fands™
A Fast AND Steady Approach

AWS

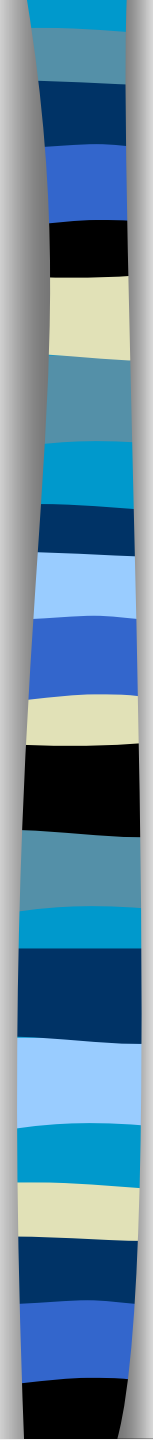
Presented by
VAISHALI TAPASWI

FANDS INFONET Pvt.Ltd.
www.fandsindia.com

Ground Rules

- Turn off cell phone. If you cannot please keep it on silent mode. You can go out and attend your call.
- If you have any questions or issues please let me know immediately.
- Let us be punctual.

Agenda



fands™

Git

Git

- As **Git** is a distributed version control system, it can be used as a server out of the box. Dedicated **Git** server software helps, amongst other features, to add access control, display the contents of a **Git** repository via the web, and help managing multiple repositories.

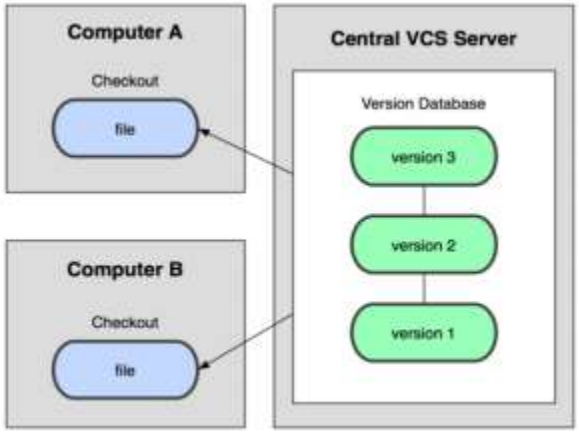
Version Control Systems

- Version Control (or Revision Control, or Source Control) is all about managing multiple versions of documents, programs, web sites, etc.
  - Almost all “real” projects use some kind of version control
  - Essential for team projects, but also very useful for individual projects
- Some well-known version control systems are CVS, Subversion, Mercurial, and Git
  - CVS and Subversion use a “central” repository; users “check out” files, work on them, and “check them in”
  - Mercurial and Git treat all repositories as equal
- Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

Why Version Control?

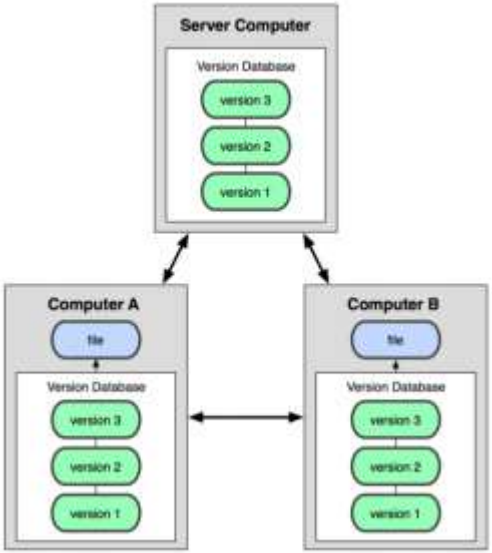
- For working by yourself:
  - Gives you a “time machine” for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes

Centralized VCS



- In Subversion, CVS, Perforce, etc.
  - A central server repository (repo) holds the "official copy" of the code
  - The server maintains the sole version history of the repo
- You make "checkouts" of it to your local copy
  - You make local modifications
  - Your changes are not versioned
- When you're done, you "check in" back to the server
  - your checkin increments the repo's version

Distributed VCS (Git)



- In git, mercurial, etc., you don't "checkout" from a central repo
  - You "clone" it and "pull" changes from it
- Your local repo is a complete copy of everything on the remote server
  - Yours is "just as good" as theirs
- Many operations are local:
  - Check in/out from local repo
  - Commit changes to local repo
  - Local repo keeps version history
- When you're ready, you can "push" changes back to server

Why Git?

- Git has many advantages over earlier systems
  - More efficient, better workflow, etc.
  - See the literature for an extensive list of reasons
  - Of course, there are always those who disagree
  - Very Popular

Version Control Terminology

- Version Control System (VCS) or (SCM)
- Repository
- Commit
- SHA
- Working Directory
- Checkout
- Staging Area/Index
- Branch

Version Control Terminology

- Version Control System :
  - A VCS allows you to: revert files back to a previous state, revert the entire project back to a previous state, review changes made over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.
- Repository:
  - A directory that contains your project work which are used to communicate with Git. Repositories can exist either locally on your computer or as a remote copy on another computer.

Version Control Terminology

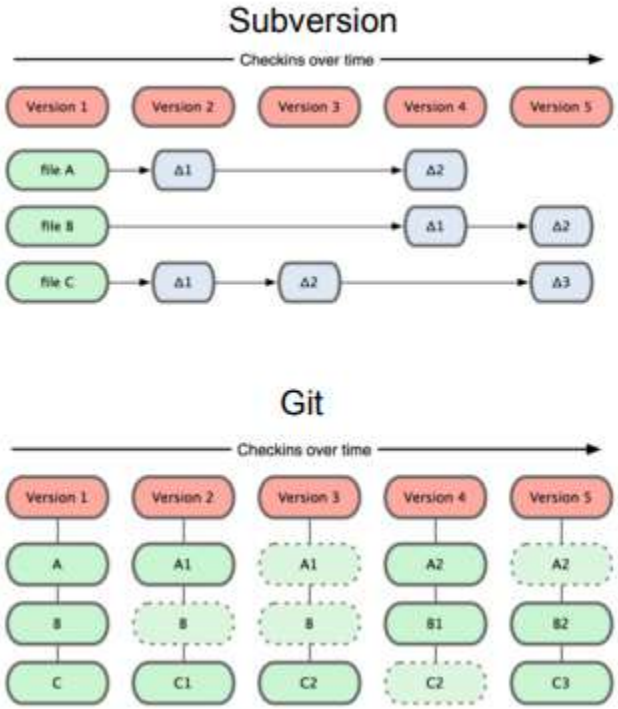
- Commit
  - Git thinks of its data like a set of snapshots of a mini file system.
  - Think of it as a save point during a video game.
- SHA
  - A SHA is basically an ID number for each commit.
  - Ex. E2adf8ae3e2e4ed40add75cc44cf9d0a869afeb6
- Branch
  - A branch is when a new line of development is created that diverges from the main line of development. This alternative line of development can continue without altering the main line.

