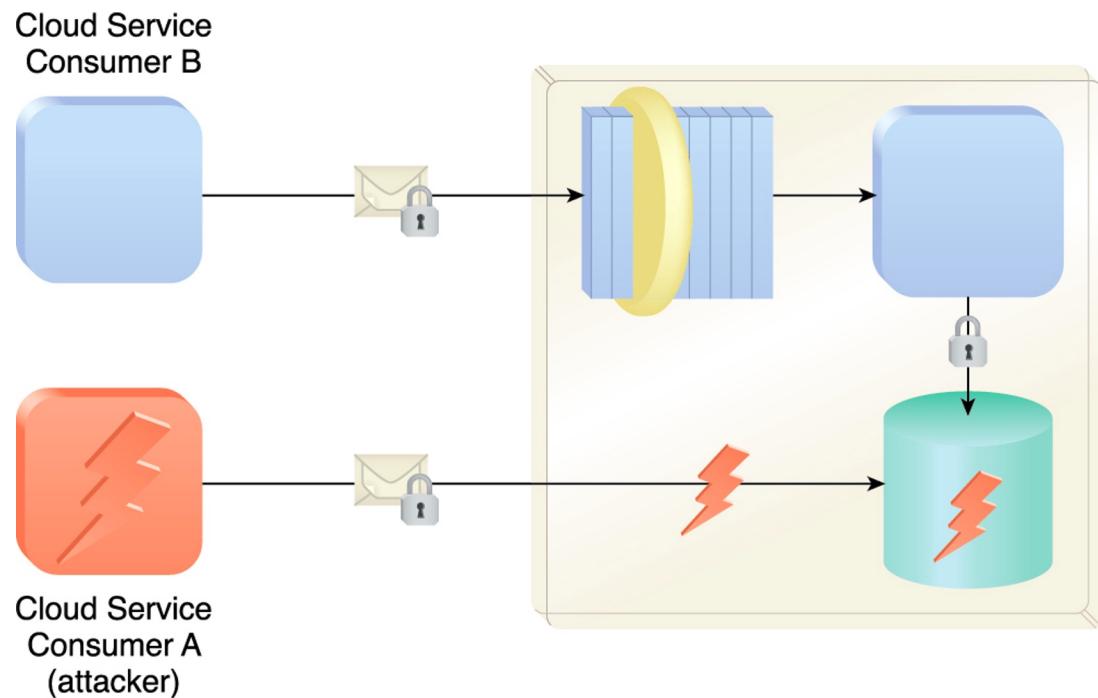
Malicious Intermediary

- The malicious intermediary threat arises when messages are intercepted and altered by a malicious service agent, thereby potentially compromising the message's confidentiality and/ or integrity.
- It may also insert harmful data into the message before forwarding it to its destination.



Insufficient Authorization

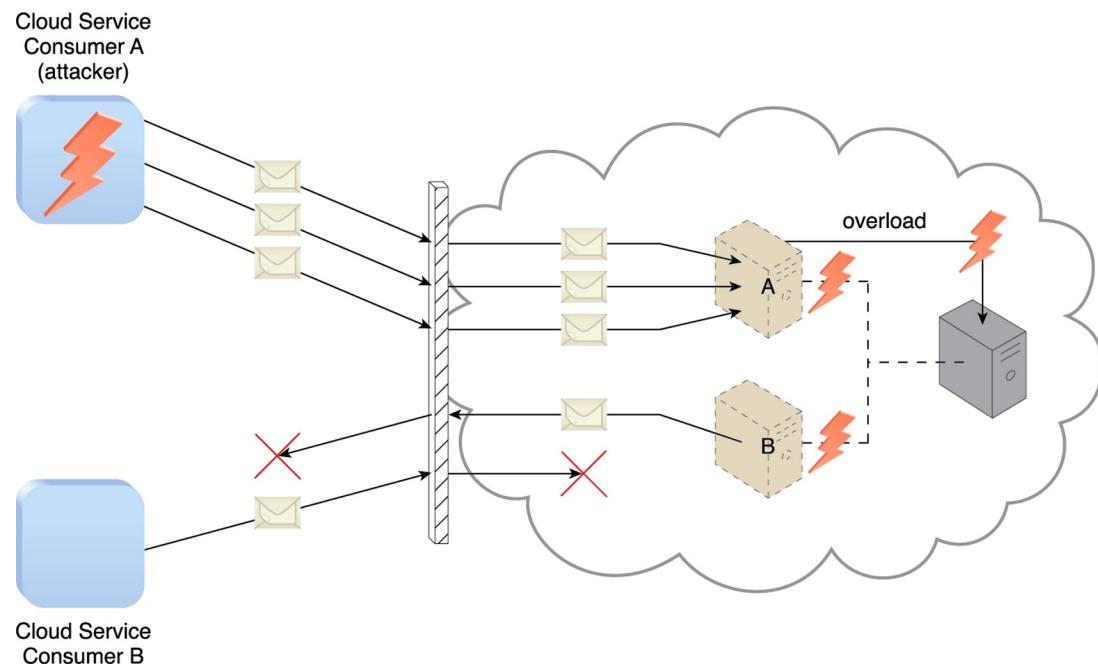
- The insufficient authorization attack occurs when access is granted to an attacker erroneously or too broadly, resulting in the attacker getting access to IT resources that are normally protected.
- This is often a result of the attacker gaining direct access to IT resources that were implemented under the assumption that they would only be accessed by trusted consumer programs.



Denial of Service

The objective of the denial of service (DoS) attack is to overload resources to the point where they cannot function properly. This form of attack is commonly launched in one of the following ways:

- The workload is artificially increased with imitation messages or repeated communication requests.
 - The network is overloaded with traffic to reduce its responsiveness and cripple its performance.
 - Multiple requests are sent, each of which is designed to consume excessive memory and processing resources.
- Successful DoS attacks produce server degradation and/ or failure.



Flawed Implementations

- The substandard design, implementation, or configuration of service and its deployments can have undesirable consequences, beyond runtime exceptions and failures.
- If the cloud provider's software and/ or hardware have inherent security flaws or operational weaknesses, attackers can exploit these vulnerabilities to impair the integrity, confidentiality, and/ or availability of cloud provider IT resources and cloud consumer IT resources hosted by the cloud provider.

