Distributed Systems -> Grid Computing -> Cloud Computing

Distributed Systems

- A network that consists of autonomous computers that are connected using a distribution middleware. They help in sharing different resources and capabilities to provide users with a single and integrated coherent network.
- Initially, computing tasks were spread over multiple machines for performance, scalability, and fault tolerance.
- **Pros**: Improved performance, scalability, resource sharing.
- **Cons**: Complexity in data consistency and system management, network dependency.

Challenges of Distributed Systems

- **Heterogeneity**: Diverse software, hardware, network technologies, and programming languages can lead to deployment issues and indirectly affect scalability. Data heterogeneity - JSON, JPG, Tables etc.
- **Openness**: Openness can lead to difficulty in system improvement as it might require major reconfiguration or redevelopment.
- **Concurrency**: Control over shared resources can be an issue, leading to possible data conflicts or data inconsistency. Solutions include locking mechanisms or distributed algorithms.
- **Scalability**: The system must have the ability to change its size and scale. Geographical partitions in a network can affect this.
- **Fault Tolerance**: Nodes in distributed systems can fail, so systems need to have mechanisms to cope with faults and maintain functionality. More nodes means less mean time to failure (MTTF) and handling more diverse faults.
- **Transparency**: The system should appear to users as a single entity rather than a collection of individual components.
- **Security**: Security issues arise from the open nature of the distributed system, including data access, communication security, and system integrity.
- **Performance**: Overhead because of communication between nodes, latency, resource waiting may significantly affect efficiency.

Erroneous Assumptions of Distributed Systems

- **Network reliability**: Assumes network is always reliable, which isn't the case due to possible data loss or corruption.
- **Latency**: Assumes zero latency, while in reality, data transmission delays may occur, especially over long distances.
- **Bandwidth**: Assumes infinite bandwidth, but sending large data can overwhelm the network as bandwidth is a limited resource.
- **Security**: Assumes a secure network, despite various potential attacks and data vulnerabilities.
- **Topology**: Assumes a stable network topology, ignoring possible changes in network nodes.
- **Administration**: Assumes a single administrator for large networks, while typically multiple administrators manage server configurations.
- **Transport cost**: Assumes zero cost, yet transmitting data over a network incurs costs in various forms.
- **Homogeneity**: Assumes a homogeneous network, even though networks consist of diverse protocols and hardware.
- **Time**: Assumes ubiquitous time, ignoring challenges with time synchronization across network nodes.

Grid Computing
- A distributed architecture of numerous computers connected together to form a grid to **perform higher-level computation.** This system involves resource sharing across a decentralised network to reach a common **computational goal.** It is frequently used to gather computing force to be a supercomputer in **scientific research field.**
- This emerged as a way to use under-utilised resources and connect machines over a network, often across organisational boundaries, focusing on large-scale computation problems.
- **Pros**: High-level computations, under-utilised resource usage, cost-effective.
- **Cons**: Requires special software, complex management, security concerns.

Challenges of Grid Computing
- Requires dedicated software for resource sharing, scheduling and security issues, complex management.

Cloud Computing
It's a concept involving numerous computers connected via a real-time communication network.
- **On-demand availabilit**y of system resources (data storage, computing power) **without the need for direct active management** by the user, leading to scalability and easier maintenance.
- A model for delivering IT services where resources are retrieved from the internet using web-based tools and applications, as opposed to direct server connections.
- Data and software packages are stored on servers and accessed via the internet.
- Cloud computing services operate on a '**Pay as you go**' model, which can be mentioned as a key characteristic in all cloud computing related discussions.
- It's like a commercial implementation of Grid computing, it provides abstract resources and operational services for multiple applications. Businesses moved to this. It's build from grid computing's technical framework.
- **Pros**: Easy access globally, scalability, cost effectiveness due to pay-as-you-go model, eliminating the need to manage physical resources.
- **Cons**: Potential for vendor lock-in, security and privacy concerns, regulatory compliance issues.

Challenges of Cloud Computing
- Dependence on service provider, data security, privacy, regulatory compliance issues, possible vendor lock-in.

Cloud Characteristics
- **On-demand self-service:**
  - Explanation: Cloud services can be used as needed, without requiring human interaction with service providers. Users can manage and adjust resources independently.
  - Scenario: This is useful when businesses need to quickly scale up resources to handle a surge in demand, such as a retail website during a big sale event.
- **Networked access:**
  - Explanation: Cloud services are available over the network and can be accessed through various client platforms, promoting heterogeneity.
  - Scenario: This feature is especially useful for remote working scenarios, where employees need to access business services and applications from different devices and locations.