Version Control Terminology

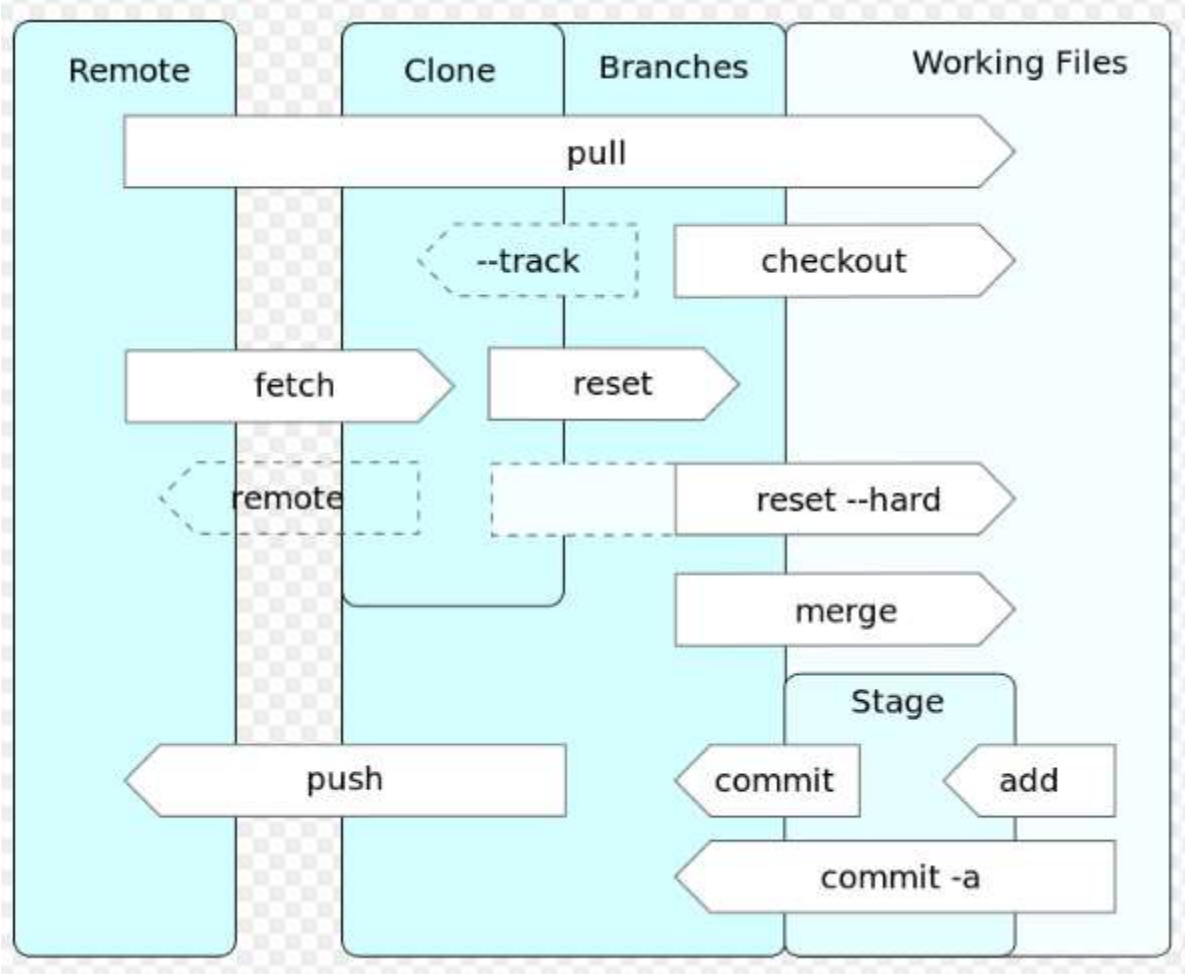
- Working Directory
  - files that you see in your computer's file system. When you open project files up on a code editor, you're working with files in the Working Directory.
- Checkout
  - Content in the repository has been copied to the Working Directory. Possible to checkout many things from a repository; a file, a commit, a branch, etc.
- Staging Area
  - You can think of the staging area as a prep table where Git will take the next commit. Files on the Staging Index are poised to be added to the repo

Git

- Centralized VCS like Subversion track version data on each individual file.
- Git keeps "snapshots" of the entire state of the project.
  - Each checkin version of the overall code has a copy of each file in it.
  - Some files change on a given checkin, some do not.
  - More redundancy, but faster.

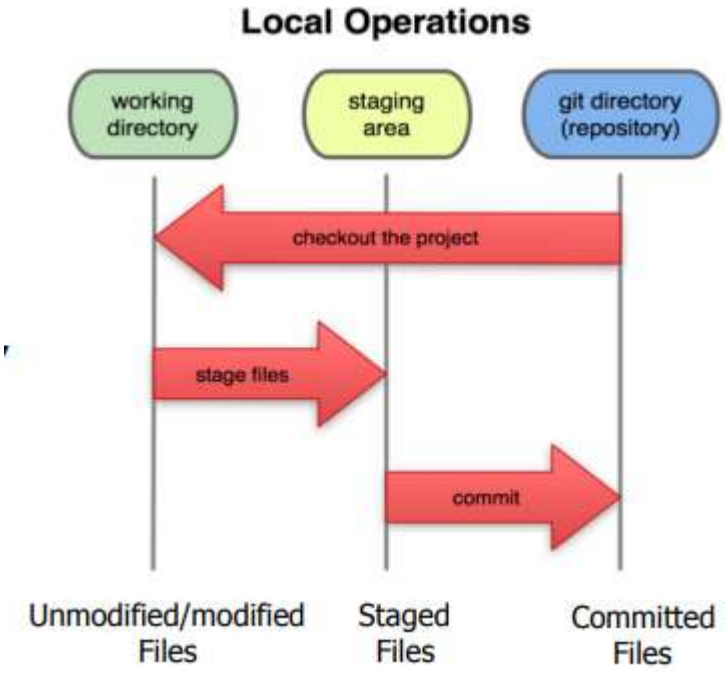


Git

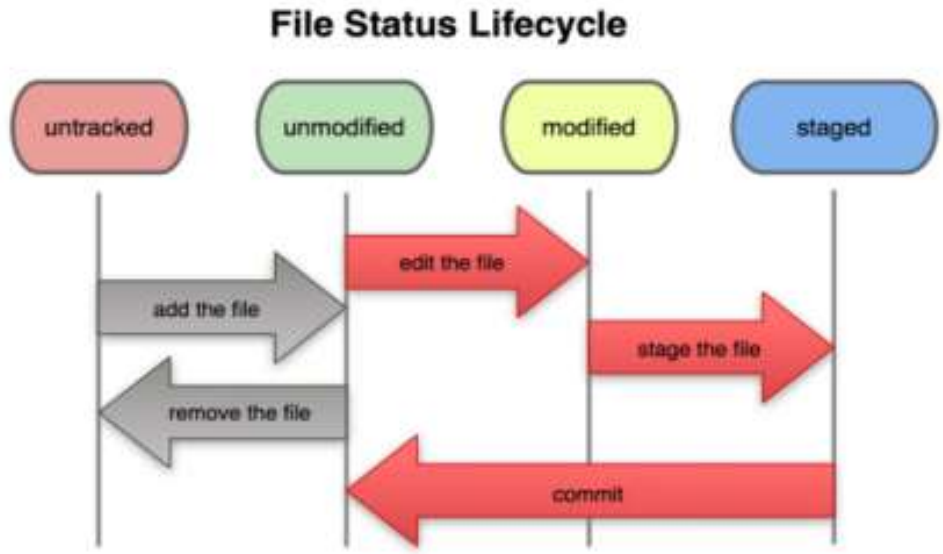

Remote
Clone
Branches
Working Files
pull
--track
checkout
fetch
reset
remote
reset --hard
merge
Stage
push
commit
add
commit -a

In your local copy on git, files can be:

- In your local repo
  - (committed)
- Checked out and modified, but not yet committed
  - (working copy)
- Or, in-between, in a "staging" area
  - Staged files are ready to be committed.
  - A commit saves a snapshot of all staged state.



Basic Git Workflow



- Modify files in your working directory.
- Stage files, adding snapshots of them to your staging area.
- Commit, which takes the files in the staging area and stores that snapshot permanently to your Git directory.

