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

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.

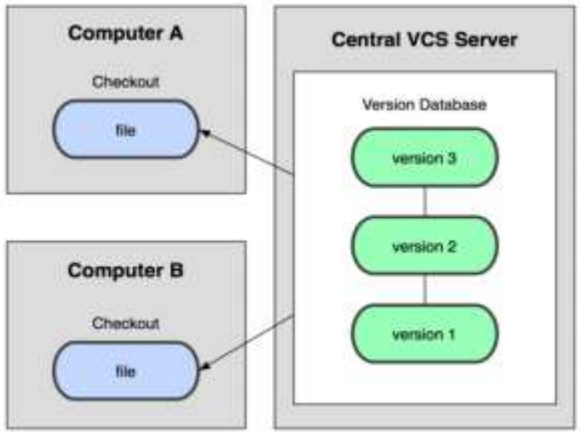
Version Control Systems

- 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?

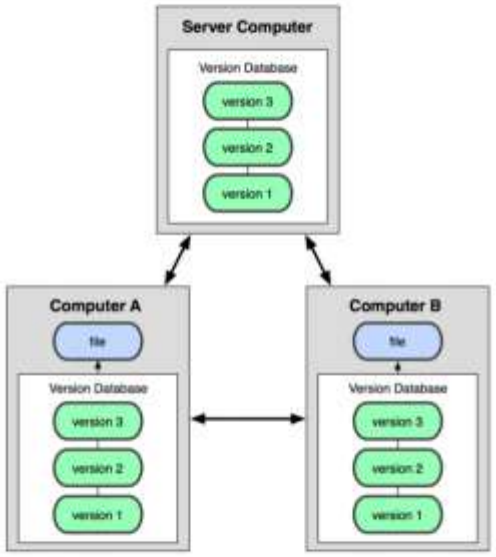
- For working by yourself:
  - Gives you a "time machine" for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes

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

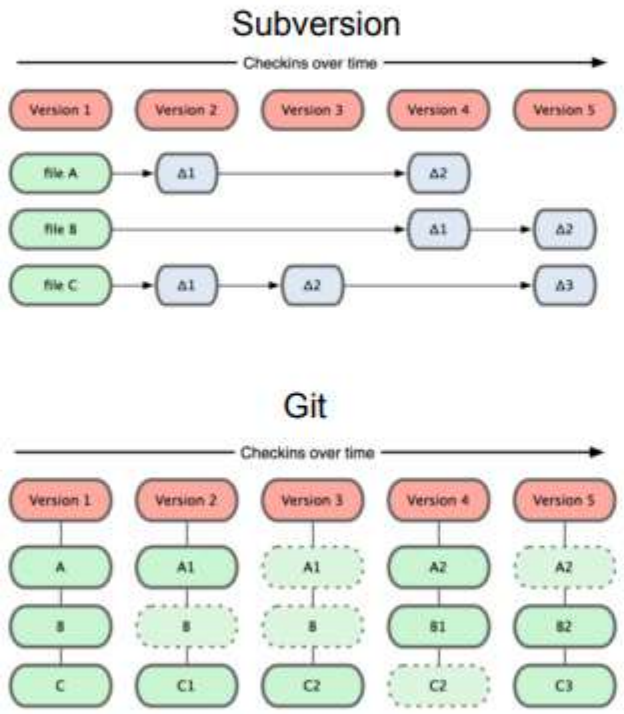
- Commit
  - Git thinks of its data like a set of snapshots of a mini file system.
  - Think of it as a save point during a video game.
- SHA
  - A SHA is basically an ID number for each commit.
  - Ex. E2adf8ae3e2e4ed40add75cc44cf9d0a869afeb6
- Branch
  - A branch is when a new line of development is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

Version Control Terminology

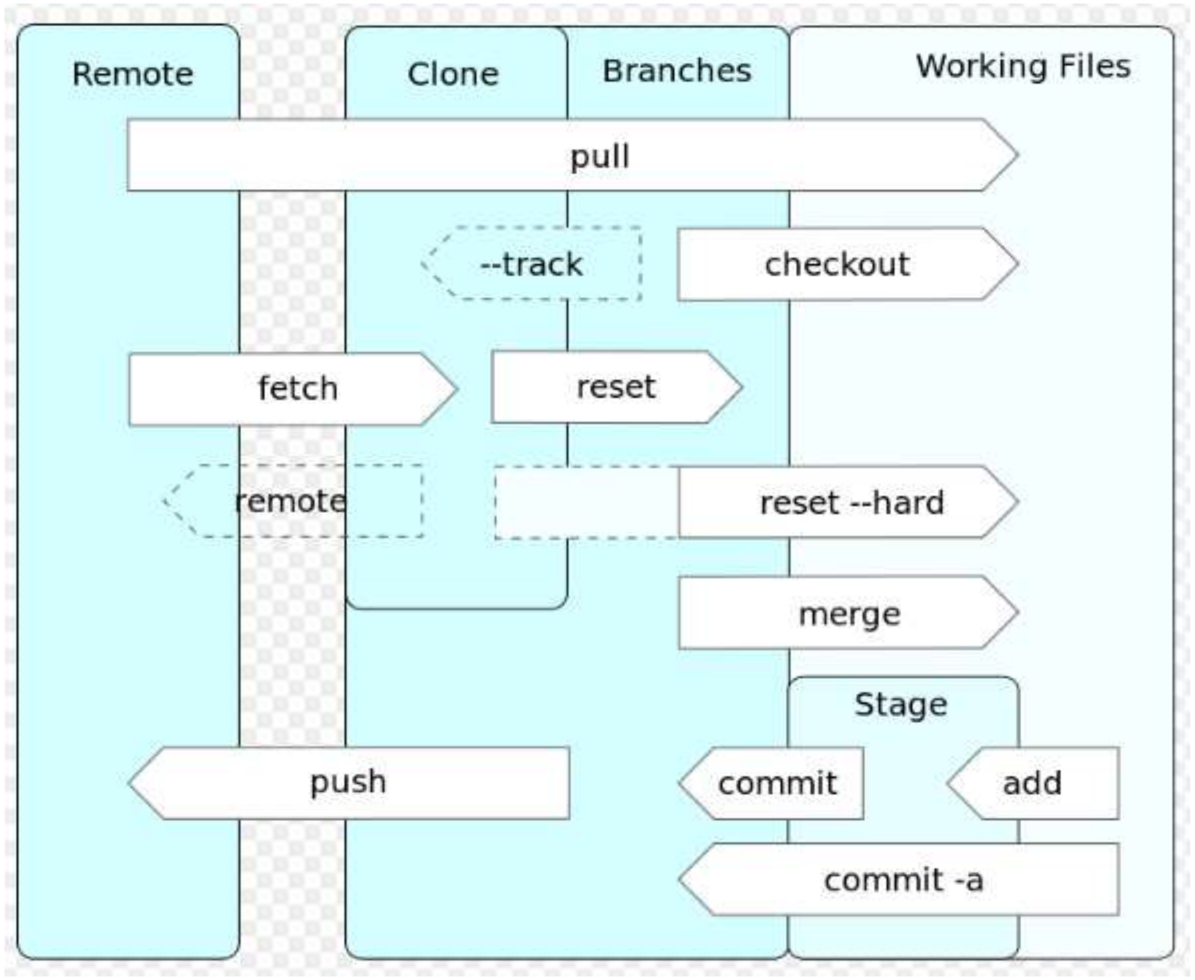
- Working Directory
  - files that you see in your computer's file system. When you open project files up on a code editor, you're working with files in the Working Directory.
- Checkout
  - Content in the repository has been copied to the Working Directory. Possible to checkout many things from a repository; 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.

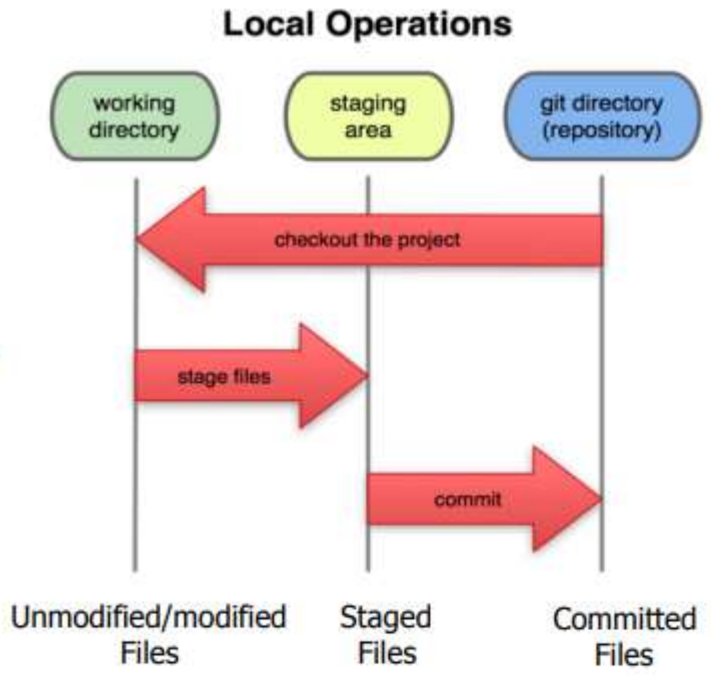


Git

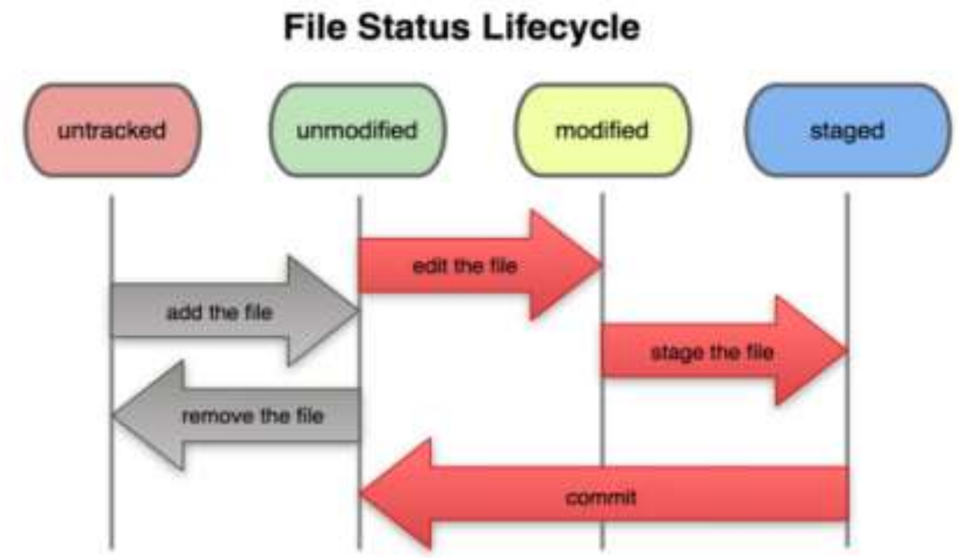

Remote
Clone
Branches
Working Files
pull
--track
checkout
fetch
reset
remote
reset --hard
merge
Stage
push
commit
add
commit -a

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.