- **Resource pooling:**
  - Explanation: The provider's computing resources are pooled to serve multiple consumers, using a multi-tenant model. Resources can be dynamically assigned and reassigned according to consumer demand.
  - Scenario: Resource pooling is beneficial in scenarios like web hosting, where multiple websites reside on the same server and resources are allocated as needed.
- **Rapid elasticity:**
  - Explanation: Cloud services can be elastically provisioned and released, sometimes automatically, to scale rapidly upon demand.
  - Scenario: This is helpful during times of fluctuating workloads, like for e-commerce sites during high traffic events (Black Friday, Cyber Monday).
- **Measured service:**
  - Explanation: Cloud systems automatically control, monitor, and optimise resource use, leveraging a metering capability at some level of abstraction appropriate to the type of service. Resource usage is monitored, controlled, and reported, providing transparency for both provider and consumer.
  - Scenario: Measured service is vital for budgeting and cost control. Businesses can track and analyse resource usage to identify areas of inefficiency or overuse.

Computing terms

- **Supercomputer:** any *single computer system* that has exceptional processing power for its time
- **Clustered Computing:** when *two or more computers* serve a single resource. This improves performance and provides redundancy
- **HPC:** High Performance Computing is any computer system whose *architecture* allows for above average performance
- **Parallel Computing:** refers to the submission of jobs or processes over *multiple processors* and by splitting up the data or tasks between them
- **Research Computing:** the software applications used by a *research community* to aid research

Compute Scaling Challenges

Faster Processors: High cost, low performance cost - Moore's Law: transistor count on a chip doubles every 18-24 months while price halves.

More Processors: Cost is acceptable

- Brings more challenges: design, testing, deployment, management, maintenance.
- This leads to the use of parallel systems and distributed systems

Calculate Speed up Terminology

π: Time cost for **non-parallelisable** (or **sequential**) tasks

σ: Time cost for tasks that **can** be parallelized, computed on a single processor

α: Proportion of tasks that **cannot** be parallelized

T(1) = σ + π: Time taken for **serial** (or **sequential)** computation.

T(N) = σ/N + π: Time taken for N parallel computations.

S(N) = T(1) / T(N): The **speed up proportion**, given by the ratio of the time for N parallel computations to the time for serial computation.

Amdahl's Law

The maximum speedup achievable with parallel processing is always limited by the sequential portion of a task.i.e. S almost = 1/α

Challenges and Problems

- As the number of processors increases, the proportion of the code that can be executed in parallel needs to be extremely high to achieve significant speedup.
- This law indicates a saturation point where adding more processors doesn't significantly reduce the computation time.
- Overhead from communication between processes are ignored in the ideal Amdahl's law scenario.

Gustafson-Barsis's Law

The speedup achieved through parallel processing can increase linearly with the number of processors, given a large enough problem size. S = α + N(1 - α) = N - α(N - 1)

Challenges and Problems

- The law assumes that problem size can be scaled indefinitely, which might not be practical in real-world scenarios.
- It also assumes that the computation resources (processors) can be increased without limit, which may not be feasible due to cost or other resource constraints.
- Overhead from communication between processes are also ignored.

Example

A program takes 3724 seconds to run on single processors. The total time spent in the sequential part is 12 seconds. What is the theoretical scaled speedup if it ran across 32 processors?

T(1) = 3724 = 12 + π, π = 3712

T(32) = σ + π / N = 12 + 3712 / 32 = 128

S(32) = T(32) / T(1) = 3724 / 128 = 29.1

Parallelism & Distributed systems models

Computer Architectures - Flynn's Taxonomy

- Single Instruction, Single Data stream (SISD): A sequential computing model where a single processor executes a single instruction on a single data stream.
- Multiple Instruction, Single Data stream (MISD): A parallel computing architecture where multiple processors execute different instructions on the same data stream, often for fault tolerance.
- Single Instruction, Multiple Data streams (SIMD): A parallel computing model where a single instruction is executed on multiple data streams simultaneously, often used in data level parallelism for tasks like image processing.
- Multiple Instruction, Multiple Data streams (MIMD): A parallel computing model where multiple processors execute different instructions on different data streams independently, typically found in modern high-performance computing systems.

Approaches for Parallelism

- **Explicit and Implicit parallelism**
  - Explicit parallelism**:** Requires programmer involvement in task management and control.
  - Implicit parallelism**:** Relies on parallel languages and compilers for task management.
- **Hardware:**
  - Aspects such as cache, multi-CPU, multi-core, symmetric multiprocessing (SMP), and non-uniform memory access (NUMA) play crucial roles in improving processing speed and parallel execution.
- **Operating System**
  - the OS manages processes and threads, essential units of execution, to enable data parallelism.
- **Software/Application:**
  - Some languages and applications, including MPI, inherently support parallel/concurrent execution.
- **Data Parallelism Approaches**
  - Big Data and Distributed Data: Approaches to handle vast amounts of data and distribute it across multiple nodes.
  - Distributed File Systems: Tools such as Hadoop, Lustre, and Ceph for managing and processing distributed data.

Distributed systems models

Master Worker/Slave Model

- Involves a master node that decomposes tasks, distributes them to worker nodes, and then gathers results.
- Ideal for tasks that can be split into independent subtasks, such as parallel computations, distributed rendering, and data processing.
- Not ideal when tasks cannot be split or when high latency between master and worker nodes affects task distribution and collection.