Cloud Security

Cloud Security Is A Shared Responsibility

- Security and Compliance is a shared responsibility between AWS and the customer.
- This shared model can help relieve the customer's operational burden as AWS operates, manages and controls the components from the host operating system and virtualization layer down to the physical security of the facilities in which the service operates.
- The customer assumes responsibility and management of the guest operating system (including updates and security patches), other associated application software as well as the configuration of the AWS provided security group firewall.
- Customers should carefully consider the services they choose as their responsibilities vary depending on the services used, the integration of those services into their IT environment, and applicable laws and regulations.

CUSTOMER

RESPONSIBILITY FOR
SECURITY 'IN' THE CLOUD

CUSTOMER DATA

PLATFORM, APPLICATIONS, IDENTITY & ACCESS MANAGEMENT

OPERATING SYSTEM, NETWORK & FIREWALL CONFIGURATION

CLIENT-SIDE DATA
ENCRYPTION & DATA INTEGRITY
AUTHENTICATION

SERVER-SIDE ENCRYPTION
(FILE SYSTEM AND/OR DATA)

NETWORKING TRAFFIC
PROTECTION (ENCRYPTION,
INTEGRITY, IDENTITY)

SOFTWARE

COMPUTE

STORAGE

DATABASE

NETWORKING

HARDWARE/AWS GLOBAL INFRASTRUCTURE

REGIONS

AVAILABILITY ZONES

EDGE LOCATIONS

RESPONSIBILITY FOR
SECURITY 'OF' THE CLOUD

AWS

Security of the Cloud – Cloud Provider Responsibility

- Cloud Platform provider is responsible for protecting the infrastructure that runs all of the services offered in their Cloud.
- This infrastructure is composed of the hardware, software, networking, and facilities that run the Cloud services.

Security in the Cloud – Cloud User Responsibility

- Customer responsibility will be determined by the Cloud services that a customer selects.
- This determines the amount of configuration work the customer must perform as part of their security responsibilities.
- For example, a service such as Amazon Elastic Compute Cloud (Amazon EC2) is categorized as Infrastructure as a Service (IaaS) and, as such, requires the customer to perform all of the necessary security configuration and management tasks. Customers that deploy an Amazon EC2 instance are responsible for management of the guest operating system (including updates and security patches), any application software or utilities installed by the customer on the instances, and the configuration of the AWS-provided firewall (called a security group) on each instance.
- For abstracted services, such as Amazon S3 and Amazon DynamoDB, AWS operates the infrastructure layer, the operating system, and platforms, and customers access the endpoints to store and retrieve data.
- Customers are responsible for managing their data (including encryption options), classifying their assets, and using IAM tools to apply the appropriate permissions.

Security Mechanisms

Encryption

- Data, by default, is coded in a readable format known as plaintext. When transmitted over a network, plaintext is vulnerable to unauthorized and potentially malicious access.
- The encryption mechanism is a digital coding system dedicated to preserving the confidentiality and integrity of data. It is used for encoding plaintext data into a protected and unreadable format.
- Encryption can help counter the traffic eavesdropping, malicious intermediary, insufficient authorization, and overlapping trust boundaries security threats.
- There are two common forms of encryption known as **symmetric** encryption and **asymmetric** encryption.

Symmetric Encryption

- Symmetric encryption uses the same key for both encryption and decryption, both of which are performed by authorized parties that use the one shared key.
- Also known as secret key cryptography, messages that are encrypted with a specific key can be decrypted by only that same key.
- Parties that rightfully decrypt the data are provided with evidence that the original encryption was performed by parties that rightfully possess the key.
- A basic authentication check is always performed, because only authorized parties that own the key can create messages. This maintains and verifies data confidentiality.
- Note that symmetrical encryption does not have the characteristic of non-repudiation, since determining exactly which party performed the message encryption or decryption is not possible if more than one party is in possession of the key.