Initial Git configuration

- Set the name and email for Git to use when you commit:
  - git config --global user.name “.."
  - git config --global user.email e@gmail.com
  - You can call git config –list to verify these are set.
- Set the editor that is used for writing commit messages:
  - git config --global core.editor nano
    - (it is vim by default)

Creating a Git Repo

- To create a new local Git repo in your current directory:
  - git init
    - This will create a .git directory in your current directory.
    - Then you can commit files in that directory into the repo.
  - git add filename
  - git commit –m "commit message"
- To clone a remote repo to your current directory:
  - git clone url localDirectoryName
    - This will create the given local directory, containing a working copy of the files from the repo, and a .git directory (used to hold the staging area and your local repo)

Git Commands

command	description
`git clone` ***`url [dir]`***	copy a Git repository so you can add to it
`git add` ***`file`***	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help` ***`[command]`***	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository
others: `init`, `reset`, `branch`, `checkout`, `merge`, `log`, `tag`	

Add and commit a file

- The first time we ask a file to be tracked, and every time before we commit a file, we must add it to the staging area:
  - git add Hello.java Goodbye.java
    - Takes a snapshot of these files, adds them to the staging area.
- To move staged changes into the repo, we commit:
  - git commit –m "Fixing bug #22"
- To undo changes on a file before you have committed it:
  - git reset HEAD -- filename (unstages the file)
  - git checkout -- filename (undoes your changes)
  - All these commands are acting on your local version of repo.

Viewing/undoing changes

- To view status of files in working directory and staging area:
  - git status or git status –s (short version)
- To see what is modified but unstaged:
  - git diff
- To see a list of staged changes:
  - git diff --cached
- To see a log of all changes in your local repo:
  - git log or git log --oneline (shorter version)
  - git log -5 (to show only the 5 most recent updates) etc

Branching and Merging

Git uses branching heavily to switch between multiple tasks.

- To create a new local branch:
  - git branch name
- To list all local branches: (* = current branch)
  - git branch
- To switch to a given local branch:
  - git checkout branchname
- To merge changes from a branch into the local master:
  - git checkout master
  - git merge branchname

fands™

Merge Conflicts

The conflicting file will contain <<< and >>> sections to indicate where Git was unable to resolve a conflict:

```
<<<<<<< HEAD:index.html
<div id="footer">todo: message here</div>
=======
<div id="footer">
  thanks for visiting our site
</div>
>>>>>>> SpecialBranch:index.html
```

branch 1's version

branch 2's version

Find all such sections, and edit them to the proper state (whichever of the two versions is newer / better / more correct).

Interaction with Remote Repo

- Push your local changes to the remote repo.
- Pull from remote repo to get most recent changes.
  - (fix conflicts if necessary, add/commit them to your local repo)
- To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:
  - git pull origin master
- To put your changes from your local repo in the remote repo:
  - git push origin master

GitHub

- GitHub.com is a site for online storage of Git repositories.
  - You can create a remote repo there and push code to it.
  - Many open source projects use it, such as the Linux kernel.
  - You can get free space for open source projects, or you can pay for private projects.

AWS Introduction

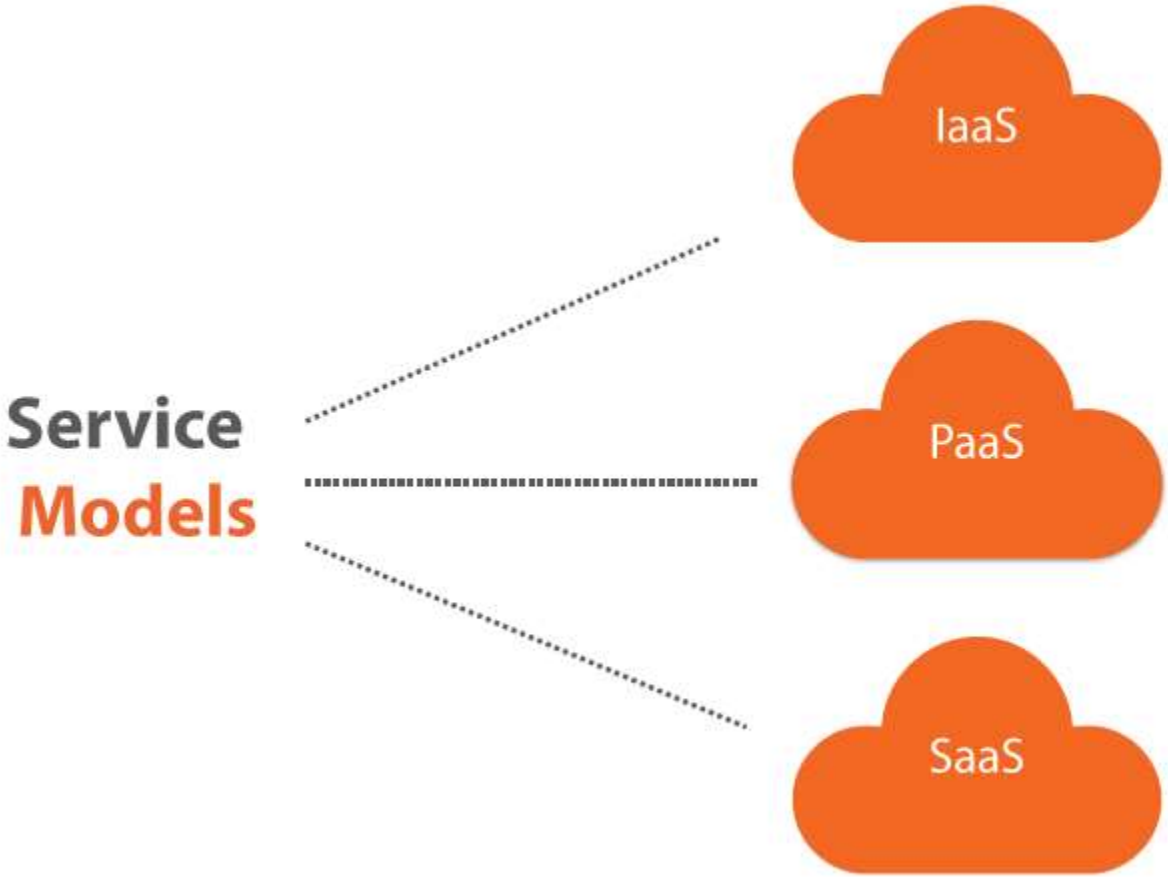
What is Cloud Computing?

- A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, server, storage, applications and services) that can be rapidly provisioned and released with minimal management effort of service provider interaction.