Single-Program Multiple-Data (SPMD)

- Each processor runs the same code on different segments of data.
- Commonly used in bioinformatics, map-reduce algorithms, and data analytics, where data can be evenly distributed among processors.
- Less suitable when data can't be evenly split, or when processors require different instructions.

Data Pipelining

- Used for applications that require multiple execution stages, typically operating on a large number of data sets.
- Ideal for tasks that have sequential dependencies, such as multimedia processing and machine learning model training.
- Not efficient when stages can't be processed in parallel or when data dependencies exist between stages.

Divide and Conquer

- A problem is divided into two or more sub-problems, and each is solved independently. The results are then combined.
- Perfect for complex tasks that can be broken down into simpler, independent tasks such as sorting algorithms, machine learning algorithms, and computational problems.
- Less efficient when the problem cannot be split into smaller tasks or when sub-problem results can't be combined easily.

Prallelization terms

1) Data parallelization
   Splitting an amount of data into partitions and distributing them among multiple processors or cores to be processed simultaneously. Same computation will be applied to each portion of data and the results are combined to generate the final output.
2) Compute parallelization
   Dividing a complex computation into smaller or simpler parallelizable parts, which can be executed simultaneously on different processors or cores. Each processor handles a different portion of the computation.

Message Passing Interface (MPI)

MPI is a library standard used for message passing in parallel systems. The interface provides several functions that enable parallel computation. These functions include:
MPI_Init: This function is used to initiate MPI computation.
MPI_Finalize: This function is used to terminate MPI computation.
MPI_COMM_SIZE: This function is used to determine the number of processors.
MPI_COMM_RANK: This function is used to determine a process's identifier.
MPI_SEND: This function is used to send a message.
MPI_RECV: This function is used to receive a message.
With the MPI system, the degree of parallelism in file processing can be increased. This is achieved in a series of steps, including broadcasting/scattering. Here are three possible solutions:
Solution1: The root process reads the file and allocates json_str/json_obj to each child process. The child processes process the data in parallel, and the root process merges the final r Message Passing Interface (MPI)
MPI is a library standard used for message passing in parallel systems. The interface provides several functions that enable parallel computation. These functions include:
MPI_Init: This function is used to initiate MPI computation.
MPI_Finalize: This function is used to terminate MPI computation.
MPI_COMM_SIZE: This function is used to determine the number of processors.
MPI_COMM_RANK: This function is used to determine a process's identifier.
MPI_SEND: This function is used to send a message.

MPI_RECV: This function is used to receive a message.
With the MPI system, the degree of parallelism in file processing can be increased. This is achieved in a series of steps, including broadcasting/scattering. Here are three possible solutions:
Solution1: The root process reads the file and allocates json_str/json_obj to each child process. The child processes process the data in parallel, and the root process merges the final results.
Solution2: Each process synchronously reads the entire file, processes corresponding data (for example, data is allocated according to rank), divides the data equally for parallel processing, and the root process merges the final results.
Solution3: Each process synchronously reads the file size, calculates file segments via rank, specifies to read a part of the file, divides the data equally for parallel processing, and the root process merges the final results. esults.
Solution2: Each process synchronously reads the entire file, processes corresponding data (for example, data is allocated according to rank), divides the data equally for parallel processing, and the root process merges the final results.
Solution3: Each process synchronously reads the file size, calculates file segments via rank, specifies to read a part of the file, divides the data equally for parallel processing, and the root process merges the final results.

Spartan

Spartan Commands

Description	Command	example response
Submit a Job	sbatch script.slurm	> Submitted batch job 18563731
Cancel a Job:	scancel 18563731	
Check Status:	squeue -j 18563731	see below

```
JOBID PARTITION     NAME      USER ST    TIME        NODES NODELIST(REASON)
18563731 physical   test      lev R      0:02        1 spartan-bm113
```

Review Output:	cat slurm-18563731.out	>…
Monitor a Job:	my-job-stats –j 18563731 -a	

Copy file to/from Spartan: use scp

Slurm Shell Script

```
#!/bin/bash
#SBATCH --nodes=1 #SBATCH --ntasks=1 #SBATCH --time=0-12:00:00
module load mpi4py/3.0.2-timed-pingpong module load python/3.7.4
time mpiexec -n 1 python3 analyze_MPI.py my-job-stats -a -n -s
————————————————————
#!/bin/bash
#SBATCH --nodes=2 #SBATCH --ntasks=8 #SBATCH --time=0-12:00:00
module load mpi4py/3.0.2-timed-pingpong module load python/3.7.4
time mpiexec -n 8 python3 analyze_MPI.py my-job-stats -a -n -s
```

Note: Remember to load the required modules before using them in python script.

- **Wall Clock Limit**: the maximum duration that is set for the job execution in high-performance computing (HPC). It is denoted by --time in the script above.
- **Nodes/Computing Resources**: the individual servers in the HPC system that carry out computations. The accuracy of this estimate is critical for ensuring efficient use of resources and completion of tasks.