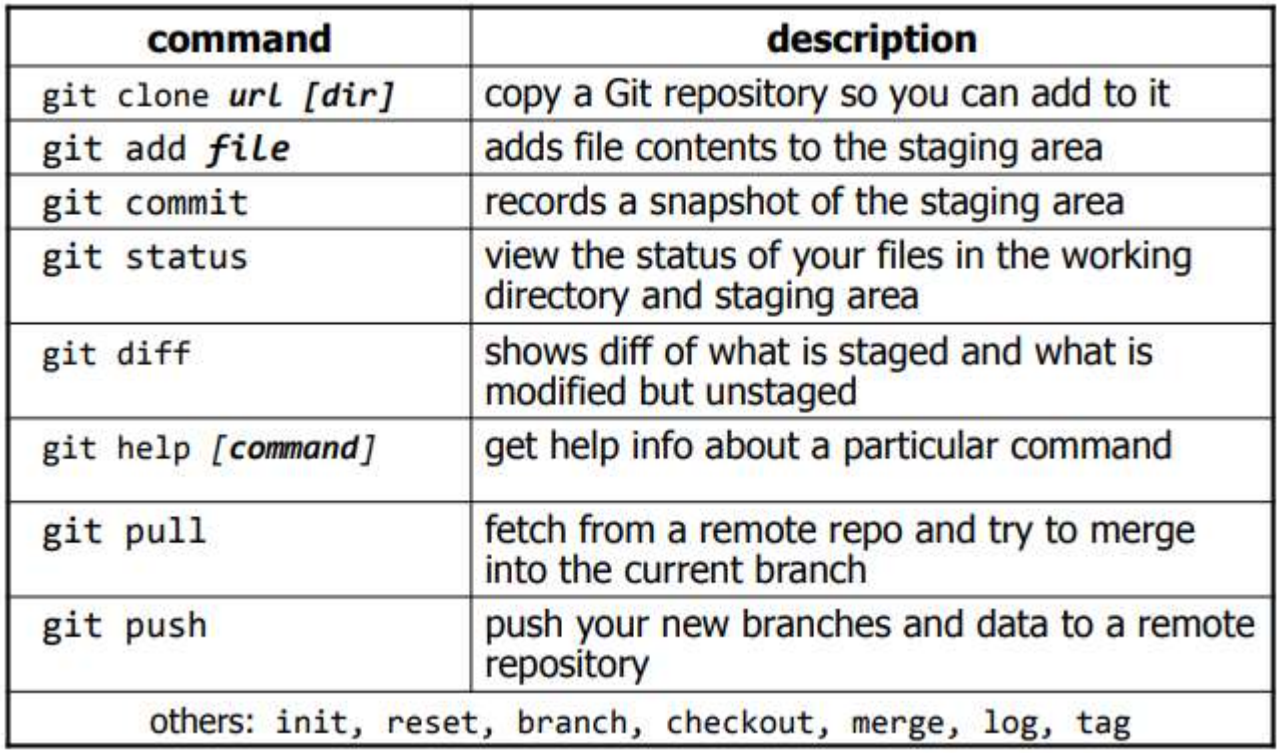
Initial Git configuration

- 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

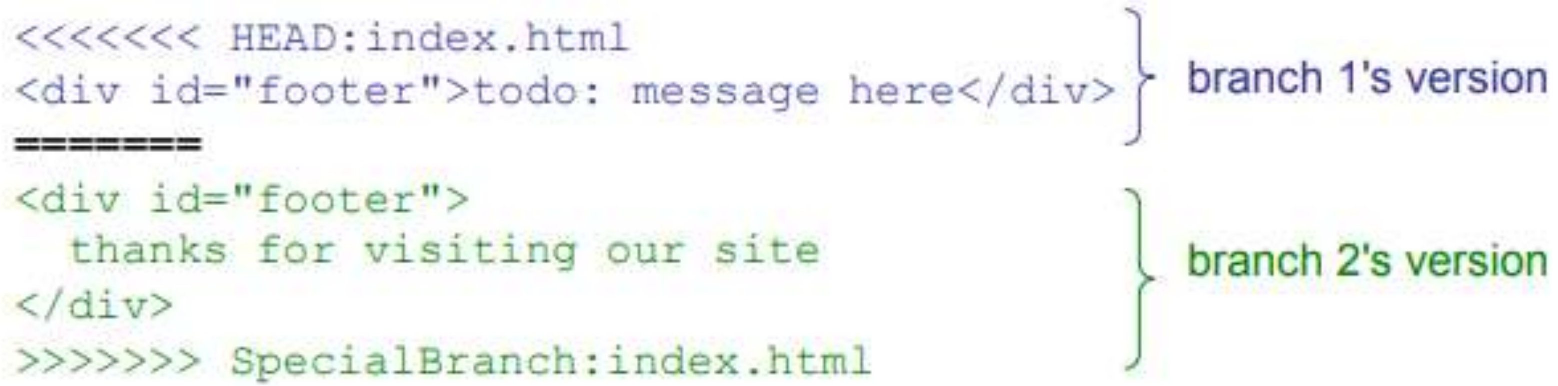
Branching and Merging

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

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

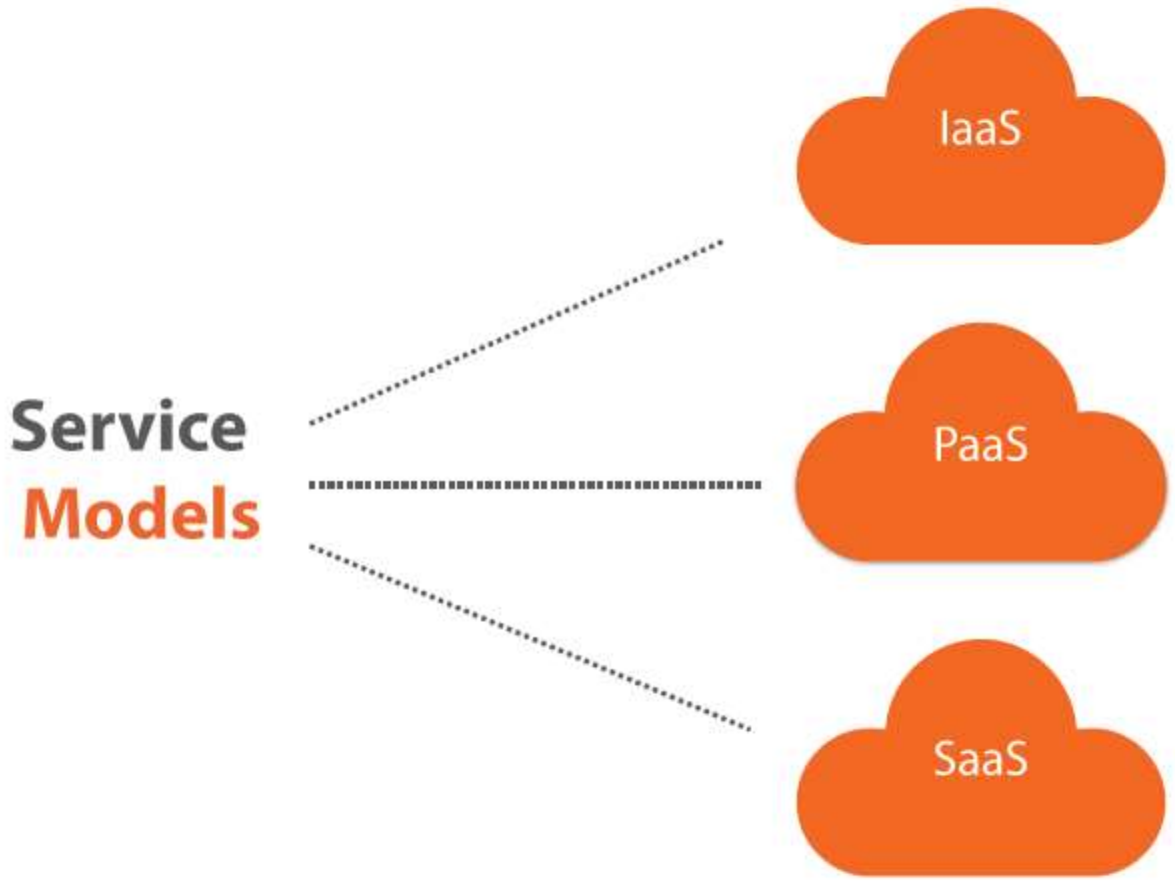
What is Cloud Computing?

- 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

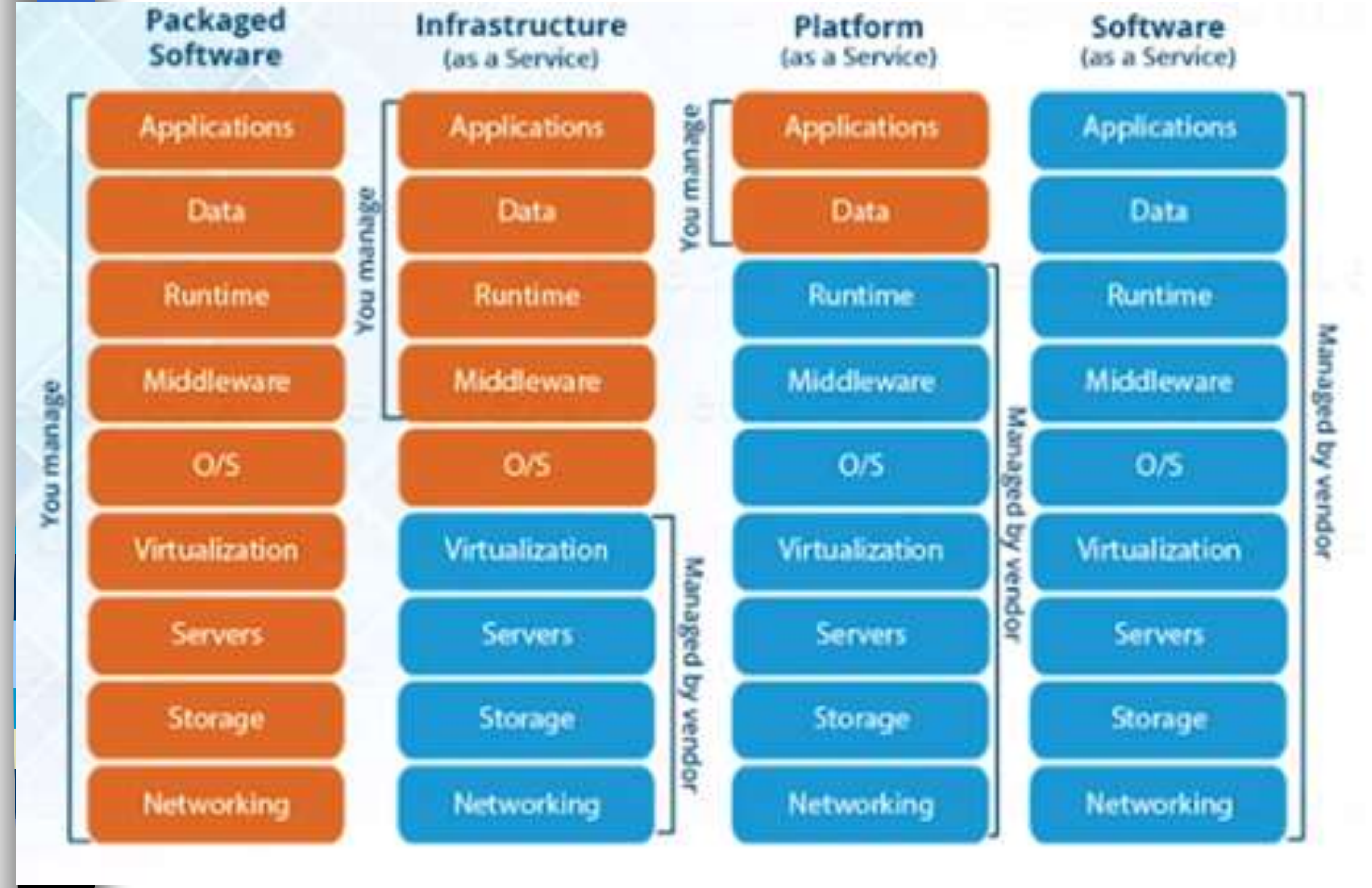
- On demand self services
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service
- Multi Tenacity

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
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
  – CDN 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

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?