5 Essential Characters

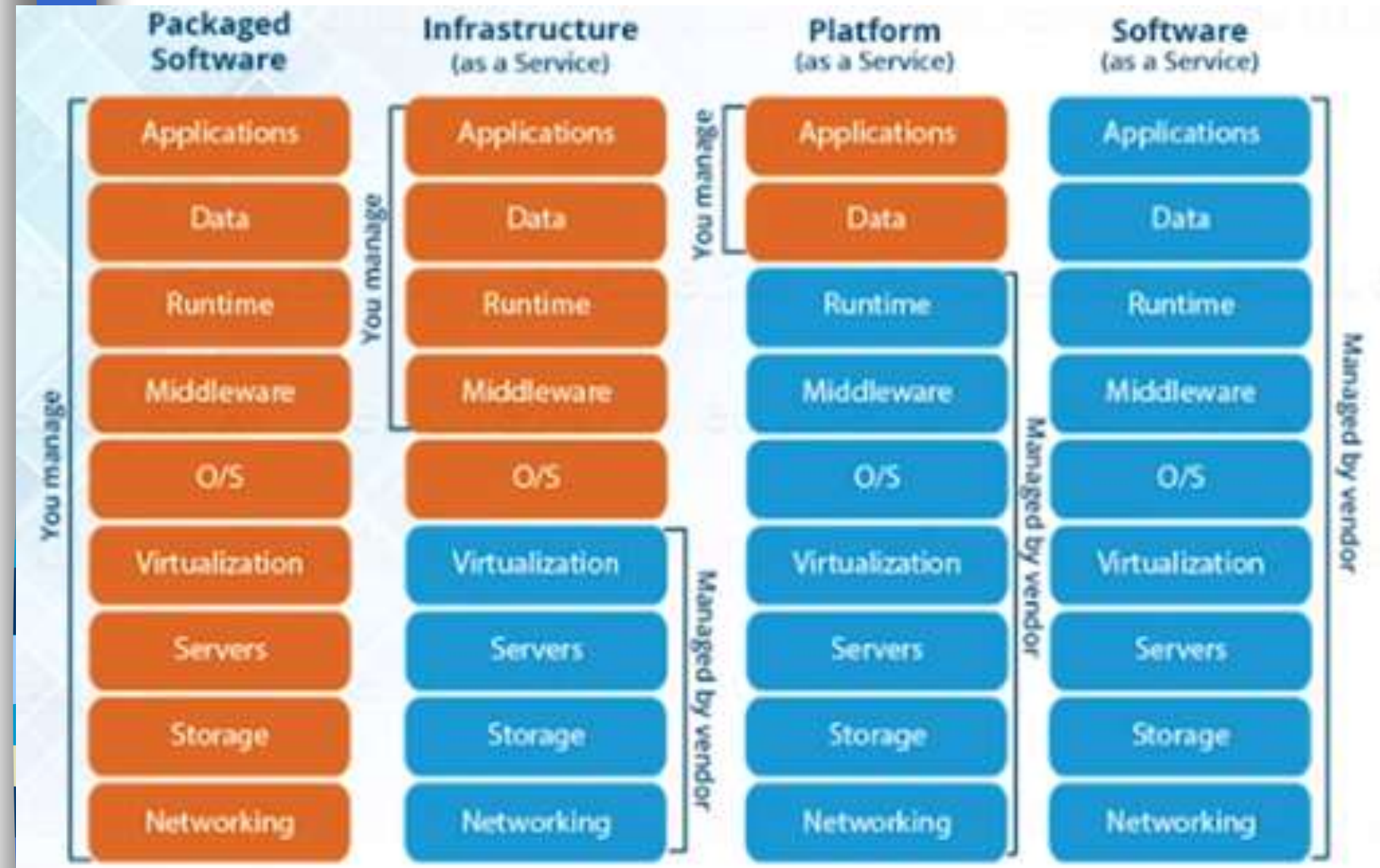
- On demand self services
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service
- Multi Tenacity

3 Service Models



AS A SERVICE

- IAAS: INFRASTRUCTURE
  - Amazon Web Services (AWS), Cisco Metapod, Microsoft Azure, Google Compute Engine (GCE)
- PAAS: PLATFORM
  - Apprenda
- SAAS: SOFTWARE
  - Google Apps, Concur, Citrix GoToMeeting, Cisco WebEx


Packaged Software
Infrastructure (as a Service)
Platform (as a Service)
Software (as a Service)
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
You manage
You manage
You manage
Managed by vendor
Managed by vendor
Managed by vendor

Cloud Compute Infrastructure

- Compute
- Storage
- Network
- Data Centres and Regions
- Shared Services
- Platforms

Who's who?

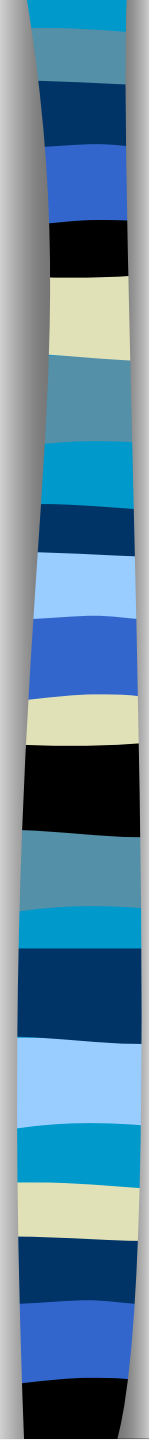
- Google Cloud Platform (GCP):
  - With only 5 years in operation, have created good presence in the market. The initial push was done to power their own services such as YouTube and Google. Later on, they built enterprise services and enabled anyone to host in the cloud.
- Amazon Web Services (AWS)
  - 11 years in operation, one of the oldest players in the cloud market. Their computing services are extensive and cover important cloud sections such as deployment, mobile networking, etc
- Microsoft Azure
  - Azure is also 6 years old and has shown great promise in the market. They can easily be associated with the leader group in the market with AWS. Provides a complete set of cloud services.

AWS Global Infrastructure

- 16 Regions
  - “the western US, eastern US, central Europe"
- Each region consists of multiple availability zones
  - Separate data centers - Ireland is the region with 3 availability zones
  - Distinct locations from within an AWS region that are engineered to be isolated from failures (one can go down, others stay up)
- 54 Edge Locations
  - CND end points - there are many more edge locations than regions
  - used by cloud front to cache files near the user where they access them to reduce latency

Services

- Storage
  - Simple Storage Service - S3
- Compute
  - Elastic Compute Cloud – EC2
- Database
  - Relational Database Service – RDS
- Security, Identity, & Compliance
  - Identity and Access Management - IAM



fands™

AWS CLI

AWS CLI

- The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services from the command line and automate them through scripts.

Simple Configuration

- IAM
  - Create user to allow cli access with admin role
- aws configure
  - Configure access key id and secret access key
  - Default Region
  - Default output format
    - Json, text, table
- Watch .aws folder and files created

IAM

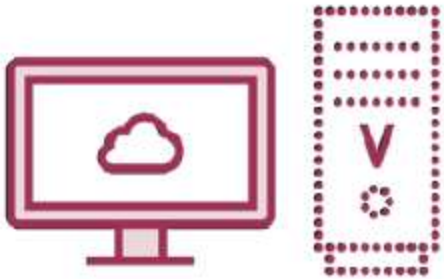
- Console
  - Create users and groups
  - Check permissions
- AWS CLI
  - Create user
  - Create User
  - Associate
  - Grant Permissions

S3

- Using Console
  - Make a file public
  - Versioning
  - Encryption
- Using CLI
  - Create s3 bucket
  - List Contents of bucket
  - Move files from local directory to s3 bucket

What Can You Do with EC2?