Optimisation and Efficiency

- **Job Scheduler**: On an HPC system like SPARTAN, the job scheduler manages job submission, execution, and completion. Commands to manage parallel jobs running on multiple nodes include:
  - Single job: *#SBATCH --nodes=1*
  - Multiple nodes: *#SBATCH--nodes=4*
- **Optimising Throughput**: Users can optimise their throughput by:
  - Setting an accurate wall clock limit.
  - Choosing the appropriate number of computing resources.
  - Loading the correct modules.
  - Benchmarking small data before scaling up to larger values.

Choose suitable wall-time

- **Estimate Based on Previous Runs**: If you have run similar jobs in the past, use those as a starting point for estimating wall time. You can also run smaller scale versions of your job to get a rough estimate of the wall time for the full job.
- **Consider Input Size**: As you mentioned, the size of the file Haystack will affect the time it takes to execute the job. If the file is large, the wall time will need to be longer.
- **Consider Your Algorithm**: The efficiency of your algorithm will significantly impact the wall time. More efficient algorithms will complete faster and thus require less wall time.
- **Account for Communication Overhead**: When running jobs that utilize multiple nodes, there will be some overhead due to inter-node communication. This should be taken into account when setting your wall time.
- **Understand Your Hardware**: The performance of the hardware you're using will affect the execution time. Understanding the hardware's performance characteristics will help you set an appropriate wall time.
- **Leave Some Buffer Time**: To account for variability in the execution time, it can be a good idea to add some buffer time to your wall time estimate.

The challenges while choosing the wall time could include

- **Predicting Execution Time**: It can be challenging to predict exactly how long a job will take to execute, particularly if you haven't run similar jobs before.
- **Variable Execution Time**: The execution time can vary depending on many factors, including the size and nature of the input data, the state of the system, and others. This can make it hard to set a precise wall time.
- **Balancing Resources**: If you set the wall time too high, you could end up wasting resources if your job completes well before the wall time limit. If you set it too low, your job may not complete.
- **Lack of Comprehensive Understanding**: Without a comprehensive understanding of the HPC platform, the scheduling strategy, and how your particular job interacts with these factors, it's challenging to set an optimal wall time.

Public & Private & Hybrid Cloud

NIST Definition of Cloud Computing

A model for on-demand network access to a shared pool of configurable computing resources.
Resources include networks, servers, storage, applications, and services.
Rapidly provisioned and released with minimal management effort.
Limited interaction with service provider required.

Public Cloud

Provided by public cloud service providers, services are available on-demand over the public internet.
e.g. Amazon, IBM, Google, VMware
Pros:

- Variety of Services: Wide range of services available.
- Support from Service Provider.
- Cost-effective: No need to consider maintenance costs.
- Scalability: Easy to scale up or down as needed.
- Regular Upgrades: Frequent updates and improvements to the services.

Cons:

- Security: Lower security compared to private clouds.
- Loss of Control: Less control over the environment.
- Vendor Lock-in: Difficulty in transferring data/services to another provider.
- Dependence: Reliant on the continued existence of the cloud provider.

Private Cloud

Also known as an internal or corporate cloud. Dedicated to a single organization or selected users.
Pros:

- Control: Full control over the cloud environment.
- Consolidation of resources.
- Easier to secure: Access is restricted.
- Trust: Higher level of trust as the environment is private.

Cons:

- Staff/Management Overheads: Requires technical staff for maintenance.
- Hardware Obsolescence: Risk of technology becoming outdated.
- Utilization Challenges: Risk of over or underutilizing resources.
- Access Limitation: Limited to certain users.
- High Maintenance Cost: More expensive than public clouds.

Hybrid Cloud

A mix of computing, storage, and services in different environments—public clouds, private clouds, including on-premises data centers.

Pros:

- Flexibility: High level of flexibility and customization.
- Cost-effective: Balance between cost and performance.
- Secure: High level of security.
- Risk Management: Ability to manage and mitigate risks effectively.

Cons:
- Implementation: Difficult to implement and maintain.
- Higher Cost: More expensive than public clouds.
- Compatibility and Data Integration Issues: Challenges with integrating data and services across different environments.

SaaS, PaaS, IaaS, FaaS

Software as a Service (SaaS)
Definition: A cloud-based software delivery model where the provider maintains the software and provides automatic updates. The software is available via the internet on a pay-as-you-go basis. Examples include Dropbox, Microsoft365, and Jira.
Pros:
- Easy setup and use
- Maintenance handled by provider
- Time and cost-effective
- User-friendly and requires little technical background

Cons:
- Limited control over infrastructure
- Difficult to transfer to another SaaS
- Lower interoperability
- Less secure on public clouds
- Limited customization options

Platform as a Service (PaaS)
Definition: A cloud-based environment for development and deployment, equipped with resources to deliver a range of applications, from simple to sophisticated enterprise applications. An example is the Google App Engine.
Pros:
- Rapid, low-cost development and deployment
- Control over developed applications
- Access to diverse tools and integrated web services/databases

Cons:
- Adherence to platform guidelines required
- Applications typically bound to platform
- Difficulties with data migration