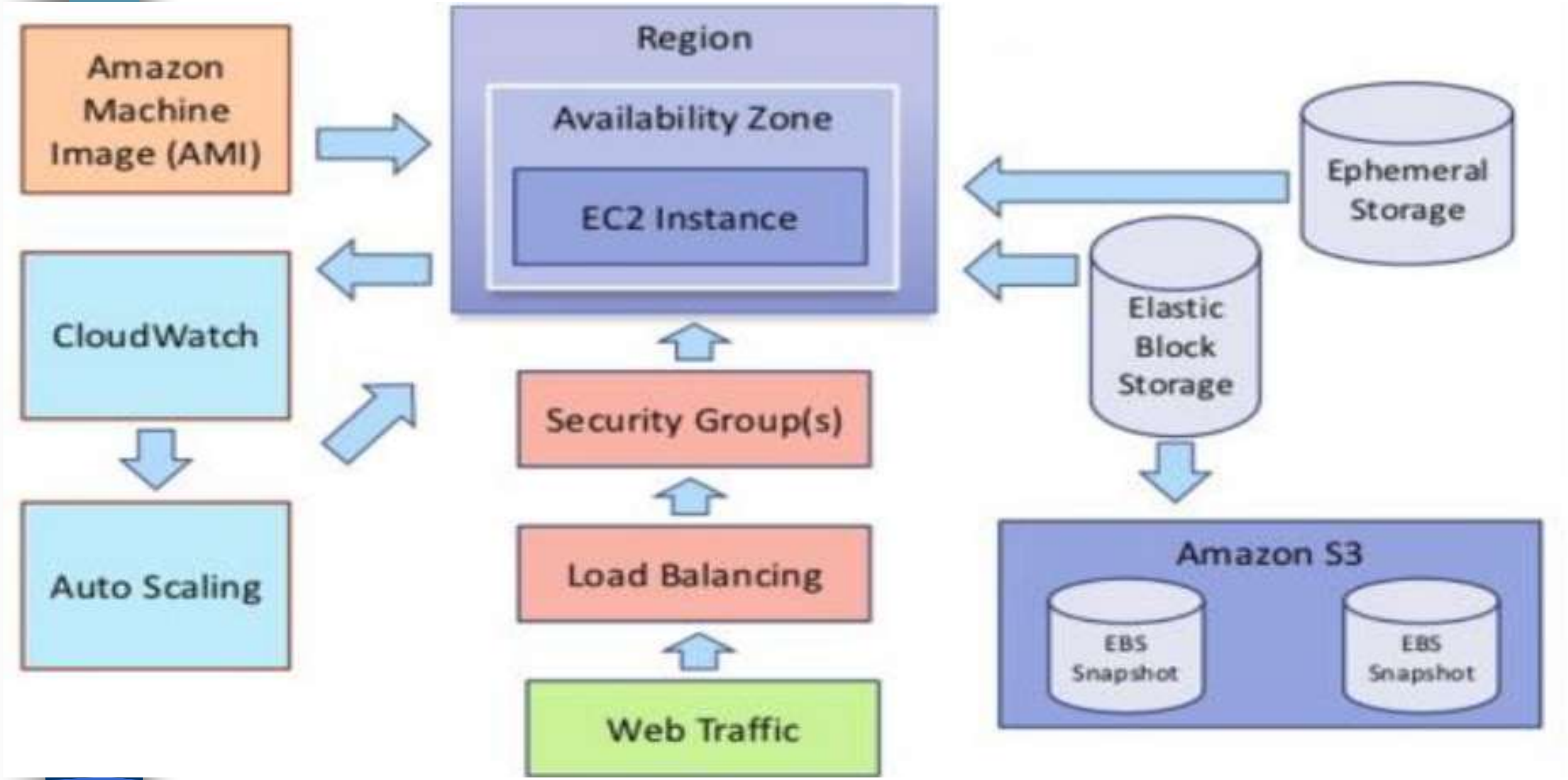
Run Applications

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)

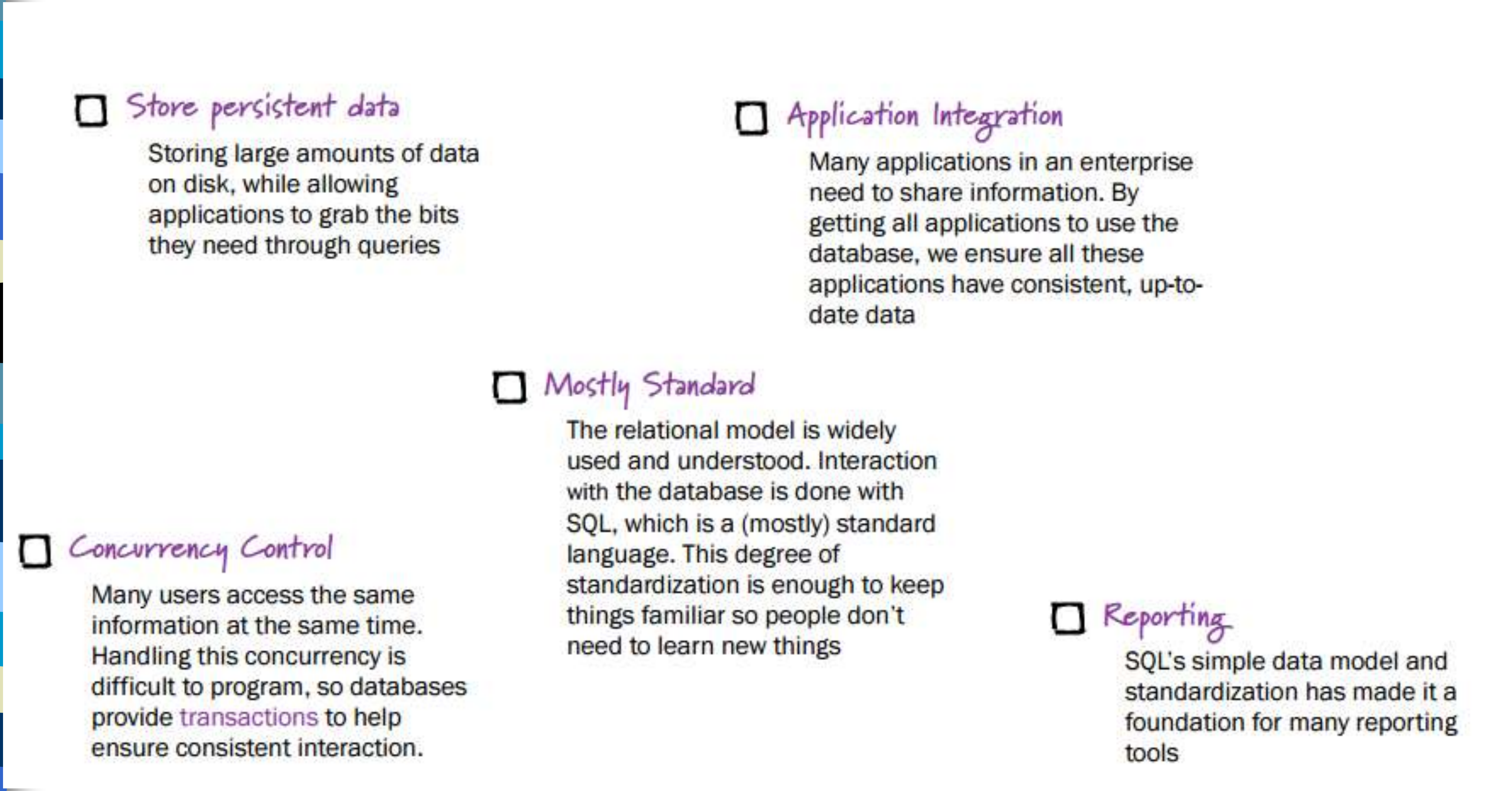
- Easy to set up, operate, and scale
- Provides cost-efficient and resizable capacity while automating time-consuming administration tasks such as hardware provisioning, database setup, patching and backups.
- Fast performance, high availability, security and compatibility
- Choose from Amazon Aurora,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

RDS

- Console
  - Create MySQL database
- MySQL Chrome plugin
  - Connect to MySQL Db
  - Create Table
  - Creat/Retrieve Records
- CLI
  - Describe

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.