Run Applications

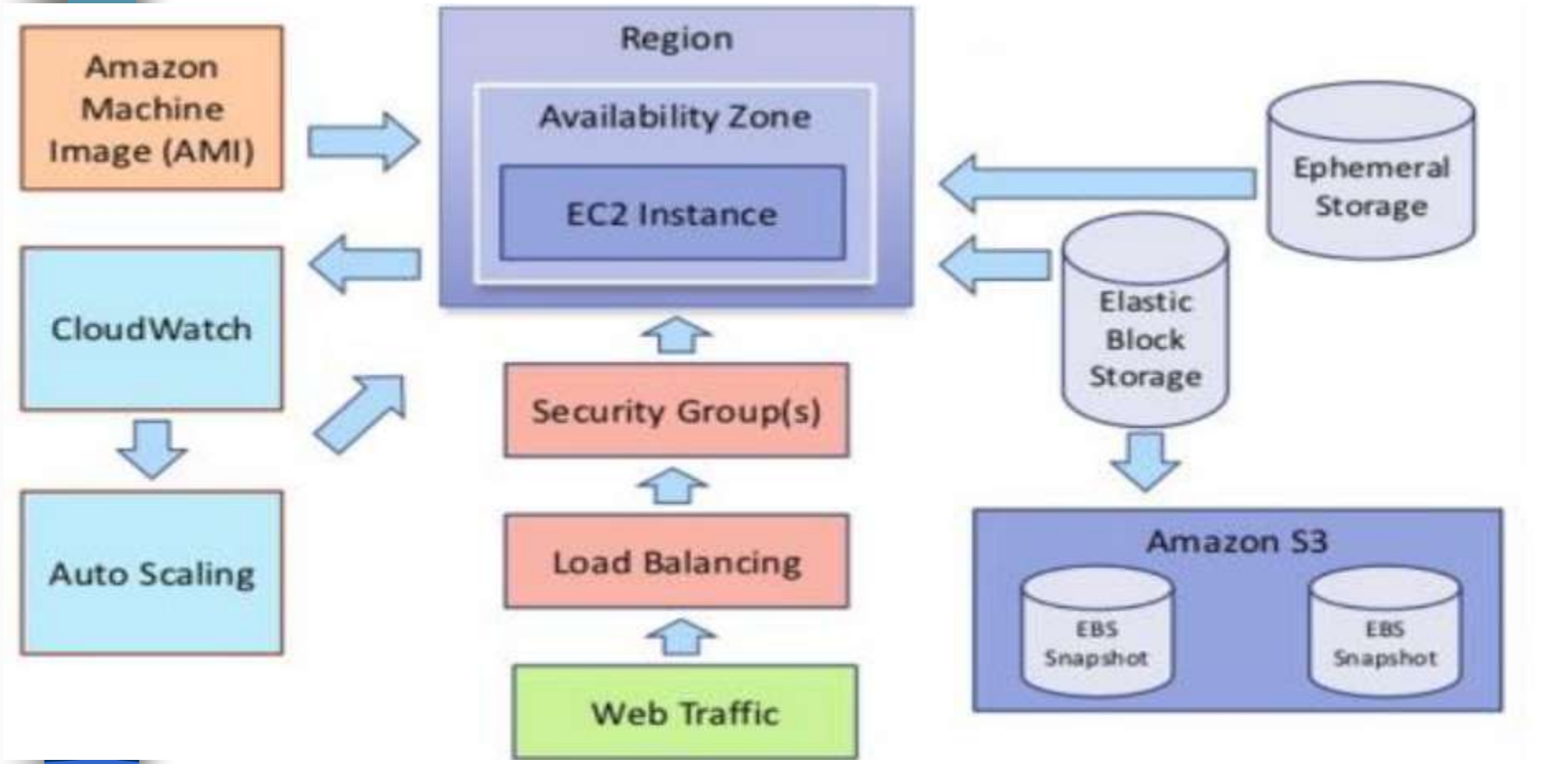


Virtual Desktop

3rd Party Software

Computing!

EC2 Architecture



EC2

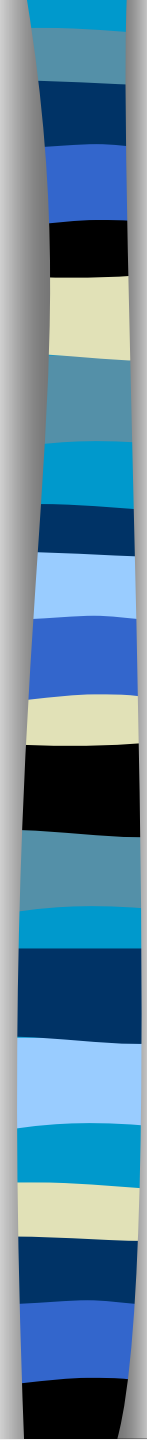
- Console
  - Create EC2 instance
  - Install Node/JRE
  - Deploy application
  - Test (networking ports)
- CLI
  - List EC2 instances
  - Stop EC2 instance
  - Terminate EC2 instance

Relational Database Service (RDS)

- Easy to set up, operate, and scale
- Provides cost-efficient and resizable capacity while automating time-consuming administration tasks such as hardware provisioning, database setup, patching and backups.
- Fast performance, high availability, security and compatibility
- Choose from Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and SQL Server

RDS

- Console
  - Create MySQL database
- MySQL Chrome plugin
  - Connect to MySQL Db
  - Create Table
  - Retrieve Records
- CLI
  - Insert
  - Delete



DynamoDB

SQL has ruled for two decades

- **Store persistent data**

  Storing large amounts of data on disk, while allowing applications to grab the bits they need through queries

- **Application Integration**

  Many applications in an enterprise need to share information. By getting all applications to use the database, we ensure all these applications have consistent, up-to-date data

- **Mostly Standard**

  The relational model is widely used and understood. Interaction with the database is done with SQL, which is a (mostly) standard language. This degree of standardization is enough to keep things familiar so people don't need to learn new things

- **Concurrency Control**

  Many users access the same information at the same time. Handling this concurrency is difficult to program, so databases provide transactions to help ensure consistent interaction.

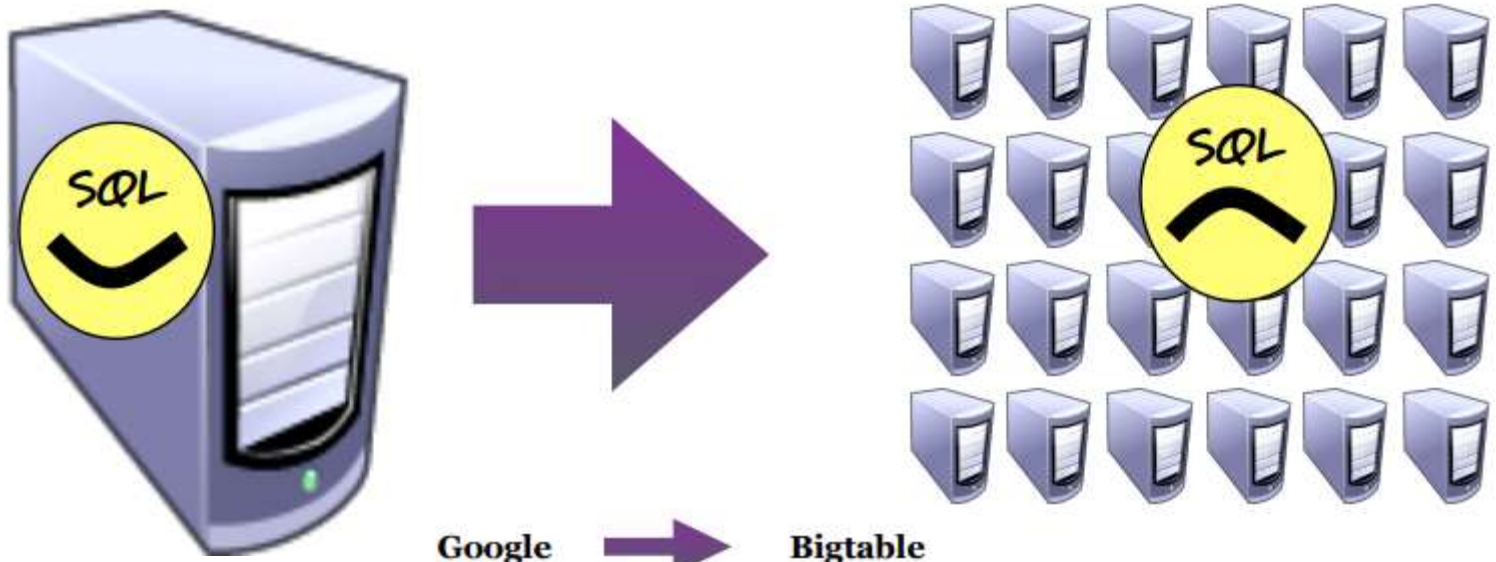
- **Reporting**

  SQL's simple data model and standardization has made it a foundation for many reporting tools

SQL's dominance is cracking

Relational databases are designed to run on a single machine, so to scale, you need buy a bigger machine

But it's cheaper and more effective to scale horizontally by buying lots of machines.



Google → Bigtable

Amazon → Dynamo

NoSQL Databases

- There is no standard definition of what NoSQL means.
- The term began with a workshop organized in 2009, but there is much argument about what databases can truly be called NoSQL.