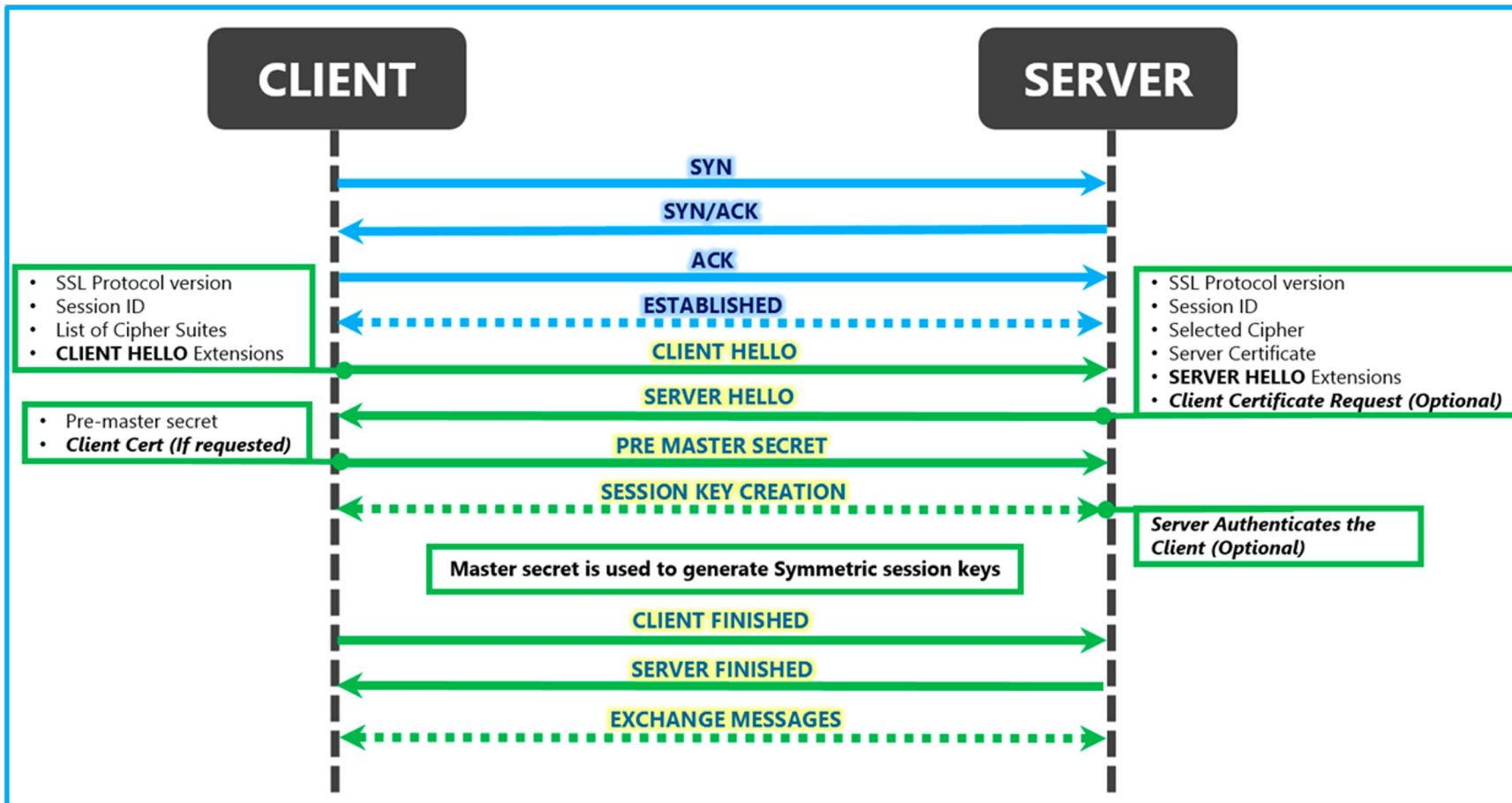
Asymmetric Encryption

- Asymmetric encryption relies on the use of two different keys, namely a private key and a public key.
- With asymmetric encryption (which is also referred to as public key cryptography), the private key is known only to its owner while the public key is commonly available.
- A document that was encrypted with a private key can only be correctly decrypted with the corresponding public key.
- Conversely, a document that was encrypted with a public key can be decrypted only using its private key counterpart.
- As a result of two different keys being used instead of just the one, asymmetric encryption is almost always computationally slower than symmetric encryption.

Securing Web-based Data Transmission

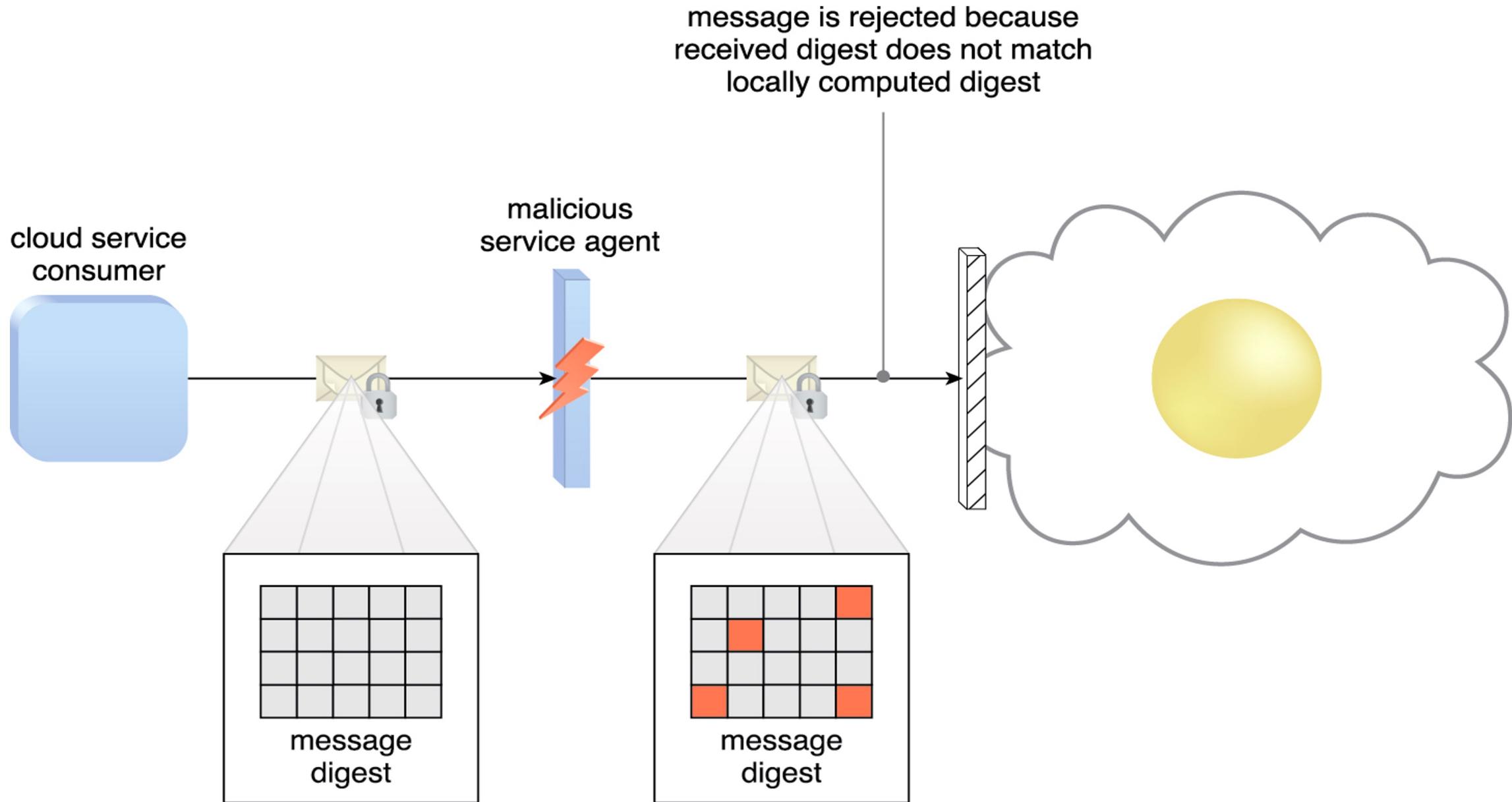
- The encryption mechanism, when used to secure Web-based data transmissions, is most commonly applied via HTTPS, which refers to the use of SSL/ TLS as an underlying encryption protocol for HTTP. TLS (transport layer security) is the successor to the SSL (secure sockets layer) technology.
- Because asymmetric encryption is usually more time-consuming than symmetric encryption, TLS uses the former only for its key exchange method.
- TLS systems then switch to symmetric encryption once the keys have been exchanged.
- Most TLS implementations primarily support RSA as the chief asymmetrical encryption cipher, while ciphers such as Triple-DES, and AES are supported for symmetrical encryption.