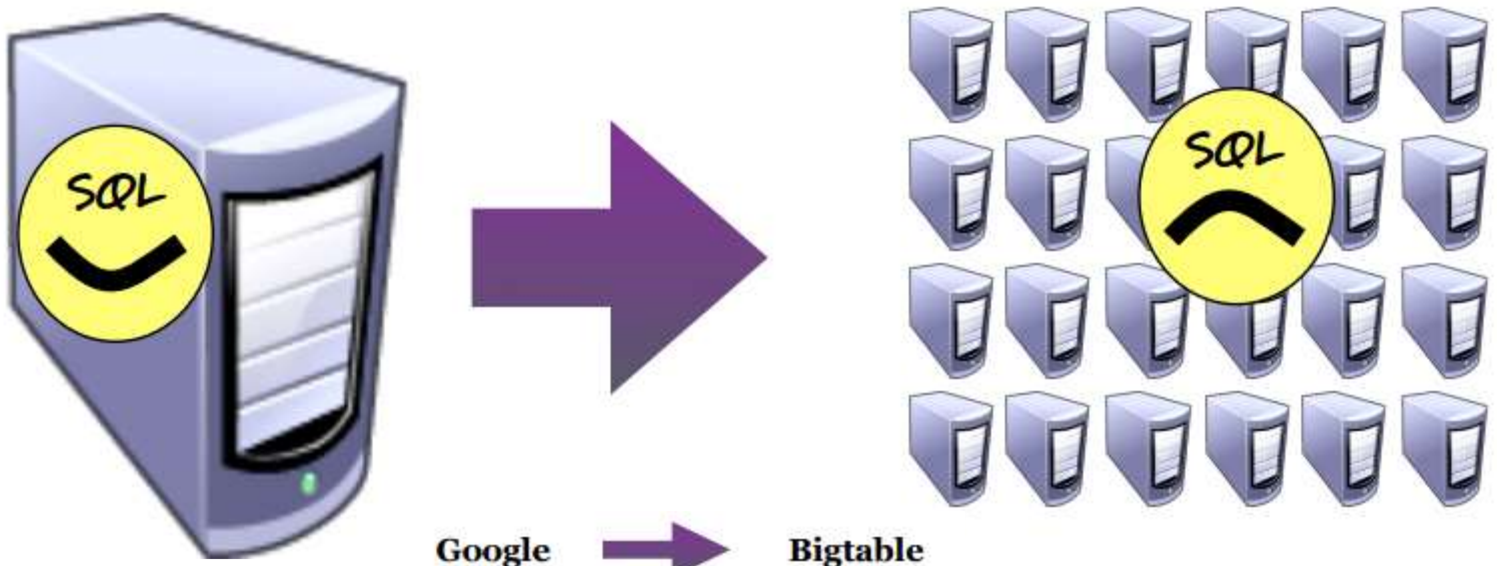
- Reporting

  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

mongoDB
{name: "mongo", type:"DB"}

apache CouchDB relax

riak

Cassandra

Neo4j
NOSQL for the Enterprise

APACHE HBASE

redis

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

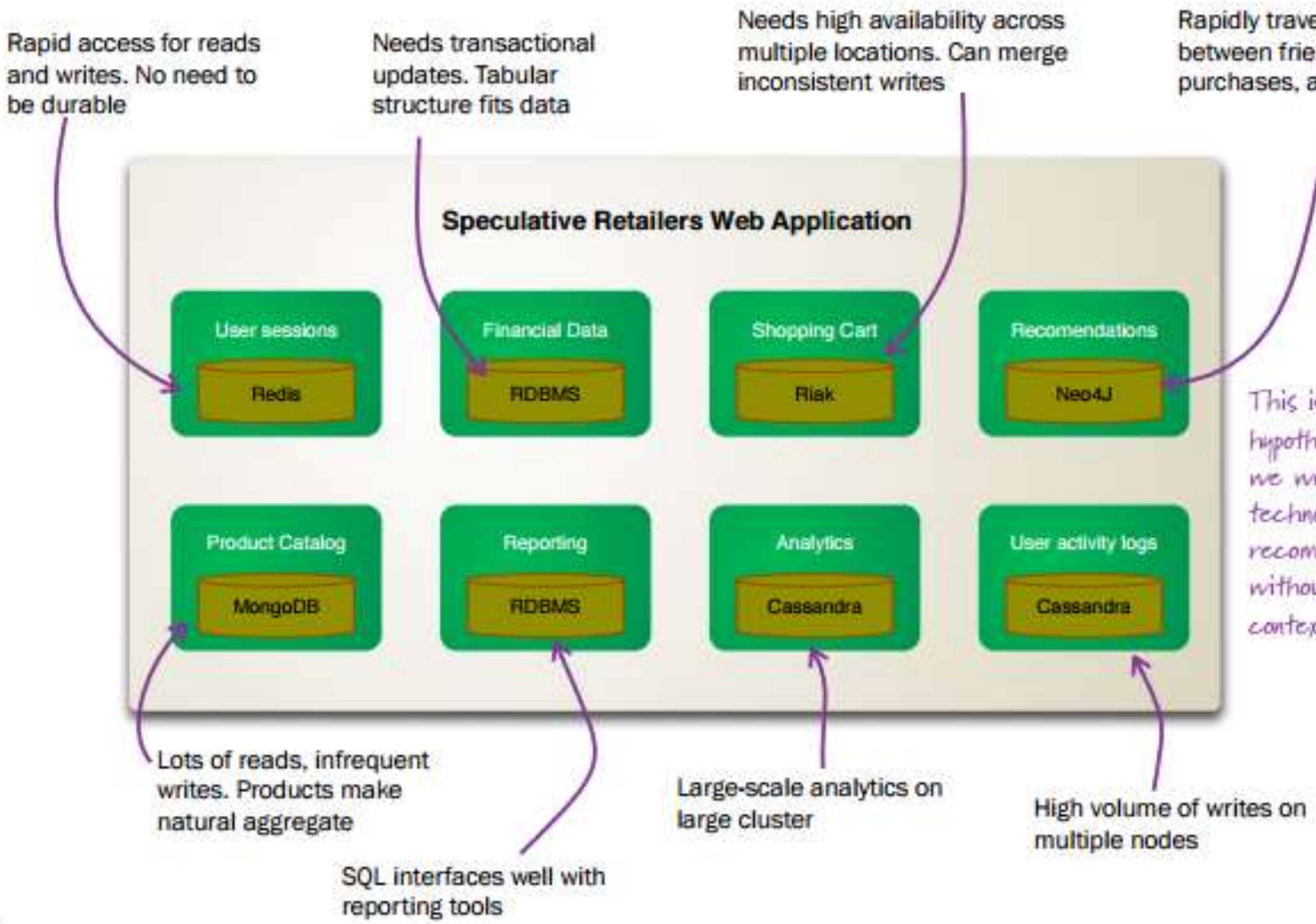
- Main Advantages
  - Reduce Development Drag
  - Embrace Large Scale
- This does not mean Relational is dead
  - the relational model is still relevant
  - ACID transactions
  - Tools
  - Familiarity

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?

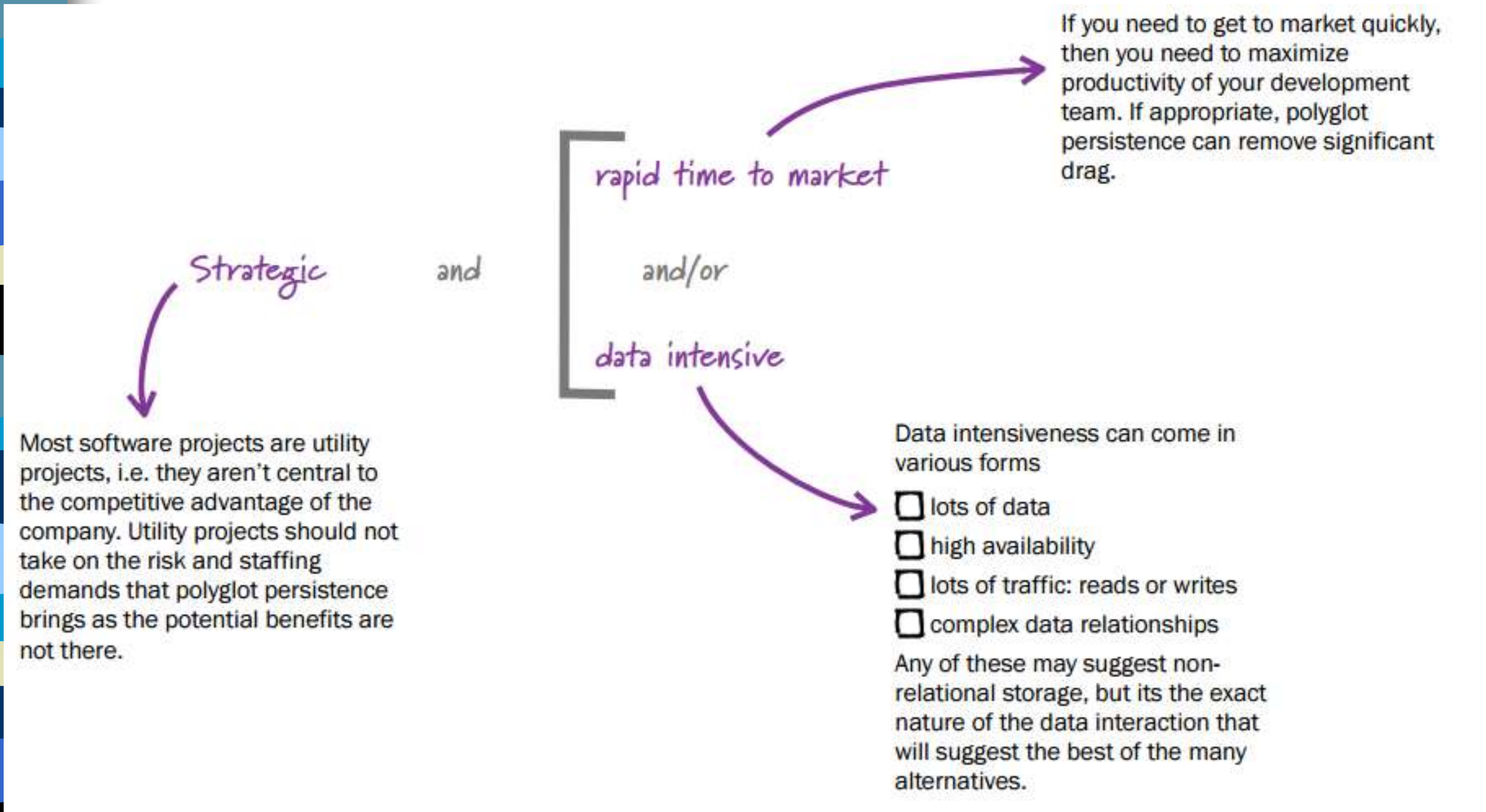
fands™

What might Polyglot Persistence look like?


Rapid access for reads and writes. No need to be durable
Needs transactional updates. Tabular structure fits data
Needs high availability across multiple locations. Can merge inconsistent writes
Rapidly traverse links between friends, product purchases, and ratings
Speculative Retailers Web Application
User sessions
Redis
Financial Data
RDBMS
Shopping Cart
Riak
Recomendations
Neo4J
Product Catalog
MongoDB
Reporting
RDBMS
Analytics
Cassandra
User activity logs
Cassandra
Lots of reads, infrequent writes. Products make natural aggregate
SQL interfaces well with reporting tools
Large-scale analytics on large cluster
High volume of writes on multiple nodes

This is a very hypothetical example, we would not make technology recommendations without more contextual information

Candidates for polyglot persistence?



If you need to get to market quickly, then you need to maximize productivity of your development team. If appropriate, polyglot persistence can remove significant drag.

Most software projects are utility projects, i.e. they aren't central to the competitive advantage of the company. Utility projects should not take on the risk and staffing demands that polyglot persistence brings as the potential benefits are not there.