NoSQL Databases

- While there is no formal definition, there are some common characteristics
  - they don't use the relational data model, and thus don't use the SQL language
  - they tend to be designed to run on a cluster
  - they tend to be Open Source
  - they don't have a fixed schema, allowing you to store any data in any record

NoSQL Databases

We should also remember Google's Bigtable and Amazon's SimpleDB. While these are tied to their host's cloud service, they certainly fit the general operating characteristics

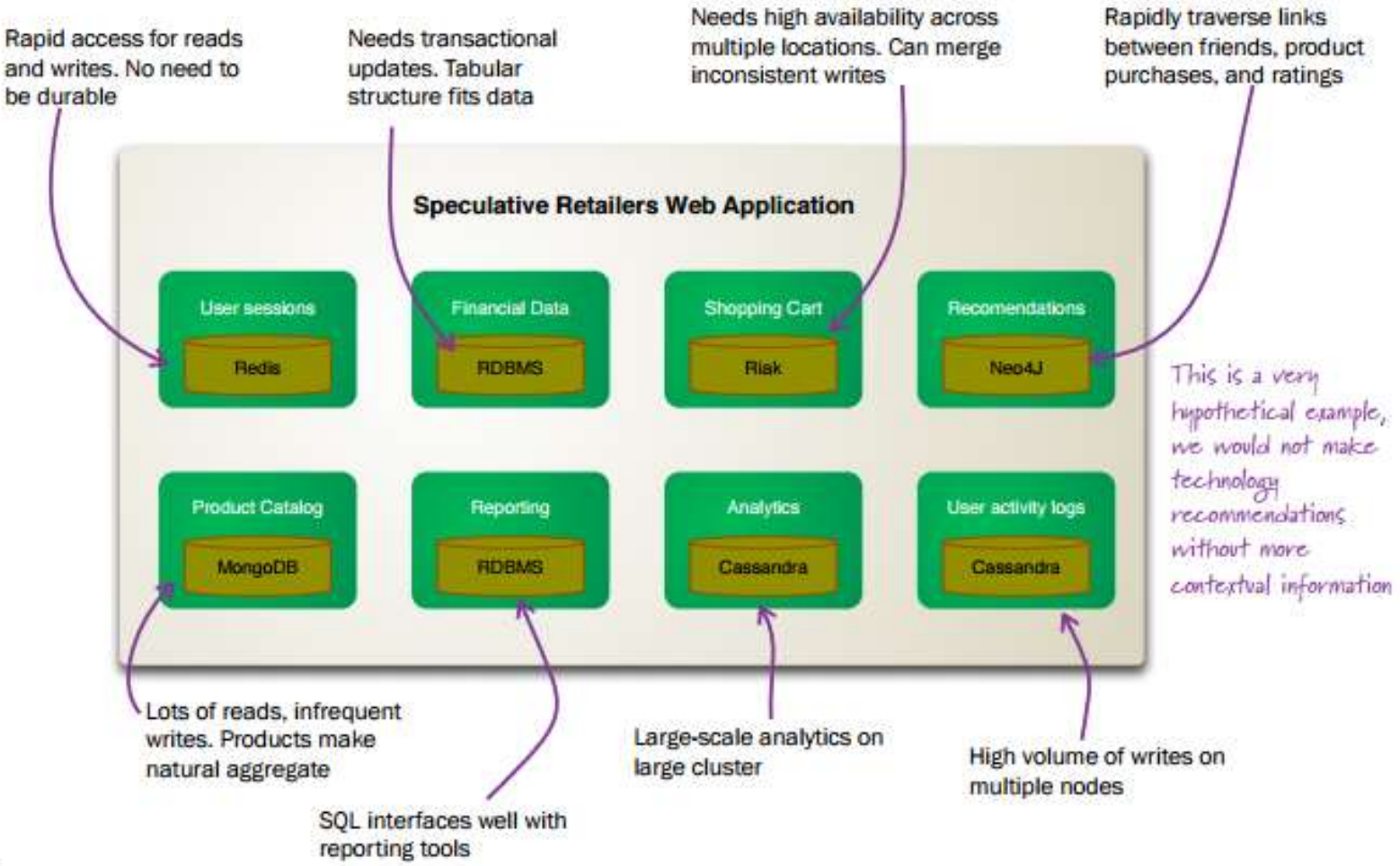
NoSQL Databases

- Main Advantages
  - Reduce Development Drag
  - Embrace Large Scale
- This does not mean Relational is dead
  - the relational model is still relevant
  - ACID transactions
  - Tools
  - Familiarity

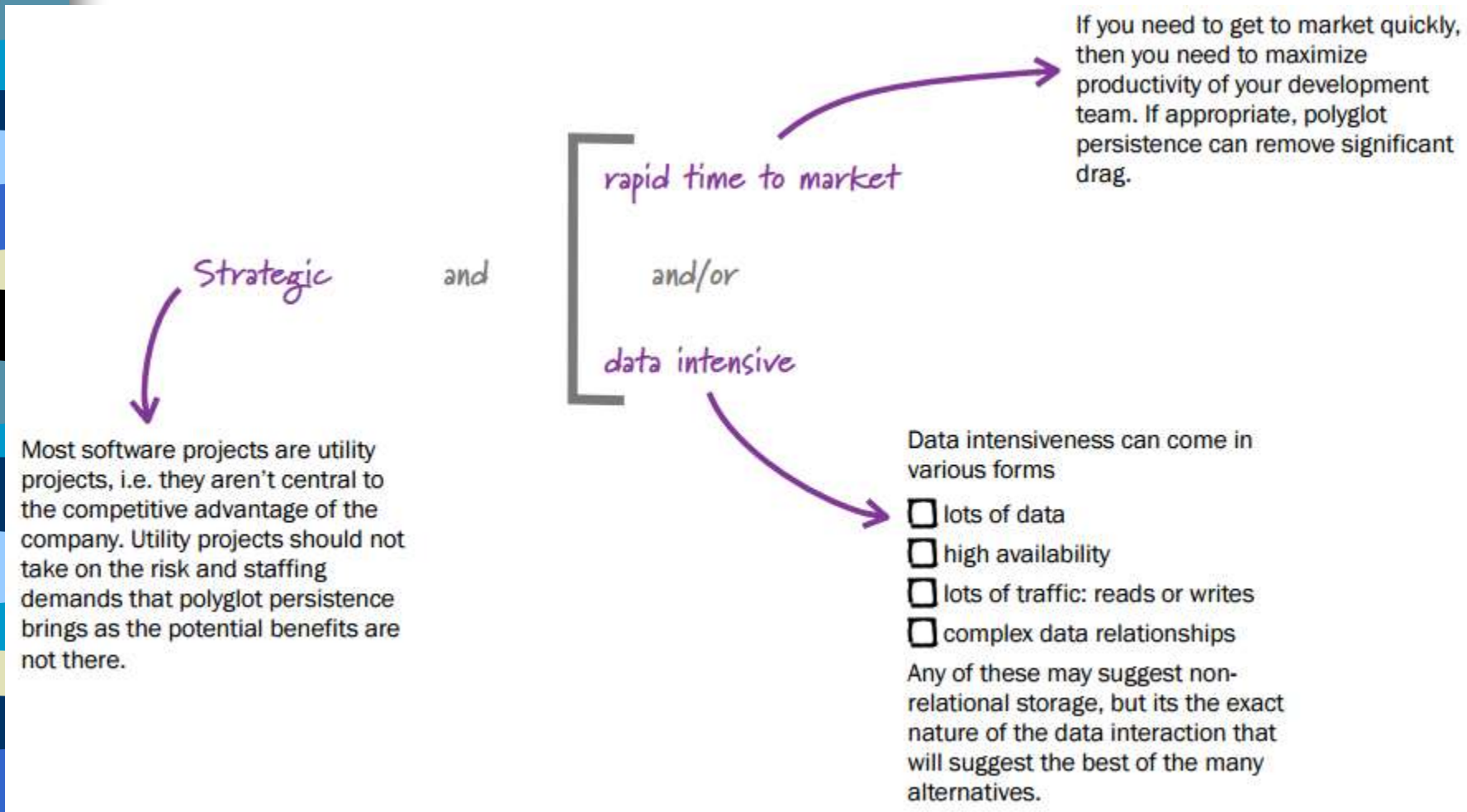
Polyglot Persistence

- Using multiple data storage technologies, chosen based upon the way data is being used by individual applications. Why store binary images in relational database, when there are better storage systems?