SSL Handshake



Hashing

- The hashing mechanism is used when a one-way, non-reversible form of data protection is required.
- Hashing can be used to derive a message digest from a message, which is often of a fixed length and smaller than the original message.
- The message sender can then utilize the hashing mechanism to attach the message digest to the message.
- The recipient applies the same hash function to the message to verify that the produced message digest is identical to the one that accompanied the message.
- Any alteration to the original data results in an entirely different message digest and clearly indicates that tampering has occurred.



Digital Signature

- The digital signature mechanism is a means of providing data authenticity and integrity through authentication and non-repudiation.
- A message is assigned a digital signature prior to transmission, which is then rendered invalid if the message experiences any subsequent, unauthorized modifications.
- A digital signature provides evidence that the message received is the same as the one created by its rightful sender.
- Both hashing and asymmetrical encryption are involved in the creation of a digital signature, which essentially exists as a message digest that was encrypted by a private key and appended to the original message.
- The recipient verifies the signature validity and uses the corresponding public key to decrypt the digital signature, which produces the message digest.
- The hashing mechanism can also be applied to the original message to produce this message digest. Identical results from the two different processes indicate that the message maintained its integrity.

Hardened Server Images

- Hardening is the process of stripping unnecessary software from a system to limit potential vulnerabilities that can be exploited by attackers.
- Removing redundant programs, closing unnecessary server ports, and disabling unused services, internal root accounts, and guest access are all examples of hardening.



Application Security

The Core Security Problem: Users Can Submit Arbitrary Input

The application must assume that all input is potentially malicious, and must take steps to ensure that attackers cannot use crafted input to compromise the application by interfering with its logic and behavior and gaining unauthorized access to its data and functionality.

User Input

- Users can interfere with any piece of data transmitted between the client and the server, including request parameters, cookies, and HTTP headers.
- Any security controls implemented on the client side, such as input validation checks, can be easily circumvented.
- Users can send requests in any sequence, and can submit parameters at a different stage than the application expects, more than once, or not at all.
- Any assumption which developers make about how users will interact with the application may be violated.
- Users are not restricted to using only a web browser to access the application.
- There are numerous widely available tools that operate alongside, or independently of, a browser, to help attack web applications. These tools can make requests that no browser would ordinarily make, and can generate huge numbers of requests quickly to find and exploit problems.

Attack Vectors

The majority of attacks against web applications involve sending input to the server which is crafted to cause some event that was not expected or desired by the application's designer.

Attack Vector Examples

- Changing the price of a product transmitted in a hidden HTML form field, to fraudulently purchase the product for a cheaper amount.
- Modifying a session token transmitted in an HTTP cookie, to hijack the session of another authenticated user.
- Removing certain parameters that are normally submitted, to exploit a logic flaw in the application's processing.
- Altering some input that will be processed by a back-end database, to inject a malicious database query and so access sensitive data.

Core Defense Mechanisms

Authentication

Authenticating a user involves establishing that the user is in fact who he claims to be.

Session Management

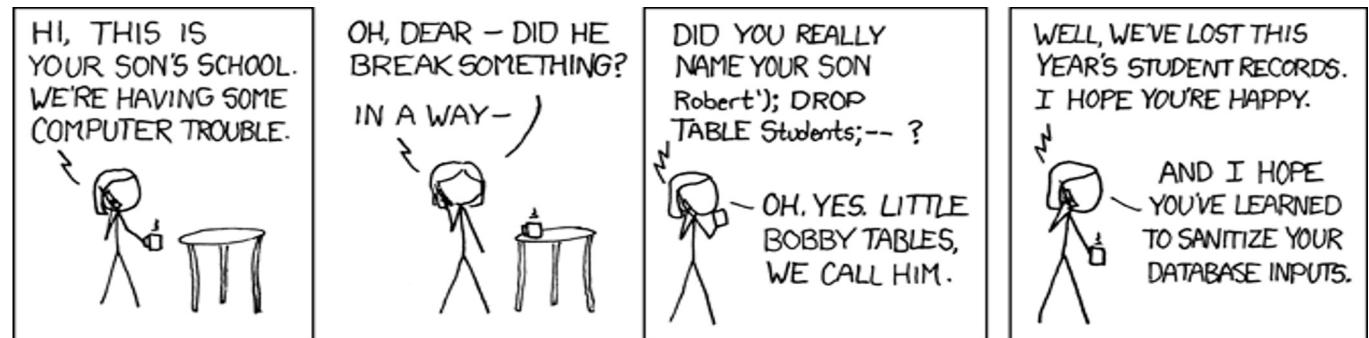
- After successfully logging in to the application, the user will access various pages and functions, making a series of HTTP requests from their browser.
- At the same time, the application will be receiving countless other requests from different users, some of whom are authenticated and some of whom are anonymous.
- In order to enforce effective access control, the application needs a way of identifying and processing the series of requests that originate from each unique user.
- Virtually all web applications meet this requirement by creating a session for each user and issuing the user a token that identifies the session.
- In terms of attack surface, the session management mechanism is highly dependent on the security of its tokens, and the majority of attacks against it seek to compromise the tokens issued to other users.

Access Control

- The access control mechanism usually needs to implement some fine-grained logic, with different considerations being relevant to different areas of the application and different types of functionality.
- An application might support numerous different user roles, each involving different combinations of specific privileges.
- Individual users may be permitted to access a subset of the total data held within the application.
- Because of the complex nature of typical access control requirements, this mechanism is a frequent source of security vulnerabilities that enable an attacker to gain unauthorized access to data and functionality.

Handling User Input

- Varieties of Input
- Input Sanitization
- Safe Data Handling



Handling Hackers

Mapping the Application

- Discovering hidden content
- Discovering hidden parameters
- Identifying entry points for user input
- Identifying server-side technologies
- Identifying server-side functionality

Bypassing Client Side Controls

- Hidden form fields
- URL Parameters
- Referrer Header
- Data length input
- Script based validation

Attacking Authentication

- Bad passwords
- Brute-Forcible Login
- Verbose Failure Messages
- Password Change Functionality
- “Remember Me” Functionality
- Storing credentials in plain-text in database
- No “salt” used

Code Injection into Interpreted Languages

- An interpreted language is one whose execution involves a runtime component that interprets the code of the language and carries out the instructions that it contains.
- Because of the way that interpreted languages are executed, there arises a family of vulnerabilities known as code injection.
- In any useful application, user-supplied data will be received, manipulated, and acted upon.
- The code that is processed by the interpreter will, therefore, comprise a mix of the instructions written by the programmer and the data supplied by the user.