Data intensiveness can come in various forms

- ☐ lots of data
- ☐ high availability
- ☐ lots of traffic: reads or writes
- ☐ complex data relationships

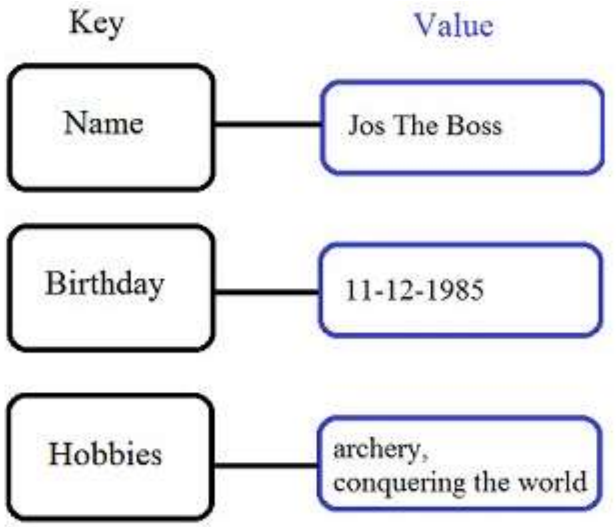
Any of these may suggest non-relational storage, but its the exact nature of the data interaction that will suggest the best of the many alternatives.

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.



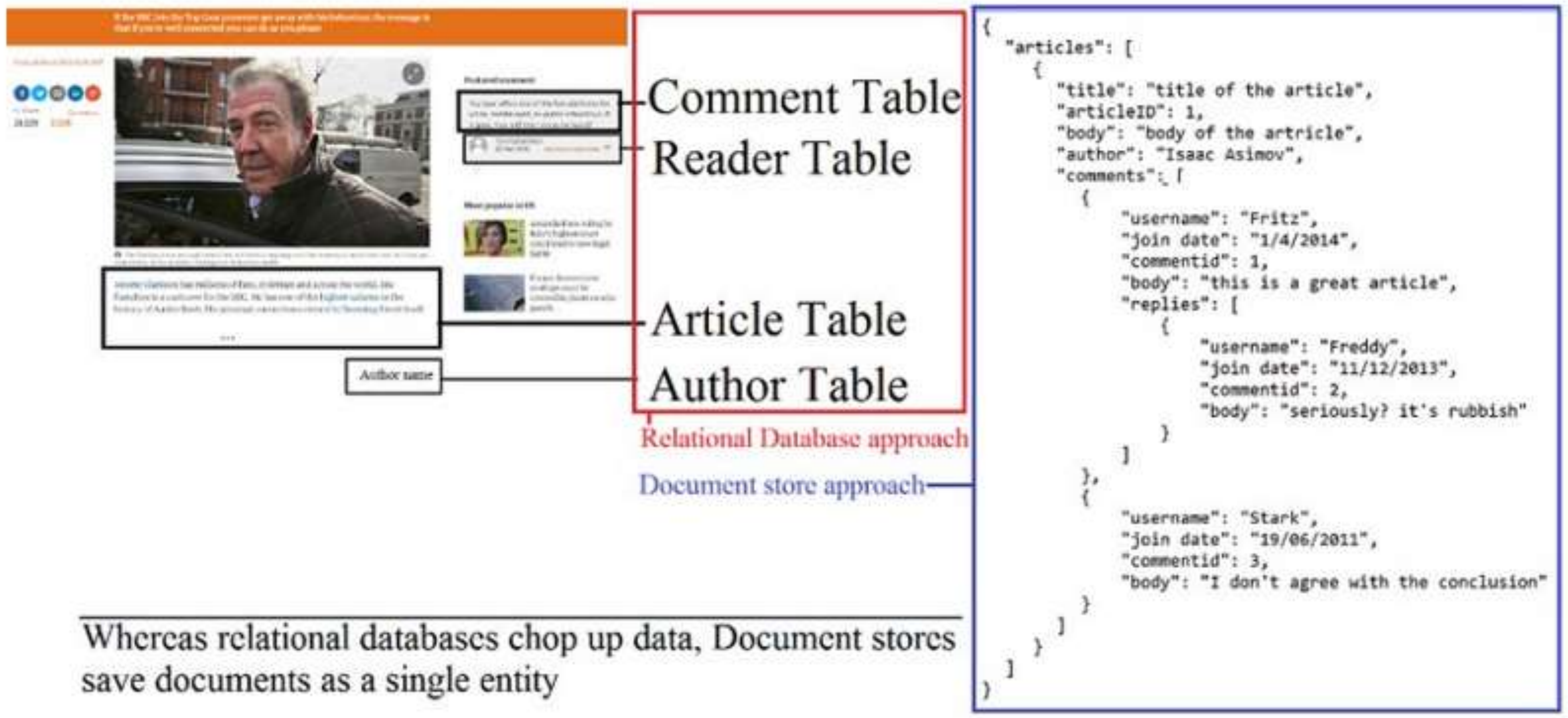
Redis, Voldemort, Riak, and Amazon's Dynamo.