What might Polyglot Persistence look like?



Candidates for polyglot persistence?

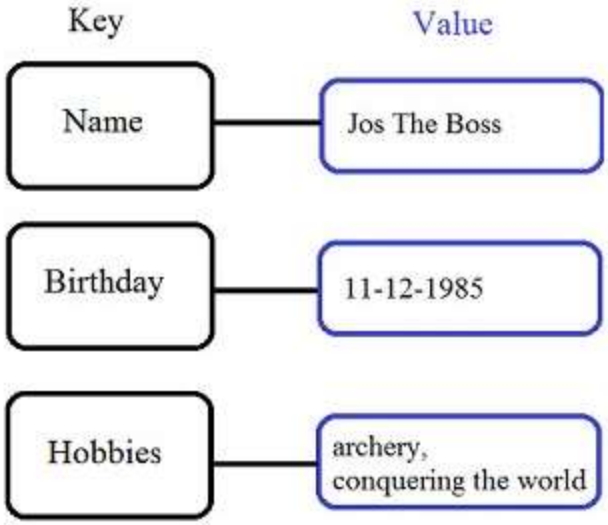


Four NoSQL db Types/Models

- Key-Value store
- Document Data Model
- Column-Oriented database
- Graph Database
- (Key-Value + Document Data Model) = Aggregate Oriented

Key-Value Stores

- Key-value stores are the least complex of the NoSQL databases.
- Simplicity makes it the most scalable of the NoSQL database types, capable of storing huge amounts of data.



Redis, Voldemort, Riak, and Amazon's Dynamo.

Document Stores

- Document stores are one step up in complexity from key-value stores
- Assume a certain document structure that can be specified with a schema.
- Document stores appear the most natural among the NoSQL database types because they’re designed to store everyday documents as is, and they allow for complex querying and calculations on this often already aggregated form of data.

Document Stores

MongoDB and CouchDB


```
{
  "articles": [
    {
      "title": "title of the article",
      "articleID": 1,
      "body": "body of the article",
      "author": "Isaac Asimov",
      "comments": [
        {
            "username": "Fritz",
            "join date": "1/4/2014",
            "commentid": 1,
            "body": "this is a great article",
            "replies": [
                {
                    "username": "Freddy",
                    "join date": "11/12/2013",
                    "commentid": 2,
                    "body": "seriously? it's rubbish"
                }
            ]
        },
        {
            "username": "Stark",
            "join date": "19/06/2011",
            "commentid": 3,
            "body": "I don't agree with the conclusion"
        }
      ]
    }
  ]
}
```

Whereas relational databases chop up data, Document stores save documents as a single entity

Column Oriented

- Traditional relational databases are row-oriented, with each row having a row-id and each field within the row stored together in a table.

Apache HBase, Facebook's Cassandra, Hypertable, and the grandfather of wide-column stores, Google BigTable.

Name	ROWID
Jos The Boss	1
Fritz Schneider	2
Freddy Stark	3
Delphine Thewiseone	4

Birthday	ROWID
11-12-1985	1
27-1-1978	2
16-9-1986	4

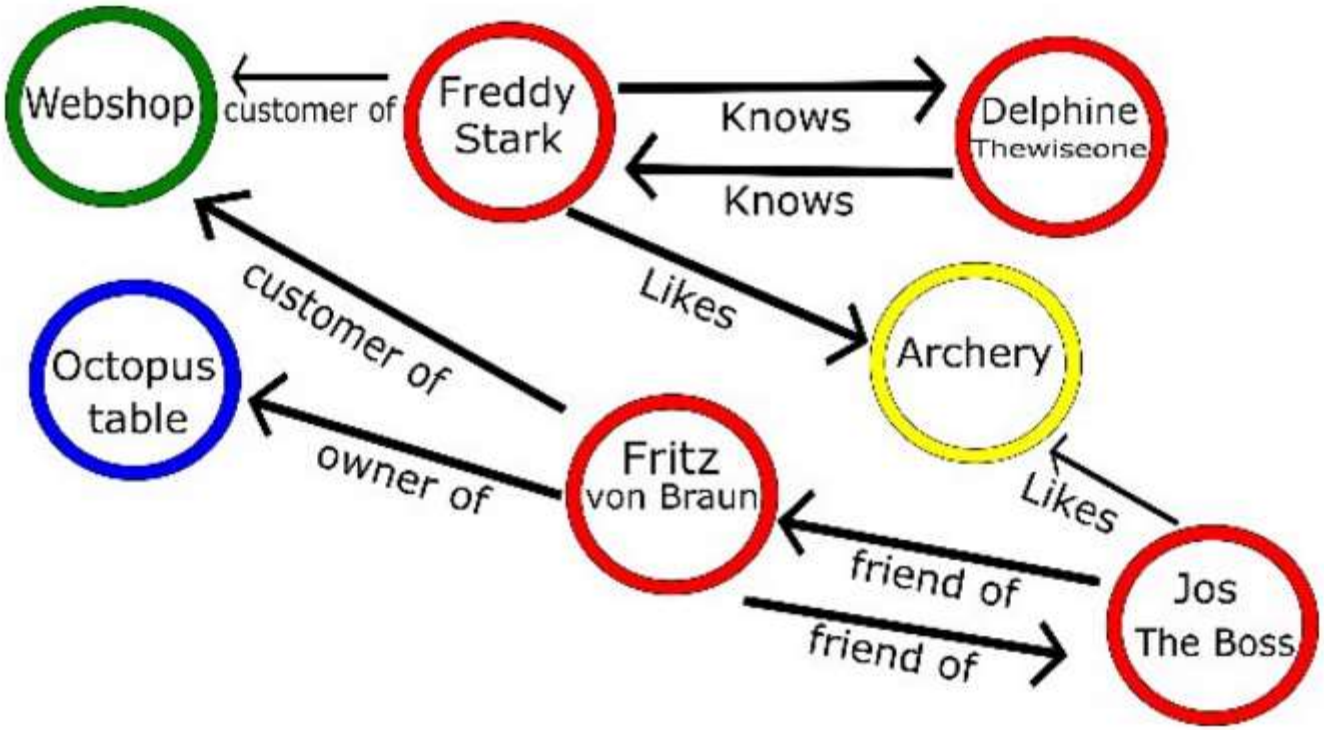
Hobbies	ROWID
archery	1, 3
conquering the world	1
building things	2
surfing	2
swordplay	3
lollygagging	3

A column-oriented database stores each column separately

Graph Databases

- The last big NoSQL database type is the most complex one, geared toward storing relations between entities in an efficient manner. When the data is highly interconnected, such as for social networks, scientific paper citations, or capital asset clusters, graph databases are the answer. Graph or network data has two main components:
  - *Node* : The entities themselves. In a social network this could be people.
  - *Edge*: The relationship between two entities. This relationship is represented by a line and has its own properties. An edge can have a direction, for example, if the arrow indicates who is whose boss.

Graph Databases



Graph databases like Neo4j also claim to uphold ACID, whereas document stores and key-value stores adhere to BASE.

DynamoDB

- Key-value and document database that delivers single-digit millisecond performance at any scale.
- Fully managed, multi-region, multi-active, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications.
- Handle more than 10 trillion requests per day and can support peaks of more than 20 million requests per second.

fands™

DynamoDB

- Create Table
- Console
  - CRUD on table
- CLI
  - CRUD on table
    - list-tables
    - get-item
    - delete-item
    - update-item
    - query

CloudWatch

CloudWatch

- Monitoring and Observability service
- Provides data and actionable insights to monitor your applications, respond to system-wide performance changes, optimize resource utilization, and get a unified view of operational health.
- Collects monitoring and operational data in the form of logs, metrics, and events, providing you with a unified view of AWS resources, applications, and services that run on AWS and on-premises servers.