Example: Code Injection

- [https://en.wikipedia.org/wiki/Code_injection#Format specifier_injection](https://en.wikipedia.org/wiki/Code_injection#Format_specifier_injection)
- The programmer may mistakenly write `printf(buffer)` instead of `printf ("%s", buffer)`.
 - The first version interprets buffer as a format string and parses any formatting instructions it may contain.
 - The second version simply prints a string to the screen, as the programmer intended.
- Consider the following short C program that has a local variable char array password which holds a password; the program asks the user for an integer and a string, then echoes out the user-provided string.
- If the user input is filled with a list of format specifiers such as `%s%s%s%s%s%s`, then `printf()` will start reading from the [stack](#). Eventually, one of the `%s` format specifier will access the address of `password`, which is on the stack, and print `Password1` to the screen.

```
char user_input[100];
int int_in;
char password[10] = "Password1";

printf("Enter an integer\n");
scanf("%d", &int_in);
printf("Please enter a string\n");
fgets(user_input, sizeof(user_input), stdin);

printf(user_input); //Safe version is: printf("%s", user_input);
printf("\n");

return 0;
```

Injecting Code in SQL

- SQL injection is the elder statesman of code injection attacks, being still one of the more prevalent vulnerabilities in the wild, and frequently one of the most devastating.
- SQL injection can enable an anonymous attacker to read and modify all data stored within the database, and even take full control of the server on which the database is running.



SQL Injection Example

The following line of code illustrates this vulnerability:

```
statement = "SELECT * FROM users WHERE name = '" + userName + "';"
```

This SQL code is designed to pull up the records of the specified username from its table of users. However, if the "userName" variable is crafted in a specific way by a malicious user, the SQL statement may do more than the code author intended. For example, setting the "userName" variable as:

```
' OR '1'='1
```

or using comments to even block the rest of the query (there are three types of SQL comments^[13]). All three lines have a space at the end:

```
' OR '1'='1' --
' OR '1'='1' ({ 
' OR '1'='1' /*
```

renders one of the following SQL statements by the parent language:

```
SELECT * FROM users WHERE name = '' OR '1'='1';
```

```
SELECT * FROM users WHERE name = '' OR '1'='1' -- ';
```

Cross-site scripting

- Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications.
- XSS enables attackers to inject client-side scripts into web pages viewed by other users.
- A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy.

`http://bobssite.org?q=<script%20type='text/javascript'>alert('xss');</script>`

XSS (Cross Site Scripting) Prevention Cheat Sheet

[https://www.owasp.org/index.php/XSS \(Cross Site Scripting\) Prevention Cheat Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)

Cross-site request forgery

- Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.
- CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.
- With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing.
- If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth.
- If the victim is an administrative account, CSRF can compromise the entire web application.

Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet

[https://www.owasp.org/index.php/Cross-Site Request Forgery \(CSRF\) Prevention Cheat Sheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet)

Additional Resources

See Lecture Page

Architecting for the Cloud

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225
Northeastern University

Building & Deploying Applications on Cloud

Lift & Shift

- Migrating application to cloud without significant changes.
- Gain benefits of a secure and cost-efficient infrastructure.

Maximize Cloud Investments

- Make the most of the elasticity and agility that are possible with cloud computing, engineers have to evolve their architectures to take advantage of cloud capabilities.
- For new applications, cloud-specific IT architecture patterns can help drive efficiency and scalability.
- Whether you are rearchitecting the applications that currently run in your on-premises environment to run on AWS, or designing cloud-native applications, you must consider the differences between traditional environments and cloud computing environments.

Differences Between Traditional and Cloud Computing Environments

On-Premises vs. Cloud

Cloud computing differs from a traditional, on-premises environment in many ways, including

- Flexible
- global and scalable capacity
- managed services
- built-in security
- options for cost optimization
- and various operating models.

IT Assets as Provisioned Resources

- In a traditional computing environment, you provision capacity based on an estimate of a theoretical maximum peak. This can result in periods where expensive resources are sitting idle or occasions of insufficient capacity.
- With cloud computing, you can access as much or as little capacity as you need and dynamically scale to meet actual demand, while only paying for what you use.
- On cloud, servers, databases, storage, and higher-level application components can be instantiated within seconds.
- You can treat these as temporary resources, free from the inflexibility and constraints of a fixed and finite IT infrastructure.
- This resets the way you approach change management, testing, reliability, and capacity planning.
- This change in approach encourages experimentation by introducing the ability in processes to fail fast and iterate quickly.

Global, Available, and Scalable Capacity

- Using the global infrastructure of cloud platform provider, you can deploy your application to the Region that best meets your requirements (e.g., proximity to your end users, compliance, data residency constraints, and cost).
- For global applications, you can reduce latency to end users around the world by using content delivery network (CDN).
- This also makes it much easier to operate production applications and databases across multiple data centers to achieve high availability and fault tolerance.
- The global infrastructure of cloud platform provider and the ability to provision capacity as needed let you think differently about your infrastructure as the demands on your applications and the breadth of your services expand.

Built-in Security

- In traditional IT environments, infrastructure security auditing can be a periodic and manual process.
- In contrast cloud provides governance capabilities that enable continuous monitoring of configuration changes to your IT resources.
- Since cloud resources are programmable using tools and APIs, you can formalize and embed your security policy within the design of your infrastructure.
- With the ability to spin up temporary environments, security testing can now become part of your continuous delivery pipeline.
- Finally, you can leverage a variety of native security and encryption features that can help you achieve higher levels of data protection and compliance.

Architecting for Cost

- Traditional cost management of on-premises solutions is not typically tightly coupled to the provision of services.
- When you provision a cloud computing environment, optimizing for cost is a fundamental design tenant for architects.
- When selecting a solution, you should not only focus on the functional architecture and feature set but on the cost profile of the solutions you select.