Infrastructure as a Service (IaaS)
Definition: A business model that provides IT infrastructure (e.g., computing, storage, network resources) on a pay-as-you-go basis over the internet. It allows configuration of required resources for IT systems and applications. An example is IBM Cloud.
Pros:
- Costs proportional to consumption
- Complete control over the infrastructure
- Highly flexible and scalable

Cons:
- Customer responsibility for security and backup
- Security and privacy risks
- High demand for skilled IT personnel

Function as a Service (FaaS)
- Also known as Serverless Computing.
- Allows functions to be added, removed, updated, executed, and auto-scaled.
- Mainly focuses on function-level computation tasks, best suited for short-term or stateless computation.
- Provides higher elasticity and reliability, eases maintenance and testing, reduces resource waste and cost.
- Examples **:** Amazon AWS Lambda, Google Cloud Functions

Pros
- Simplified deployment: The service provider handles the infrastructure, allowing developers to focus on the code.
- Cost-effective: Billing is based on the actual execution time of the functions.
- Loosely coupled architecture: Functions operate independently, reducing application complexity and easing maintenance and updates.

Cons
- Cold start problem: There may be latency during the first function call as the execution environment is provisioned.
- More suitable for short-lived, stateless tasks: Handling long-lived connections or maintaining state between function calls can be complex.
- Vendor lock-in risk: Moving to a different FaaS provider may require substantial changes in the application code.

Requirements for functions in FaaS:
- Stateless: The function should not store any internal state information. The same input should always produce the same output.
- Ephemeral: The function exists only for the time it executes. It starts when a function call is made and ends after execution.
- Side-effect free: Ideally, the function should not change the state of the system.

Suitable Applications for Each aaS

Cloud Services Ranked Based on Flexibility

IaaS > PaaS > FaaS > SaaS

IaaS: e.g. IBM Cloud
- Ideal for projects that require a high degree of control and flexibility over the infrastructure.
- Suitable for businesses looking to set up complex IT systems and applications, and who possess or can afford skilled IT personnel.

PaaS: e.g. Google App Engine
- Best used for rapid development and deployment of a range of applications, from simple to sophisticated enterprise applications.
- Requires adherence to platform guidelines and can pose difficulties with data migration.

FaaS: e.g. Amazon AWS Lambda, Google Cloud Functions

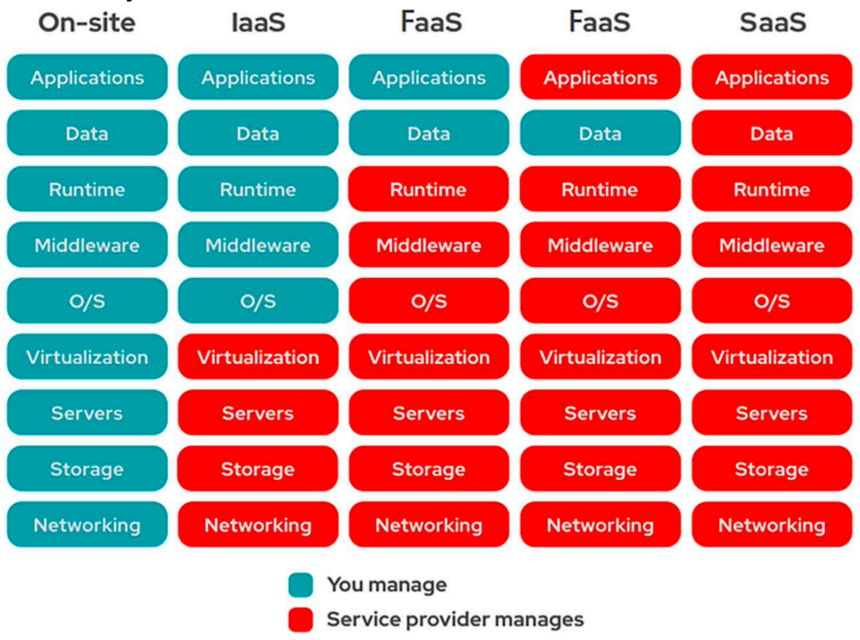
- Perfect for event-driven, independent functions such as real-time file processing, data transformation tasks, and microservice architectures.
- Best used for short-term or stateless computation tasks, as well as applications with unpredictable or highly variable workloads requiring rapid scalability.

SaaS: e.g. Dropbox, Microsoft365, Jira

- Ideal for businesses seeking easy setup, use, and automatic updates without needing to manage the underlying infrastructure.
- Suitable for users who prefer user-friendly software requiring minimal technical background, and who do not require extensive customization options.

IaaS & FaaS & SaaS Summary

In summary, there is a trade-off between cost and flexibility/scalability. The more flexible and scalable the service, the higher the cost. Conversely, lower-cost options typically offer less flexibility and scalability.