CloudWatch

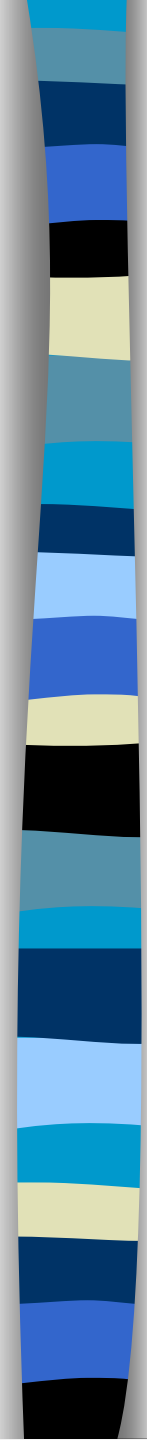
- Use CloudWatch to detect anomalous behavior in your environments, set alarms, visualize logs and metrics side by side, take automated actions, troubleshoot issues, and discover insights to keep your applications running smoothly.

Benefits

- Observability on a single platform across applications and infrastructure
- Easiest way to collect metrics in AWS and on-premises
- Improve operational performance and resource optimization
- Get operational visibility and insight
- Derive actionable insights from logs

CloudWatch

- Enable CloudWatch
  - Check metrics
  - Graphs
  - Logs



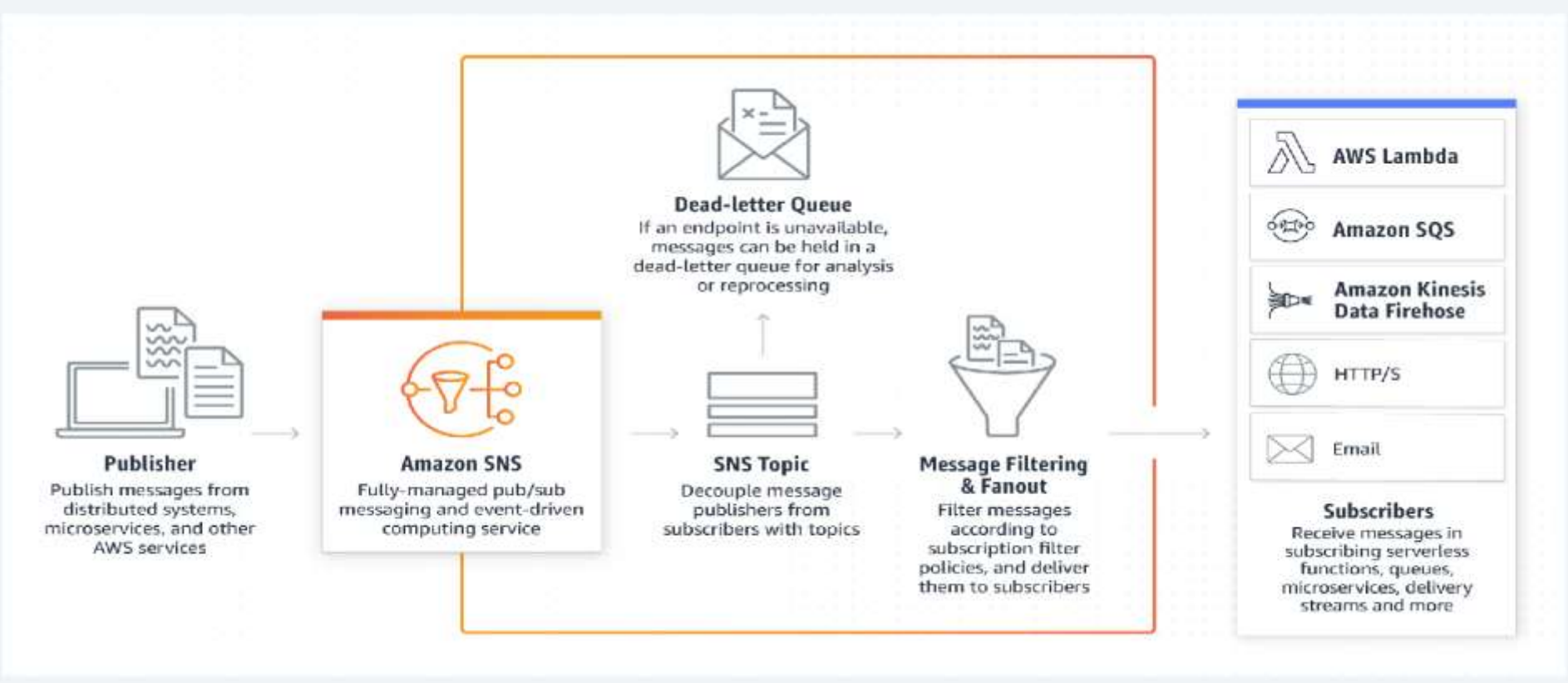
fands™

SQS and SNS

Messaging Fundamentals

- In modern cloud architecture, applications are decoupled into smaller, independent building blocks that are easier to develop, deploy and maintain.
- Messaging
  - Point to Point
    - Queue
    - Only Single Consumer
  - Publish Subscribe
    - Topic
    - Multiple Consumers

Message Topic



Message Queue

- Message queues allow asynchronous communication
- A message queue provides a lightweight buffer which temporarily stores messages, and endpoints that allow software components to connect to the queue in order to send and receive messages.
- The messages are usually small, and can be things like requests, replies, error messages, or just plain information.
- To send a message, a component called a producer adds a message to the queue. The message is stored on the queue until another component called a consumer retrieves the message and does something with it.

SQS and SNS

- Amazon Simple Queue Service (SQS) and Amazon Simple Notification Service (SNS) are both messaging services within AWS, which provide different benefits for developers.
- SNS allows applications to send time-critical messages to multiple subscribers through a “push” mechanism, eliminating the need to periodically check or “poll” for updates.

SNS

- SNS is a fully managed messaging service for both application-to-application (A2A) and application-to-person (A2P) communication.
- The A2A pub/sub functionality provides topics for high-throughput, push-based, many-to-many messaging between distributed systems, microservices, and event-driven serverless applications.
- Using Amazon SNS topics, your publisher systems can fanout messages to a large number of subscriber systems including SQS queues, Lambda functions and HTTPS endpoints, for parallel processing.
- The A2P functionality enables you to send messages to users at scale via SMS, mobile push, and email.

SQS

- Many producers and consumers can use the queue, but each message is processed only once, by a single consumer. For this reason, this messaging pattern is often called one-to-one, or point-to-point, communications.
- When a message needs to be processed by more than one consumer, message queues can be combined with Pub/Sub messaging in a fanout design pattern

Message Ordering and deduplication

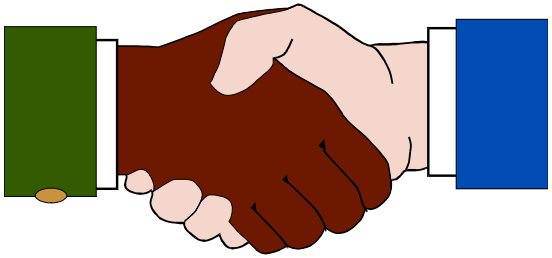
- Ensure accuracy with message ordering and deduplication
- Amazon SNS FIFO topics work with Amazon SQS FIFO queues to ensure messages are delivered in a strictly ordered manner and are only processed once (deduplicated). This enables you to maintain consistency when processing transactions across a single or multiple independent services where

Configure

- SNS
- SQS
- Fifo
- A small auto scaling demo with SNS

QUESTION / ANSWERS

THANKING YOU !



www.fandsindia.com