Design Principles

Scalability

- Systems that are expected to grow over time need to be built on top of a scalable architecture.
- Such an architecture can support growth in users, traffic, or data size with no drop-in performance.
- It should provide that scale in a linear manner where adding extra resources results in at least a proportional increase in ability to serve additional load.
- Growth should introduce economies of scale, and cost should follow the same dimension that generates business value out of that system.
- While cloud computing provides virtually unlimited on-demand capacity, your design needs to be able to take advantage of those resources seamlessly.
- There are generally two ways to scale an IT architecture: vertically and horizontally.

Scaling Vertically

- Scaling vertically takes place through an increase in the specifications of an individual resource, such as upgrading a server with a larger hard drive or a faster CPU.
- With Amazon EC2, you can stop an instance and resize it to an instance type that has more RAM, CPU, I/O, or networking capabilities.
- This way of scaling can eventually reach a limit, and it is not always a cost-efficient or highly available approach.
- However, it is very easy to implement and can be sufficient for many use cases especially in the short term.

Scaling Horizontally

- Scaling horizontally takes place through an increase in the number of resources, such as adding more hard drives to a storage array or adding more servers to support an application.
- This is a great way to build internet-scale applications that leverage the elasticity of cloud computing.
- Not all architectures are designed to distribute their workload to multiple resources.

Stateless Applications

- When users or services interact with an application, they will often perform a series of interactions that form a session.
- A session is unique data for users that persists between requests while they use the application.
- A stateless application is an application that does not need knowledge of previous interactions and does not store session information.
- Stateless applications can scale horizontally because any of the available compute resources can service any request.
- Without stored session data, you can simply add more compute resources as needed.
- When that capacity is no longer required, you can safely terminate those individual resources, after running tasks have been drained.
- Those resources do not need to be aware of the presence of their peers—all that is required is a way to distribute the workload to them.

Distribute Load to Multiple Nodes

- To distribute the workload to multiple nodes in your environment, you can choose either a push or a pull model.
- With a push model, you can use Elastic Load Balancing (ELB) to distribute a workload. ELB routes incoming application requests across multiple EC2 instances. When routing traffic, a Network Load Balancer operates at layer 4 of the Open Systems Interconnection (OSI) model to handle millions of requests per second. With the adoption of container-based services, you can also use an Application Load Balancer. An Application Load Balancer provides Layer 7 of the OSI model and supports content- based routing of requests based on application traffic.
- Alternatively, you can use Amazon Route 53 to implement a DNS round robin. In this case, DNS responses return an IP address from a list of valid hosts in a round-robin fashion. While easy to implement, this approach does not always work well with the elasticity of cloud computing. This is because even if you can set low time to live (TTL) values for your DNS records, caching DNS resolvers are outside the control of Amazon Route 53 and might not always respect your settings.
- Instead of a load balancing solution, you can implement a pull model for asynchronous, event-driven workloads. In a pull model, tasks that need to be performed or data that needs to be processed can be stored as messages in a queue or as a streaming data solution.

Stateless Components

- In practice, most applications maintain some kind of state information.
- Store session information in database or external cache.
- Store files in Object Storage (S3) or Network File Share (EFS).

Stateful Components

- Inevitably, there will be layers of your architecture that you won't turn into stateless components. By definition, databases are stateful. In addition, many legacy applications were designed to run on a single server by relying on local compute resources. Other use cases might require client devices to maintain a connection to a specific server for prolonged periods.
- For example, real-time multiplayer gaming must offer multiple players a consistent view of the game world with very low latency. This is much simpler to achieve in a non-distributed implementation where participants are connected to the same server.
- You might still be able to scale those components horizontally by distributing the load to multiple nodes with session affinity. In this model, you bind all the transactions of a session to a specific compute resource. But this model does have some limitations. Existing sessions do not directly benefit from the introduction of newly launched compute nodes. More importantly, session affinity cannot be guaranteed. For example, when a node is terminated or becomes unavailable, users bound to it will be disconnected and experience a loss of session-specific data, which is anything that is not stored in a shared resource such as Amazon S3, Amazon EFS, or a database.

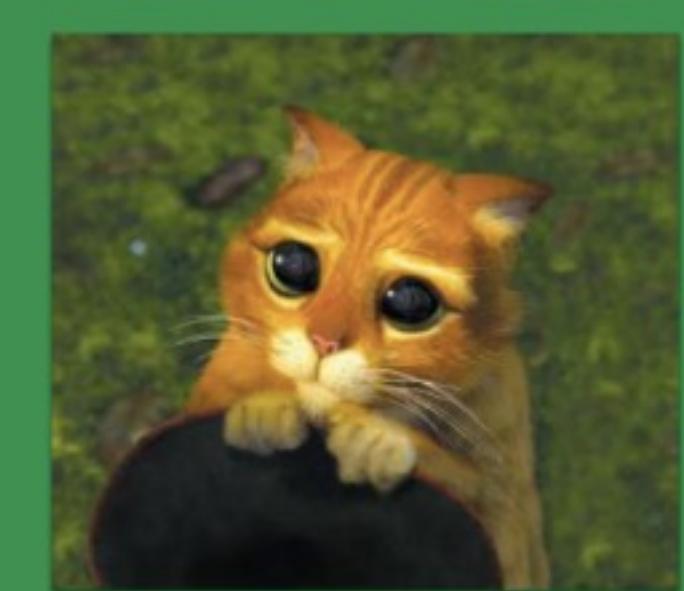
Implement Session Affinity

- For HTTP and HTTPS traffic, you can use the sticky sessions feature of an Application Load Balancer to bind a user's session to a specific instance.
- With this feature, an Application Load Balancer will try to use the same server for that user for the duration of the session.

Distributed Processing

- Use cases that involve the processing of very large amounts of data—anything that can't be handled by a single compute resource in a timely manner—require a distributed processing approach.
- By dividing a task and its data into many small fragments of work, you can execute them in parallel across a set of compute resources.