On-site	IaaS	FaaS	FaaS	SaaS
🟦 Applications	🟦 Applications	🟦 Applications	🟥 Applications	🟥 Applications
🟦 Data	🟦 Data	🟦 Data	🟦 Data	🟥 Data
🟦 Runtime	🟦 Runtime	🟥 Runtime	🟥 Runtime	🟥 Runtime
🟦 Middleware	🟦 Middleware	🟥 Middleware	🟥 Middleware	🟥 Middleware
🟦 O/S	🟦 O/S	🟥 O/S	🟥 O/S	🟥 O/S
🟦 Virtualization	🟥 Virtualization	🟥 Virtualization	🟥 Virtualization	🟥 Virtualization
🟦 Servers	🟥 Servers	🟥 Servers	🟥 Servers	🟥 Servers
🟦 Storage	🟥 Storage	🟥 Storage	🟥 Storage	🟥 Storage
🟦 Networking	🟥 Networking	🟥 Networking	🟥 Networking	🟥 Networking

🟦 You manage
🟥 Service provider manages

Architecture Styles - SoA vs ReST

Service-oriented Architecture (SoA)

It's an architectural style where software components provide services to other components via a **communication protocol**, typically over a network.

SoA emphasizes **loose coupling** between services, enabling them to be developed, deployed, and scaled **independently**.

It is defined by principles such as standard service contract, loose coupling, abstraction, reusability, autonomy, statelessness, discoverability, and composability.

Benefits of SoA:

- **Modularity:** Enables breaking down complex applications into smaller, manageable services.
- **Scalability**: Easier to scale services independently as per the demand.
- **Interoperability**: Promotes integration of various services via standardized interfaces.
- **Reuse and Composability**: Services can be reused across different applications or combined to form complex applications.
- **Flexibility**: Changes can be made to one service without affecting others.
- **Abstraction**: Services hide their internal logic, providing a defined interface to the users.
- **Cost-effectiveness**: Efficient resource use due to the stateless nature of services and the ability to scale as per demand.

SOAP (Simple Object Access Protocol) is a powerful, complex protocol used for data exchange, primarily within web services.

It operates through XML-based messages, allowing structured and standardized interaction over web services.

ReST

- Representational State Transfer (ReST) is a **resource-oriented** architecture style used to build web services, making use of **HTTP protocols**.
- It is lightweight, flexible, and utilizes URI (Uniform Resource Identifier) for each resource, with operations performed through HTTP methods.
- Each interaction between the client and server is **stateless**, meaning that the server does not need to remember previous requests, which helps scalability.

ReST Design Principles

- Create unchanging, short URIs for reliability and usability. These URIs should be discovered by following hyperlinks, not constructed by the client.
- Use nouns, not verbs in URLs to improve readability and maintainability.
- HTTP GET requests should not modify or delete any resources, conforming to ReSTful design principles of statelessness and idempotency.
- Use links in responses to connect current responses with other related data for flexibility and scalability.
- Avoid complex and lengthy URLs by minimizing the use of query strings and parameters.
- Convey operation request results using HTTP status codes for a standard way of sharing request result information.

HTTP Methods

- Safe methods: HTTP methods like GET, OPTION, and HEAD that do not change the server state. These are read-only and repeating a call has the same effect as not making the call at all.
- Idempotent methods: PUT and DELETE, whose effect remains the same regardless of how many times the request is made.
- Neither safe nor idempotent: POST. This HTTP method can change the server state and the effect may be different each time the request is made.

When to use ReST vs. SOAP

- ReST is simpler and more flexible, using URIs and HTTP methods for operations, while SOAP uses service interfaces and operations. Use ReST when you need a lightweight, stateless, and cacheable communication that is quick to set up. It is suitable for public APIs over the internet.
- SOAP is service-oriented and works well for complex, powerful applications while ReST is resource-oriented, suitable for simple data exchange and resource management. Use SOAP for more complex, enterprise-level applications where security, ACID-compliant transactions, and other advanced features are needed. It is commonly used for internal network applications due to its higher complexity and overhead.

Melbourne Research Cloud (MRC) Operations

MRC is created based on OpenStack, which is a open-sourced platform to create and manage cloud environment. It only implemented a subset of functions in OpenStack. It provides high performance computing and data storage services.
Amazon AWS (Amazon Web Services): a commercial cloud computing platform, provide flexible, usable, secure and cost-reasonable cloud computing solution for companies.

Instances
Instances are nodes, where applications can be deployed.

Volumes
A volume is similar to a hard disk for storage.
Volume Management:
Attach: Used to connect physical or virtual devices to a computer system.
Mount: Used to attach the filesystem to a specific location in the filesystem hierarchy, making the contents of the filesystem available.
Steps:

• Check the device name:	sudo fdisk -l
• Create the mounting point:	sudo mkdir /data
• Format the volume:	sudo mkfs.ext4 /dev/vdb
• Mount the volume:	sudo mount /dev/vdb /data
• Check the result:	df –h
• Persist the mounting:	sudo vi /etc/fstab /dev/vdb /data ext4 defaults 0 1

Images
An image can be created by taking a snapshot of a system.
It contains variables, states, and configurations.
Can be used to create an instance or a volume.

Access & Security
Security Groups
Manage communication (ports) between instance and outer networks like the Unimelb network or Internet.
For example, ports 22 (ssh), 5984 (CouchDB), 80 (HTTP), and 443 (HTTPS) can be opened to the world.
Key Pairs
Used for authentication when logging into an MRC instance.