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



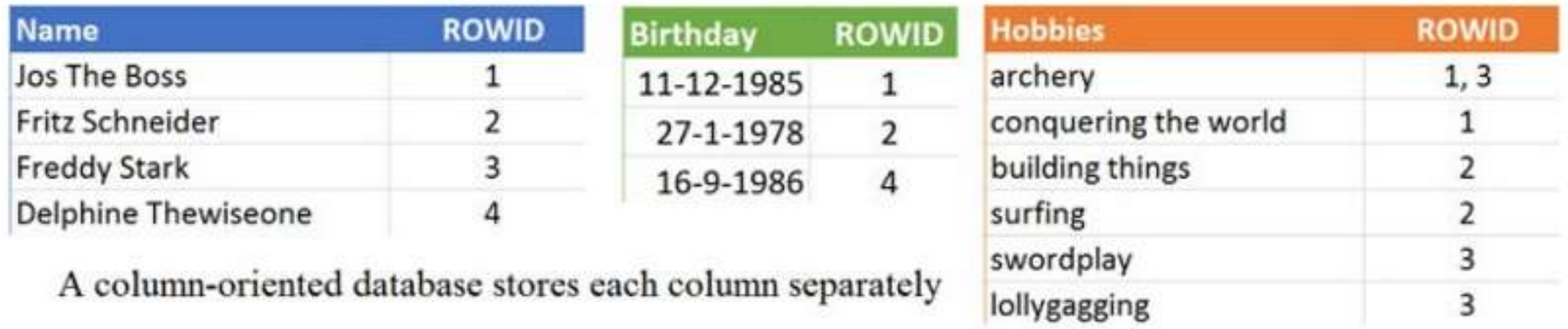
```
{
  "articles": [
    {
      "title": "title of the article",
      "articleID": 1,
      "body": "body of the artricle",
      "author": "Isaac Asimov",
      "comments": [
        {
          "username": "Fritz",
          "join date": "1/4/2014",
          "commentid": 1,
          "body": "this 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

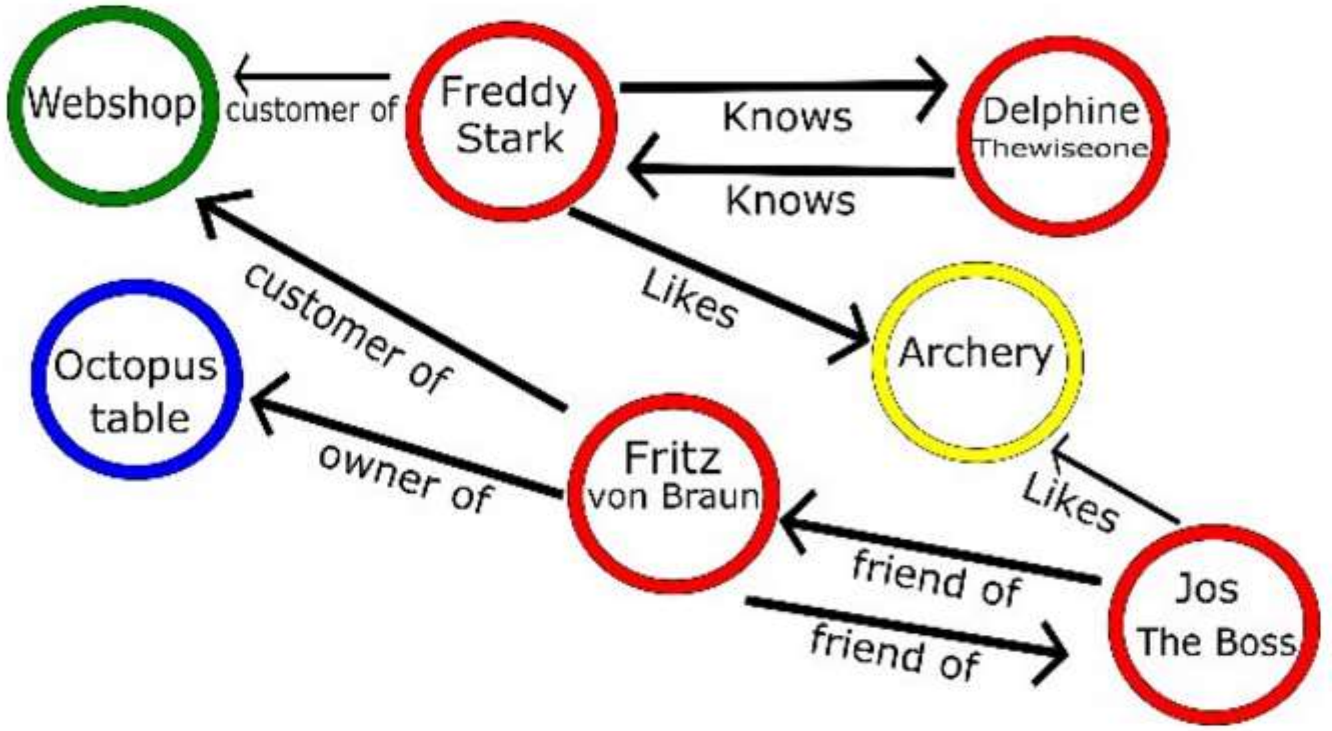
Hobbies	ROWID
archery	1, 3
conquering the world	1
building things	2
surfing	2
swordplay	3
lollygagging	3

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.

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

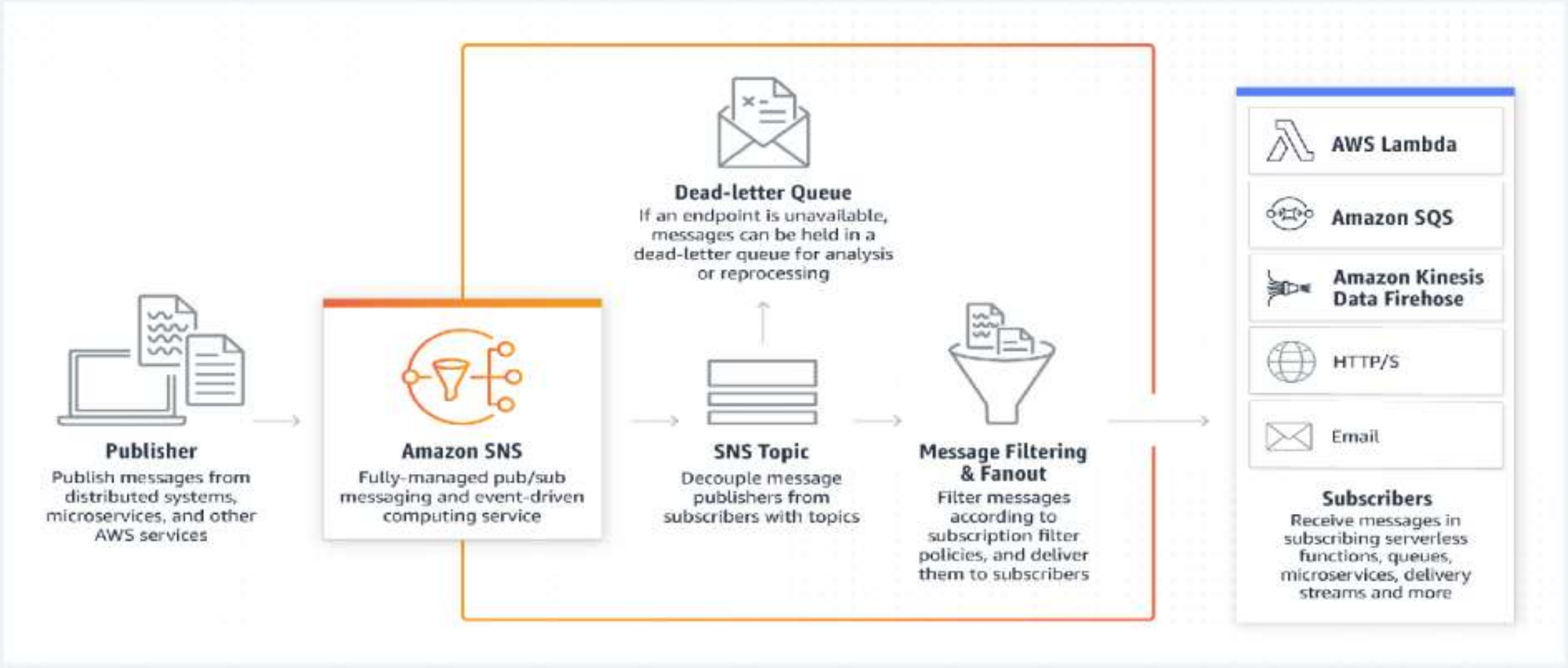
- Enable CloudWatch
  - – Check metrics
  - – Graphs
  - – Logs

SQS and SNS

Messaging Fundamentals

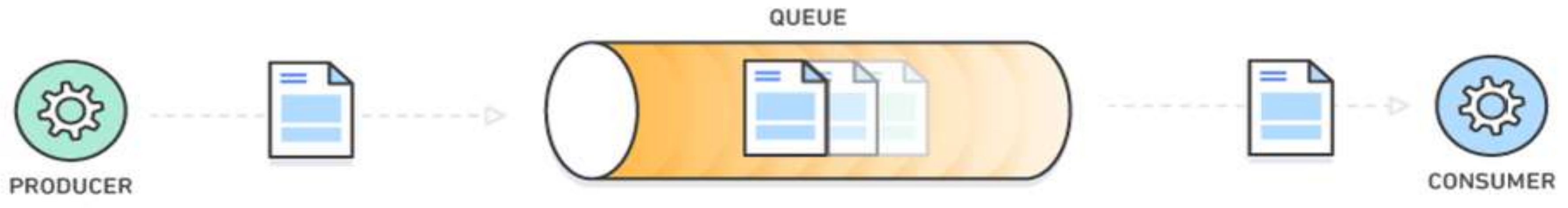
- In modern cloud architecture, applications are decoupled into smaller, independent building blocks that are easier to develop, deploy and maintain.
- Messaging
  - Point to Point
    - Queue
    - Only Single Consumer
  - Publish Subscribe
    - Topic
    - Multiple Consumers

Message Topic


Dead-letter Queue
If an endpoint is unavailable, messages can be held in a dead-letter queue for analysis or reprocessing
Publisher
Publish messages from distributed systems, microservices, and other AWS services
Amazon SNS
Fully-managed pub/sub messaging and event-driven computing service
SNS Topic
Decouple message publishers from subscribers with topics
Message Filtering & Fanout
Filter messages according to subscription filter policies, and deliver them to subscribers
AWS Lambda
Amazon SQS
Amazon Kinesis Data Firehose
HTTP/S
Email
Subscribers
Receive messages in subscribing serverless functions, queues, microservices, delivery streams and more

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

Lambda

Serverless on AWS

- Build & Run apps without thinking about servers
- Serverless is a way to describe the services, practices, and strategies that enable you to build more agile applications so you can innovate and respond to change faster.
- Infrastructure management tasks like capacity provisioning and patching are handled by AWS.
- Serverless services like AWS Lambda come with automatic scaling, built-in high availability, and a pay-for-value billing model.

Benefits

- Move from idea to market, faster
  - By eliminating operational overhead, your teams can release quickly, get feedback, and iterate to get to market faster.
- Lower your costs
  - pay-for-value billing, no need to over-provision
- Adapt at scale
  - Automatically scale from zero to peak demands,
- Build better applications, easier
  - Serverless applications have built-in service integrations, so you can focus on building your application instead of configuring it.

Serverless Services on AWS

- Compute
  - AWS Lambda
  - Amazon Fargate
- Data Store
  - Amazon S3
  - Amazon DynamoDB
  - Amazon RDS Proxy
  - Amazon Aurora Serverless
- Application Integration
  - Amazon EventBridge
  - AWS Step Functions
  - Amazon SQS
  - Amazon SNS
  - Amazon API Gateway
  - AWS AppSync

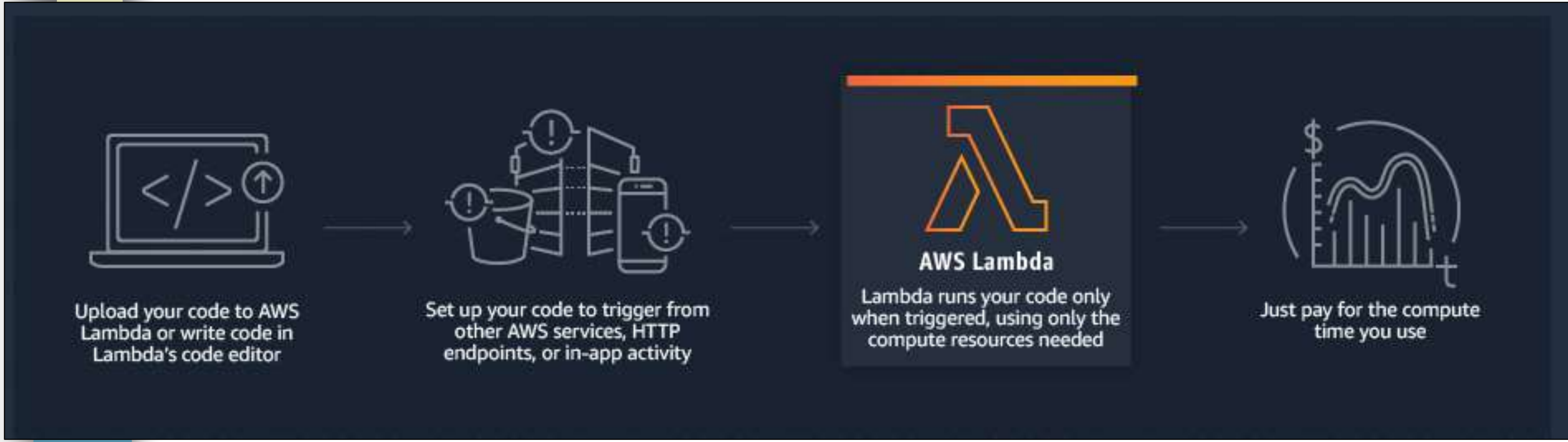
Lambda

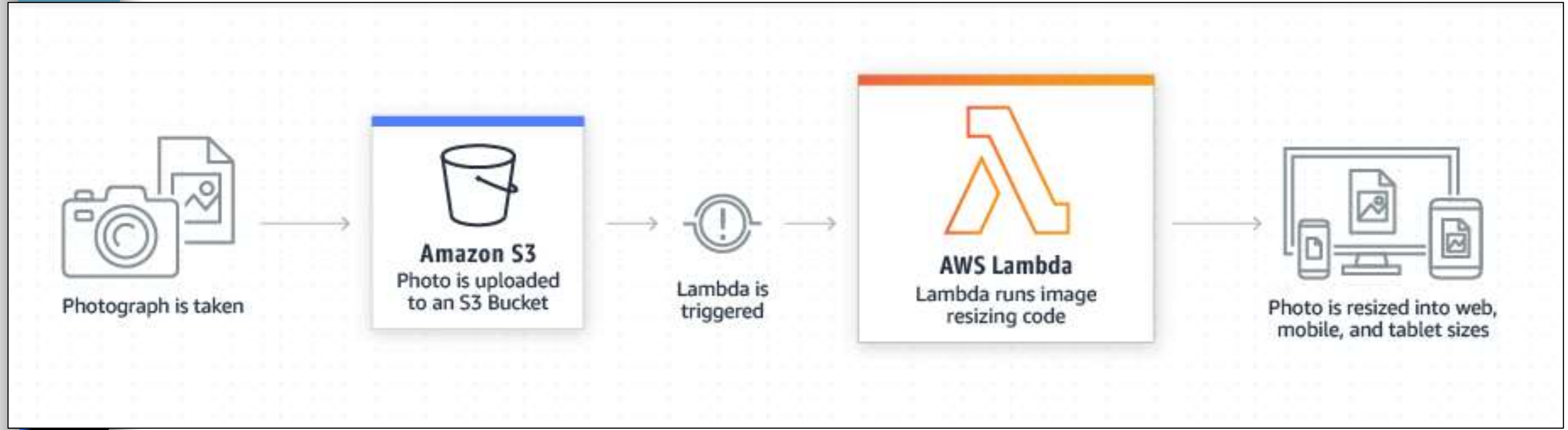
- Run code without provisioning or managing servers and pay only for the resources you consume
- Benefits
  - No servers to manage
  - Continuous scaling
  - Cost optimized with millisecond metering
  - Consistent performance at any scale

Lambda

- Run Code for virtually any type of application or backend service - with zero administration
- Just upload your code as a ZIP file or container image, and Lambda will allocate compute execution power and run your code based on the incoming request or event, for any scale of traffic.
- You can set up your code to automatically trigger from over 200 AWS services and SaaS applications or call it directly from any web or mobile app.
- Language Support for Node.js, Python, Go, Java, and more and use both serverless and container tools, such as AWS SAM or Docker CLI, to build, test, and deploy your functions.

Lambda


Upload your code to AWS Lambda or write code in Lambda's code editor
Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity
AWS Lambda
Lambda runs your code only when triggered, using only the compute resources needed
Just pay for the compute time you use


Photograph is taken
Amazon S3
Photo is uploaded to an S3 Bucket
Lambda is triggered
AWS Lambda
Lambda runs image resizing code
Photo is resized into web, mobile, and tablet sizes

What is a Lambda function?

- The code you run on AWS Lambda is called a “Lambda function.”
- After you create your Lambda function it is always ready to run as soon as it is triggered, similar to a formula in a spreadsheet.
- Each function includes your code as well as some associated configuration information, including the function name and resource requirements.

Lambda Functions

- Lambda functions are "stateless," with no affinity to the underlying infrastructure, so that Lambda can rapidly launch as many copies of the function as needed to scale to the rate of incoming events.
- After you upload your code to AWS Lambda, you can associate your function with specific AWS resources (e.g. S3 bucket, Amazon DynamoDB table, Amazon Kinesis stream, or Amazon SNS notification).
- When the resource changes, Lambda will execute your function and manage the compute resources as needed in order to keep up with incoming requests.

Built-in Fault Tolerance

- AWS Lambda maintains compute capacity across multiple Availability Zones in each region to help protect your code against individual machine or data center facility failures.
- Both AWS Lambda and the functions running on the service provide predictable and reliable operational performance.
- AWS Lambda is designed to provide high availability for both the service itself and for the functions it operates. No maintenance windows or scheduled downtimes.

Lambda Concepts

- Function
- Trigger
- Event
- Execution environment
- Deployment package
- Runtime
- Layer
- Extension
- Concurrency
- Qualifier

Lambda Concepts

- Function
  - A function is a resource that you can invoke to run your code in Lambda. A function has code to process the events that you pass into the function or that other AWS services send to the function.
- Trigger
  - A trigger is a resource or configuration that invokes a Lambda function. This includes AWS services that you can configure to invoke a function, applications that you develop, and event source mappings. An event source mapping is a resource in Lambda that reads items from a stream or queue and invokes a function

Lambda Concepts

- Event
  - An event is a JSON-formatted document that contains data for a Lambda function to process. The runtime converts the event to an object and passes it to your function code. When you invoke a function, you determine the structure and contents of the event.
- Execution environment
  - An execution environment provides a secure and isolated runtime environment for your Lambda function. An execution environment manages the processes and resources that are required to run the function. The execution environment provides lifecycle support for the function and for any extensions associated with your function.

Lambda Concepts

- Deployment package
  - You deploy your Lambda function code using a deployment package. Lambda supports two types of deployment packages:
    - A .zip file archive that contains your function code and its dependencies. Lambda provides the operating system and runtime for your function.
    - A container image that is compatible with the Open Container Initiative (OCI) specification. You add your function code and dependencies to the image. You must also include the operating system and a Lambda runtime.

Lambda Concepts

- Runtime
  - The runtime provides a language-specific environment that runs in an execution environment. The runtime relays invocation events, context information, and responses between Lambda and the function. You can use runtimes that Lambda provides, or build your own. If you package your code as a .zip file archive, you must configure your function to use a runtime that matches your programming language. For a container image, you include the runtime when you build the image.
- Layer
  - A Lambda layer is a .zip file archive that can contain additional code or other content. A layer can contain libraries, a custom runtime, data, or configuration files.

Lambda Concepts

- Extension
  - Lambda extensions enable you to augment your functions. For example, you can use extensions to integrate your functions with your preferred monitoring, observability, security, and governance tools.
- Concurrency
  - Concurrency is the number of requests that your function is serving at any given time. When your function is invoked, Lambda provisions an instance of it to process the event. When the function code finishes running, it can handle another request. If the function is invoked again while a request is still being processed, another instance is provisioned, increasing the function's concurrency.

Lambda Concepts

- Qualifier
  - When you invoke or view a function, you can include a qualifier to specify a version or alias. A version is an immutable snapshot of a function's code and configuration that has a numerical qualifier. For example, my-function:1. An alias is a pointer to a version that you can update to map to a different version, or split traffic between two versions. For example, my-function:BLUE. You can use versions and aliases together to provide a stable interface for clients to invoke your function.

Context