Pets vs. Cattle



pets

vs



cattle

Disposable Resources Instead of Fixed Servers

- In a traditional infrastructure environment, you have to work with fixed resources because of the upfront cost and lead time of introducing new hardware. This drives practices such as manually logging in to servers to configure software or fix issues, hardcoding IP addresses, and running tests or processing jobs sequentially.
- When designing for cloud, you can take advantage of the dynamically provisioned nature of cloud computing. You can think of servers and other components as temporary resources. You can launch as many as you need and use them only for as long as you need them.
- Another issue with fixed, long-running servers is configuration drift. Changes and software patches applied through time can result in untested and heterogeneous configurations across different environments. You can solve this problem with an immutable infrastructure pattern. With this approach, a server—once launched—is never updated. Instead, when there is a problem or need for an update, the problem server is replaced with a new server that has the latest configuration. This enables resources to always be in a consistent (and tested) state and makes rollbacks easier to perform. This is more easily supported with stateless architectures.

Bootstrapping

- When you launch an AWS resource such as an EC2 instance or Amazon Relational Database Service (Amazon RDS) DB instance, you start with a default configuration.
- You can then execute automated bootstrapping actions, which are scripts that install software or copy data to bring that resource to a particular state.
- You can parameterize configuration details that vary between different environments (such as production or test) so that you can reuse the same scripts without modifications.
- You can set up new EC2 instances with user data scripts and cloud-init directives.

Golden Images



Golden Images

- Certain AWS resource types, such as EC2 instances, Amazon RDS DB instances, and Amazon Elastic Block Store (Amazon EBS) volumes, can be launched from a golden image, which is a snapshot of a particular state of that resource.
- When compared to the bootstrapping approach, a golden image results in faster start times and removes dependencies to configuration services or third-party repositories. This is important in auto-scaled environments where you want to be able to quickly and reliably launch additional resources as a response to demand changes.
- You can customize an EC2 instance and then save its configuration by creating an Amazon Machine Image (AMI).
- You can launch as many instances from the AMI as you need, and they will all include those customizations.
- Each time you want to change your configuration you must create a new golden image, so you must have a versioning convention to manage your golden images over time.

Hybrid

- You can also use a combination of the two approaches: some parts of the configuration are captured in a golden image, while others are configured dynamically through a bootstrapping action.
- Items that do not change often or that introduce external dependencies will typically be part of your golden image.
- An example of a good candidate is your web server software that would otherwise have to be downloaded by a third-party repository each time you launch an instance.

Infrastructure as Code

- Application of the principles we have discussed does not have to be limited to the individual resource level. Because assets are programmable, you can apply techniques, practices, and tools from software development to make your whole infrastructure reusable, maintainable, extensible, and testable.
- AWS CloudFormation templates give you an easy way to create and manage a collection of related AWS resources, and provision and update them in an orderly and predictable fashion.
- You can describe the AWS resources and any associated dependencies or runtime parameters required to run your application.
- Your CloudFormation templates can live with your application in your version control repository, which allows you to reuse architectures and reliably clone production environments for testing.

Loose Coupling

- As application complexity increases, a desirable attribute of an IT system is that it can be broken into smaller, loosely coupled components.
- This means that IT systems should be designed in a way that reduces interdependencies—a change or a failure in one component should not cascade to other components.

Service Discovery

- Applications that are deployed as a set of smaller services depend on the ability of those services to interact with each other.
- Because each of those services can be running across multiple compute resources, there needs to be a way for each service to be addressed.
- For example, in a traditional infrastructure, if your front-end web service needs to connect with your back-end web service, you could hardcode the IP address of the compute resource where this service was running.
- Although this approach can still work in cloud computing, if those services are meant to be loosely coupled, they should be able to be consumed without prior knowledge of their network topology details.
- Apart from hiding complexity, this also allows infrastructure details to change at any time.
- Loose coupling is a crucial element if you want to take advantage of the elasticity of cloud computing where new resources can be launched or terminated at any point in time.
- In order to achieve that you will need some way of implementing service discovery.

Implement Service Discovery

- For an Amazon EC2-hosted service, a simple way to achieve service discovery is through Elastic Load Balancing (ELB). Because each load balancer gets its own hostname, you can consume a service through a stable endpoint. This can be combined with DNS and private Amazon Route 53 zones, so that the particular load balancer's endpoint can be abstracted and modified at any time.
- Another option is to use a service registration and discovery method to allow retrieval of the endpoint IP addresses and port number of any service. Because service discovery becomes the glue between the components, it is important that it is highly available and reliable.
- If load balancers are not used, service discovery should also allow options such as health checks. Amazon Route 53 supports auto naming to make it easier to provision instances for microservices. Auto naming lets you automatically create DNS records based on a configuration you define.
- Other example implementations include custom solutions using a combination of tags, a highly available database, custom scripts that call the AWS APIs, or open-source tools such as Netflix Eureka, Airbnb Synapse, or HashiCorp Consul.

Asynchronous Integration

- Asynchronous integration is another form of loose coupling between services. This model is suitable for any interaction that does not need an immediate response and where an acknowledgement that a request has been registered will suffice.
- It involves one component that generates events and another that consumes them.
- The two components do not integrate through direct point-to-point interaction but usually through an intermediate durable storage layer, such as a queue or a streaming data platform.
- This approach decouples the two components and introduces additional resiliency.

Services, Not Servers

- Developing, managing, and operating applications, especially at scale, requires a wide variety of underlying technology components. With traditional IT infrastructure, companies would have to build and operate all those components.
- Cloud offers a broad set of compute, storage, database, analytics, application, and deployment services that help organizations move faster and lower IT costs.
- Architectures that do not leverage that breadth might not be making the most of cloud computing and might be missing an opportunity to increase developer productivity and operational efficiency.

Databases

- With traditional IT infrastructure, organizations are often limited to the database and storage technologies they can use.
- There can be constraints based on licensing costs and the ability to support diverse database engines.
- On cloud, these constraints are removed by managed database services that offer enterprise performance at open- source cost.
- As a result, it is not uncommon for applications to run on top of a Polyglot data layer choosing the right technology for each workload.

Choose the Right Database Technology for Each Workload

These questions can help you make decisions about which solutions to include in your architecture:

- Is this a read-heavy, write-heavy, or balanced workload? How many reads and writes per second are you going to need? How will those values change if the number of users increases?
- How much data will you need to store and for how long? How quickly will this grow? Is there an upper limit in the near future? What is the size of each object (average, min, max)? How will these objects be accessed?
- What are the requirements in terms of durability of data? Is this data store going to be your “source of truth?”
- What are your latency requirements? How many concurrent users do you need to support?
- What is your data model and how are you going to query the data? Are your queries relational in nature (e.g., JOINs between multiple tables)? Could you denormalize your schema to create flatter data structures that are easier to scale?