Object Store
Allows data to be stored as an object.

Snapshots
A snapshot is essentially a copy of the disk state of an instance or a volume, preserving the state *at a specific time*. The instance state includes software installation, variables, configuration, and profiles.
Snapshots provide various benefits:

- They simplify configuration by installing all required packages on one instance, then using a snapshot image to configure other instances with the same configuration easily.
- They facilitate testing, allowing snapshots to be taken before package removal or system changes, which can then be reverted if needed.
- They provide stability, version control, and *repeatable* backup.

Common reasons for taking a snapshot include:

- Backing up the instance and allowing restoring from the snapshot.
- Copying the setup of an instance to create a template image for reuse.
- Re-launching an instance as a new one with a different flavour.
- Sharing the setup of an instance with other users.
- Creating a volume from a base image to launch an instance.
- Testing the system functionality before deployment.

APIs of OpenStack will be called when create instance
Nova: Create the instance,
Cinder: Authentication

Ansible

Why Ansible?

- Manage Large Systems: Ansible helps manage a vast array of systems (frontend, backend, data, etc.), thus simplifying complex deployments.
- Mitigate Errors: Manual configuration can lead to errors and incomplete setups. Ansible reduces these mistakes through automation.
- Simplify Operations: Ansible reduces tedious manual work, making it easier to manage tasks and systems.
- Track Changes: Unlike snapshot mechanisms, Ansible records the system's change history, which is crucial for problem tracking and debugging.

Benefits of Ansible

- *Ease of Use*: Ansible uses YAML for playbooks and Jinja2 for templates, both are easy to learn and use. It's also easy to install and set up.
- *Idempotent*: Ansible avoids duplicate execution side effects, ensuring that repeating a task doesn't cause different results.
- *Extensibility*: Ansible is highly extensible, allowing customization and the ability to write your own modules.
- *Non-disruptive Deployment*: Ansible supports rolling updates for continuous deployment without downtime.
- *Security*: Ansible supports data encryption with Ansible Vault.
- *Inventory Management*: Ansible handles dynamic inventory from external data sources and allows task execution against host patterns.
- *Community Support*: Ansible Galaxy offers community-developed roles while Ansible Tower provides enterprise-level features.

Benefit and Drawback of Scripting automation

Applications can be deployed across Clouds either through creation and deployment of virtual images (snapshots) or through scripting the installation and configuration of software application.

What are the benefits and drawbacks of the scripting?

- The strengths contains: the deployment is automatic and repeatable, which avoid human error; the tool provide an approach to simplify the management of resources; the deployment is scalable, which provide flexibility in customization; the scripting support historical track to help debugging.
- The shortcomings are the complexity to learn the skill of using scripting tool; There might be conflict between the scripting tool and the environment; The script need to be updated regularly to meet the demand.

Describe the approach that would be taken using Ansible for scripted deployment of SaaS solutions onto the cloud.

1. Prepare environment to ensure the availability of the network and the access permission.
2. Create Ansible playbook which contains variables, servers and databases used, tasks to execute application related command and other resources.
3. Deploy and execute Ansible command.
4. Verify and test.

Ansible YAML Examples

Setting variables

```
# Common vars
instance_flavor: uom.mse.2c9g
availability_zone: melbourne-qh2-uom
instance_image: 356ff1ed-5960-4ac2-96a1-0c0198e6a999
instance_key_name: ccc_assignment2
```

Setting volumes

```
# Volumes
volumes:
  - vol_name: server-1-volume-data
    vol_size: 80
  - vol_name: server-1-volume-docker
    vol_size: 20
  - vol_name: server-2-volume-data
    vol_size: 80
```

Setting security group to open ports

```
security_group_rules_private:
  - name: private_communication
    protocol: tcp
    port_range_min: 9100
    port_range_max: 9200
    remote_group: private_communication
  - name: private_communication
    protocol: tcp
    port_range_min: 5984
    port_range_max: 5984
    remote_group: private_communication
  - name: private_communication
    protocol: tcp
    port_range_min: 4369
    port_range_max: 4369
    remote_group: private_communication
```

Setup python and run mastodon_harvest

```
- name: install python with Mastodon and CouchDB
  hosts: localhost
  become: yes          #set as super user
  tasks:
    - name: install Python3
      apt:
        name: python3
        state: latest
    - name: install pip
      apt:
        name: python3-pip
        state: latest
    - name: install Mastodon and CouchDB
      pip:
        name:
          - Mastodon.py
          - couchdb
        state: latest
    - name: run Python application
      ansible.builtin.command:
        cmd: "python3 Mastodon_demo.py"
```

Virtualisation

- Virtualisation is a process whereby physical hardware resources are abstracted and divided into multiple virtual environments using a hypervisor or Virtual Machine Monitor (VMM).
- **Pros**: improve hardware utilization, flexibility, reliability, management efficiency, and security while reducing costs and energy consumption.