```
import time

def lambda_handler(event, context):
    print("Lambda function ARN:", context.invoked_function_arn)
    print("CloudWatch log stream name:", context.log_stream_name)
    print("CloudWatch log group name:",  context.log_group_name)
    print("Lambda Request ID:", context.aws_request_id)
    print("Lambda function memory limits in MB:", 
context.memory_limit_in_mb)
    # We have added a 1 second delay so you can see the time 
remaining in get_remaining_time_in_millis.
    time.sleep(1)
    print("Lambda time remaining in MS:", 
context.get_remaining_time_in_millis())
```

Logging

- Base Logging
  - print()
- Logging Library

```
import os
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES')
    logger.info(os.environ) logger.info('## EVENT')
    logger.info(event)
```

Versioning

- Use versions to manage deployment of your functions
- A function version includes the following information:
  - The function code and all associated dependencies.
  - The Lambda runtime that invokes the function.
  - All of the function settings, including the environment variables.
  - A unique Amazon Resource Name (ARN) to identify the specific version of the function.

Invocations

- Invoke using Lambda console, the Lambda API, the AWS SDK, the AWS CLI, and AWS toolkits.
- You can also configure other AWS services to invoke your function, or you can configure Lambda to read from a stream or queue and invoke your function.
- When you invoke a function, you can choose to invoke it synchronously or asynchronously.

Synchronous Vs Asynchronous

- Synchronous invocation
  - Wait for the function to process the event and return a response.
- Asynchronous invocation
  - Lambda queues the event for processing and returns a response immediately. For asynchronous invocation, Lambda handles retries and can send invocation records to a destination.

AWS CLI

- Synchronous
  - aws lambda invoke --function-name my-function --payload '{ "key": "value" }' response.json
- Asynchronous
  - aws lambda invoke --function-name my-function --invocation-type Event --payload '{ "key": "value" }' response.json

Concurrency for a Lambda function

- Two types of Concurrency Controls:
  - Reserved concurrency – Guarantees the maximum number of concurrent instances for the function. When a function has reserved concurrency, no other function can use that concurrency. There is no charge for configuring reserved concurrency for a function.
  - Provisioned concurrency –Initializes a requested number of execution environments so that they are prepared to respond immediately to your function's invocations. Note that configuring provisioned concurrency incurs charges to your AWS account.

Cold Vs Hot Container

- When running a serverless function, it will stay active (a.k.a., hot) as long as you're running it. Your container stays alive, ready and waiting for execution.
- After a period of inactivity, your cloud provider will drop the container, and your function will become inactive, (a.k.a., cold).
- A cold start happens when you execute an inactive function. The delay comes from your cloud provider provisioning your selected runtime container and then running your fn.

Temporary storage with /tmp

- The Lambda execution environment provides a file system for your code to use at /tmp. This space has a fixed size of 512 MB. The same Lambda execution environment may be reused by multiple Lambda invocations to optimize performance. The /tmp area is preserved for the lifetime of the execution environment and provides a transient cache for data between invocations. Each time a new execution environment is created, this area is deleted.

```
import os, zipfile
os.chdir('/tmp')
with zipfile.ZipFile(myzipfile, 'r') as zip:
    zip.extractall()
```

API Gateway

- Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.
- APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services.
- Create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications.
- API Gateway supports containerized and serverless workloads & web applications.

Benefits of API Gateway

- Efficient API development
- Easy Monitoring
- Performance at any scale
- Flexible security controls
- Cost savings at scale
- RESTful API options

Exposing Lambda function as API Gateway

Step Functions

Step Functions

- Serverless function orchestrator that makes it easy to sequence AWS Lambda functions and multiple AWS services into business-critical applications.
- Through its visual interface, you can create and run a series of checkpointed and event-driven workflows that maintain the application state. The output of one step acts as an input to the next.
- Each step in your application executes in order, as defined by your business logic.

Main Features

- Sequencing
- Error Handling
- Retry Logic
- Branching
- Parallel Paths
- Human Interaction
- Retaining state across function calls
- Tracing

Workflow Types

- Step Functions has two workflow types.
- Standard workflows
  - have exactly-once workflow execution and can run for up to one year.
  - Ideal for long-running, auditable workflows, as they show execution history and visual debugging
- Express workflows
  - have at-least-once workflow execution and can run for up to five minutes.
  - Ideal for high-event-rate workloads, such as streaming data processing and IoT data ingestion.

Standard and Express workflows

- Standard workflows
  - 2,000 per second execution rate
  - 4,000 per second state transition rate
  - Priced per state transition
  - Shows execution history and visual debugging
  - Supports all service integrations and patterns
- Express workflows
  - 100,000 per second execution rate
  - Nearly unlimited state transition rate
  - Priced per number and duration of executions
  - Sends execution history to CloudWatch
  - Supports all service integrations and most patterns

Use Cases

- Use case #1: Function orchestration
- Use case #2: Branching
- Use case #3: Error handling
- Use case #4: Human in the loop
- Use case #5: Parallel processing
- Use case #6: Dynamic parallelism

Service Integrations

- Request a response (default)
  - Call a service, and let Step Functions progress to the next state after it gets an HTTP response.
- Run a job (.sync)
  - Call a service, and have Step Functions wait for a job to complete.
- Wait for a callback with a task token (.waitForTaskToken)
  - Call a service with a task token, and have Step Functions wait until the task token returns with a callback.

States

- Individual states can make decisions based on their input, perform actions, and pass output to other states.
- In AWS Step Functions you define your workflows in the Amazon States Language.
- The Step Functions console provides a graphical representation of that state machine to help visualize your application logic.

State Functions

- Task
  - Do some work in your state machine (a Task state)
- Choice
  - Make a choice between branches of execution
- Fail or Succeed
  - Stop an execution with a failure or success
- Pass
  - Simply pass its input to its output or inject some fixed data
- Wait
  - Provide a delay for a certain amount of time or until a specified time/date
- Parallel
  - Begin parallel branches of execution
- Map
  - Dynamically iterate steps

Pass State

```
{
  "Comment": "A Hello World example for  Pass states",
  "StartAt": "Hello",
 "States": {
    "Hello": {
      "Type": "Pass",
      "Result": "Hello",
      "Next": "World"
    },
    "World": {
      "Type": "Pass",
      "Result": "World",
      "End": true } } }
```

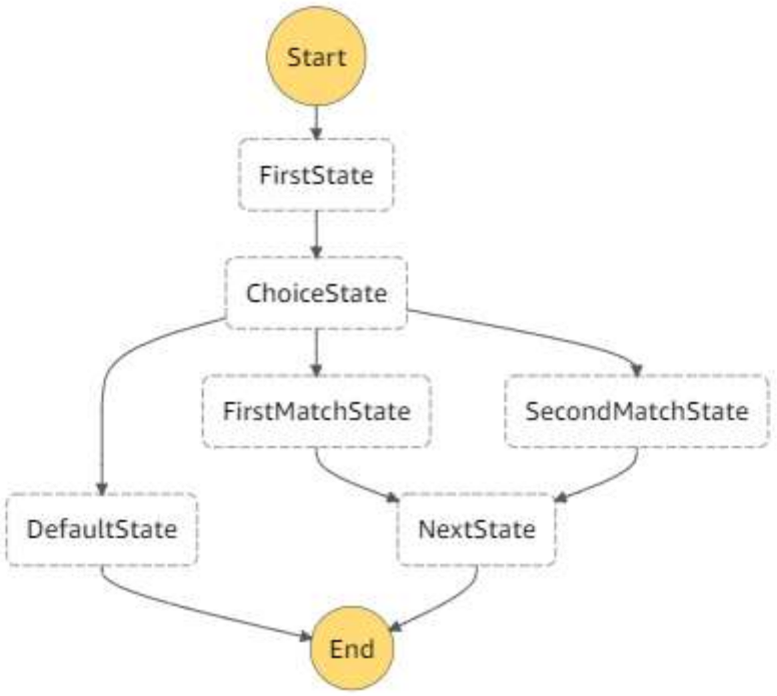
Map – Array Processing