Relational Databases

- Relational databases (also known as RDBS or SQL databases) normalize data into well-defined tabular structures known as tables, which consist of rows and columns.
- They provide a powerful query language, flexible indexing capabilities, strong integrity controls, and the ability to combine data from multiple tables in a fast and efficient manner.
- Managed database makes it easy to set up, operate, and scale a relational database in the cloud with support for many familiar database engines.

Database Scalability

- Relational databases can scale vertically by upgrading to a larger instance or adding more and faster storage.
- For read-heavy applications, you can also horizontally scale beyond the capacity constraints of a single DB instance by creating one or more read replicas.
- Read replicas are separate database instances that are replicated asynchronously. As a result, they are subject to replication lag and might be missing some of the latest transactions.
- Application designers need to consider which queries have tolerance to slightly stale data. Those queries can be executed on a read replica, while the remainder should run on the primary node.
- Read replicas can not accept any write queries.

High Availability

- For any production relational database use the Amazon RDS Multi-AZ deployment feature, which creates a synchronously replicated standby instance in a different Availability Zone.
- In case of failure of the primary node, Amazon RDS performs an automatic failover to the standby without the need for manual administrative intervention.
- When a failover is performed, there is a short period during which the primary node is not accessible.
- Resilient applications can be designed for Graceful Failure by offering reduced functionality, such as read-only mode by using read replicas.
- Amazon Aurora provides multi-master capability to enable reads and writes to be scaled across Availability Zones and also supports cross-Region replication.

Anti-Patterns

- If your application primarily indexes and queries data with no need for joins or complex transactions—especially if you expect a write throughput beyond the constraints of a single instance—consider a NoSQL database instead.
- If you have large binary files (audio, video, and image), it will be more efficient to store the actual files in Amazon S3 and only hold the metadata for the files in your database.

NoSQL Databases

- NoSQL databases trade some of the query and transaction capabilities of relational databases for a more flexible data model that seamlessly scales horizontally.
- NoSQL databases use a variety of data models, including graphs, key-value pairs, and JSON documents, and are widely recognized for ease of development, scalable performance, high availability, and resilience.

Removing Single Points of Failure

- Production systems typically come with defined or implicit objectives for uptime.
- A system is highly available when it can withstand the failure of an individual component or multiple components, such as hard disks, servers, and network links.
- To help you create a system with high availability, you can think about ways to automate recovery and reduce disruption at every layer of your architecture.

Introducing Redundancy

- Single points of failure can be removed by introducing redundancy, which means you have multiple resources for the same task. Redundancy can be implemented in either standby or active mode.
- In standby redundancy, when a resource fails, functionality is recovered on a secondary resource with the failover process.
- The failover typically requires some time before it completes, and during this period the resource remains unavailable.
- The secondary resource can either be launched automatically only when needed (to reduce cost), or it can already be running idle (to accelerate failover and minimize disruption).
- Standby redundancy is often used for stateful components such as relational databases.
- In active redundancy, requests are distributed to multiple redundant compute resources. When one of them fails, the rest can simply absorb a larger share of the workload.
- Compared to standby redundancy, active redundancy can achieve better usage and affect a smaller population when there is a failure.

Detect Failure

- You should aim to build as much automation as possible in both detecting and reacting to failure.
- You can use services such as ELB and Amazon Route 53 to configure health checks and mask failure by routing traffic to healthy endpoints.
- You can replace unhealthy nodes automatically using Auto Scaling or by using the Amazon EC2 auto-recovery.
- It won't be possible to predict every possible failure scenario on day one.
- Make sure you collect enough logs and metrics to understand normal system behavior. After you understand that, you will be able to set up alarms for manual intervention or automated response.

Optimize for Cost

- When you move your existing architectures into the cloud, you can reduce capital expenses and drive savings as a result of the AWS economies of scale.
- By iterating and using more cloud capabilities, you can realize further opportunity to create cost- optimized cloud architectures.

Right Sizing

- AWS offers a broad range of resource types and configurations for many use cases. For example, services such as Amazon EC2, Amazon RDS, Amazon Redshift, and Amazon ES offer many instance types. In some cases, you should select the cheapest type that suits your workload's requirements. In other cases, using fewer instances of a larger instance type might result in lower total cost or better performance. You should benchmark your application environment and select the right instance type depending on how your workload uses CPU, RAM, network, storage size, and I/O.
- Similarly, you can reduce cost by selecting the right storage solution for your needs. For example, Amazon S3 offers a variety of storage classes, including Standard, Reduced Redundancy, and Standard-Infrequent Access. Other services, such as Amazon EC2, Amazon RDS, and Amazon ES, support different EBS volume types (magnetic, general purpose SSD, provisioned IOPS SSD) that you should evaluate.
- Over time, you can continue to reduce cost with continuous monitoring and tagging. Just like application development, cost optimization is an iterative process. Because, your application and its usage will evolve over time, and because AWS iterates frequently and regularly releases new options, it is important to continuously evaluate your solution.

Caching

- Caching is a technique that stores previously calculated data for future use.
- This technique is used to improve application performance and increase the cost efficiency of an implementation.
- It can be applied at multiple layers of an IT architecture.

Application Data Caching

- Applications can be designed so that they store and retrieve information from fast, managed, in-memory caches.
- Cached information might include the results of I/O- intensive database queries, or the outcome of computationally intensive processing.
- When the result set is not found in the cache, the application can calculate it, or retrieve it from a database or expensive, slowly mutating third-party content, and store it in the cache for subsequent requests.
- However, when a result set is found in the cache, the application can use that result directly, which improves latency for end users and reduces load on back-end systems.
- Your application can control how long each cached item remains valid.
- In some cases, even a few seconds of caching for very popular objects can result in a dramatic decrease on the load for your database.

Conclusion

- When you design your architecture in the Cloud, it is important to consider the important principles and design patterns, including how to select the right database for your application, and how to architect applications that can scale horizontally and with high availability.
- Because each implementation is unique, you must evaluate how to apply this guidance to your implementation.
- The topic of cloud computing architectures is broad and continuously evolving.

Additional Resources

See Lecture Page