```
{
  "Comment": "..",
  "StartAt": "Map",
  "States": {
    "Map": {
      "Type": "Map",
      "ItemsPath": "$.array",
      "ResultPath": "$.array",
      "MaxConcurrency": 2,
      "Next": "Final State",
      "Iterator": {
        "StartAt": "Pass",
}
```

```
"States": {
          "Pass": {
            "Type": "Pass",
            "Result": "Done!",
            "End": true
          }
        }
      }
    },
    "Final State": {
      "Type": "Pass",
      "End": true
    }
  }
```

Choice - Branching

```
"ChoiceState": {
   "Type" : "Choice",
   "Choices": [
   {
    "Variable": "$.foo",
    "NumericEquals": 1,
    "Next": "FirstMatchState"
     },
     {
     "Variable": "$.foo",
     "NumericEquals": 2,
     "Next": "SecondMatchState"
       }
     ],
     "Default": "DefaultState"
   }
```



Wait State – Timed wait

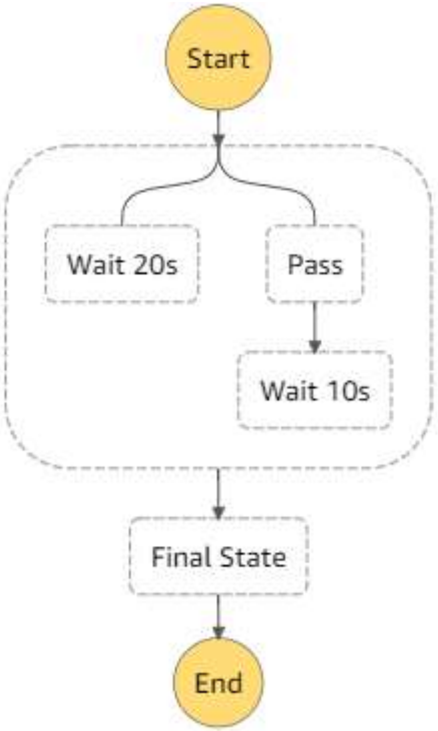
```
"wait_using_seconds": {
     "Type": "Wait",
     "Seconds": 10,
  },

"wait_using_timestamp": {
     "Type": "Wait",
     "Timestamp": "2015-
09-04T01:59:00Z",
  },
```

```
"wait_using_timestamp_p
ath": {
     "Type": "Wait",
     "TimestampPath":
"$.expirydate",
  },
"wait_using_seconds_path
": {
     "Type": "Wait",
     "SecondsPath":
"$.expiryseconds",
  },
```

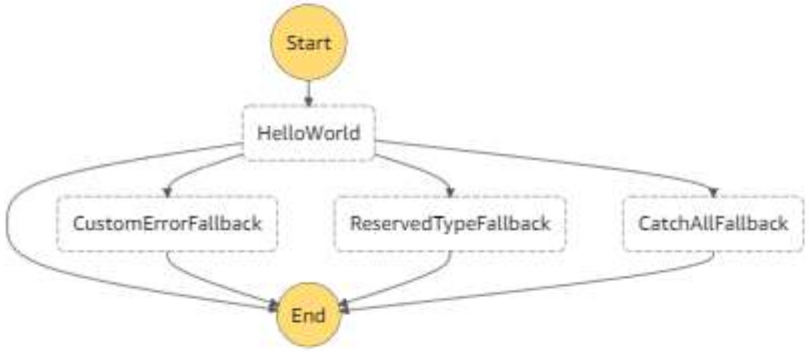
Parallel – Parallel Branches

```
"Parallel": {
      "Type": "Parallel",
      "Next": "Final State",
      "Branches": [
        { "StartAt": "Wait 20s",
          "States": { "Wait 20s": {
              "Type": "Wait",
              "Seconds": 20,
              "End": true      } } },
        { "StartAt": "Pass",
          "States": {
             "Pass": {
              "Type": "Pass",
              "Next": "Wait 10s“     },
            "Wait 10s": {
              "Type": "Wait",
              "Seconds": 10,
              "End": true            }
```



Catch – Handling Errors

```
"HelloWorld": {
 "Type": "Task",
 "Resource": "lambdaArn",
 "Catch": [     {
 "ErrorEquals":["CustomError"],
 "Next": "CustomErrorFallback“
},
 {
"ErrorEquals":["States.TaskFailed"],
 "Next": ReservedTypeFallback"
},
 {    "ErrorEquals": ["States.ALL"],
       "Next": "CatchAllFallback“ }
   ],
     "End": true
   },
```



Periodically Starting

- CloudWatch Events
  – Delivers a near real-time stream of system events that describe changes in AWS resources
- CloudWatch Events
  – Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console

Periodically Starting

- Start state machine execution after every one minute
  - Create State Machine
  - Select State machine
    - Actions
    - Create event bridge(CloudWatch) rule
    - After every 1 minute
  - Check
    - State Machine Executions

Exposing as API Gateway

- You can use Amazon API Gateway to associate your AWS Step Functions APIs with methods in an API Gateway API. When an HTTPS request is sent to an API method, API Gateway invokes your Step Functions API actions.
- Start a Step Functions execution by calling StartExecution and DescribeExecution to get the result.

Amazon S3 Events as trigger

- Amazon EventBridge to execute an AWS Step Functions state machine in response to an event or on a schedule.

Activities

- Activities are an AWS Step Functions feature that enables you to have a task in your state machine where the work is performed by a worker that can be hosted on Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Container Service (Amazon ECS), AWS Lambda, mobile devices—basically anywhere.

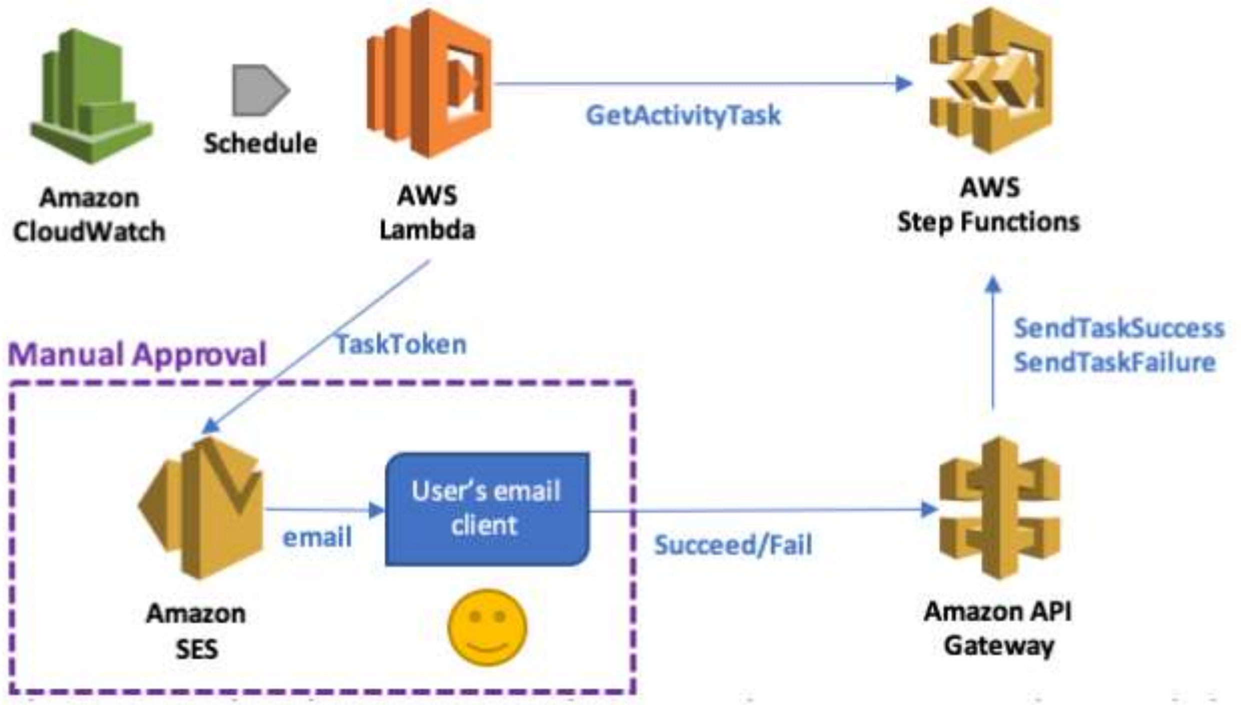
Overview

- In AWS Step Functions, activities are a way to associate code running somewhere (known as an activity worker) with a specific task in a state machine.
- You can create an activity using the Step Functions console, or by calling CreateActivity. This provides an Amazon Resource Name (ARN) for your task state. Use this ARN to poll the task state for work in your activity worker.

Human Interaction

- The most important feature of creating workflows with Step Functions is the ability to control its execution with external input. We can pause, continue, or stop the workflows whenever we want. This is especially important when we need to have human interactions with the workflow.

Serverless Manual Approval



https://aws.amazon.com/blogs/compute/implementing-serverless-manual-approval-steps-in-aws-step-functions-and-amazon-api-gateway/

Maintaining

- Logging and monitoring
  - Cloud Watch
    - Metrics
    - Alarms
  - EventBridge Events
  - AWS CloudTrail
- Logging using CloudWatch Logs
  - X-Ray

X-Ray

- Use AWS X-Ray to visualize the components of your state machine, identify performance bottlenecks, and troubleshoot requests that resulted in an error.
- State machine sends trace data to X-Ray, and X-Ray processes the data to generate a service map and searchable trace summaries.

X-Ray

- This gives you a detailed overview of an entire Step Functions request. Step Functions will send traces to X-Ray for state machine executions, even when a trace ID is not passed by an upstream service. You can use an X-Ray service map to view the latency of a request, including any AWS services that are integrated with X-Ray. You can also configure sampling rules to tell X-Ray which requests to record, and at what sampling rates, according to criteria that you specify.

AWS SAM

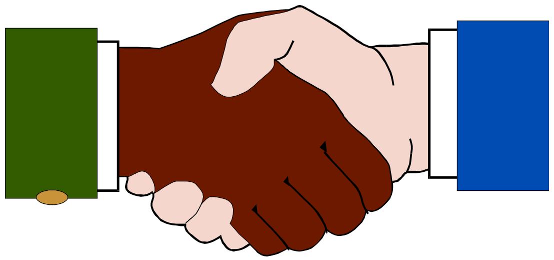
- Install SAM CLI
- Download, build, and deploy a sample AWS SAM application that contains an AWS Step Functions state machine. This application creates a mock stock trading workflow which runs on a pre-defined schedule

fands™

QUESTION / ANSWERS

www.fandsindia.com

THANKING YOU !



www.fandsindia.com