Virtual Machine Monitor (VMM)/Hypervisor

- The VMM or Hypervisor is a system-level software that manages virtual machines and provides access to computer hardware.
- It acts as the intermediary between the underlying hardware and the virtual machines and guest operating systems it supports, simulating a physical machine environment for each virtual machine (VM).

Responsibilities of VMM

- De-privileging: VMM uses de-privileging to downgrade instructions, preventing direct access or modifications of critical system resources by VMs. Lower privilege instructions need to be rerouted through the VMM to protect system security and stability.
- Primary/Shadow structures: VMM uses primary/shadow structure to duplicate parts of the memory. Guest OS modifies the copied data, and VMM synchronizes the changes, ensuring system security and isolation.
- Memory Traces: VMM records the memory accesses of the guest machines in memory traces, checks and redirects the accesses when needed to protect system security and stability.

Virtual Machine

- A Virtual Machine is a software emulation of a physical computing system, using physical computer hardware.
- VMs host guest operating systems and provide them with virtual hardware resources.

Guest Operating System

- A guest operating system is an operating system that runs within a virtual machine. It behaves as if it's running on a physical system.

Virtual Machine Working Principle

- Applications running on a guest OS believe they are interacting with physical hardware, such as a hard disk, but these interactions are translated by the VMM to the host's virtualized hardware resources.
- Virtual Network Devices communicate with external systems through the Host Network Device. The VMM manages this mapping from virtual to physical devices.

Instruction Classification

- Privileged Instructions: These are commands that can only be accessed by the operating system kernel or programs with specific privilege levels. They usually involve system resource management and affect system security and stability.
- Sensitive Instructions: These are commands that could lead to security vulnerabilities or risks, often related to system security, such as reading or modifying sensitive information, bypassing access controls.
- Innocuous Instructions: These are commands that do not pose a threat to system security or risk. They typically involve ordinary calculations and data operations and do not require special permissions or restrictions.

Aspects of VMMs

1. **Full Virtualization**

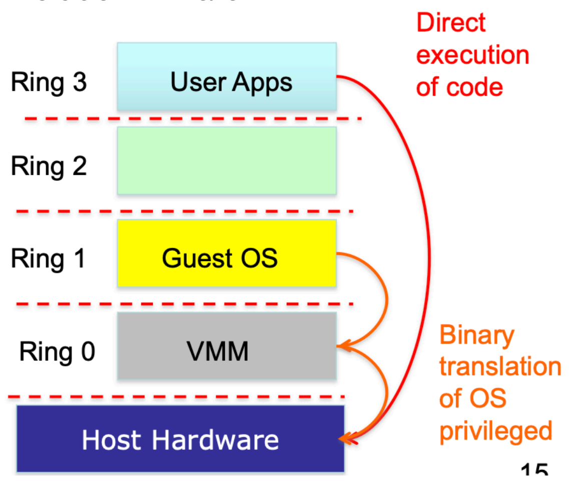
Simulates the entire computer system including hardware, allowing an unmodified guest OS to run on top of VMM as if it were running on physical hardware. Examples include VMWare.

Advantages:

- Guest OS is unaware it's in a virtual environment.
- No modification required for the guest OS.
- Can run legacy OS.
- Doesn't require hardware or OS assistance.

Disadvantages:

- Efficiency can be less due to overhead of simulating entire hardware.
- Lower performance due to simulation overhead.



2. **Para-virtualization**

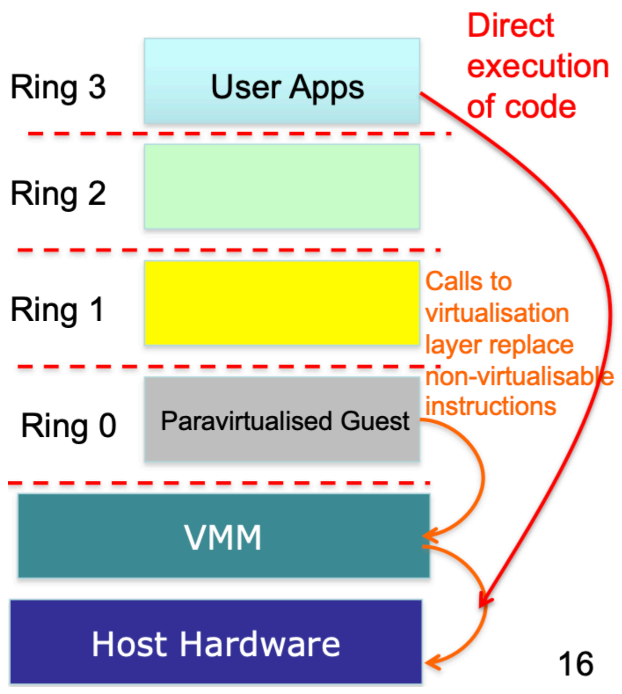
Provides a special interface to guest OS, allowing it to interact directly with the physical hardware. It requires a modified or hypervisor-aware Guest OS for better performance. Example includes Xen.

Advantages:

- Lower virtualization overheads leading to better performance.
- Lower cost in creating and maintaining virtual machines.

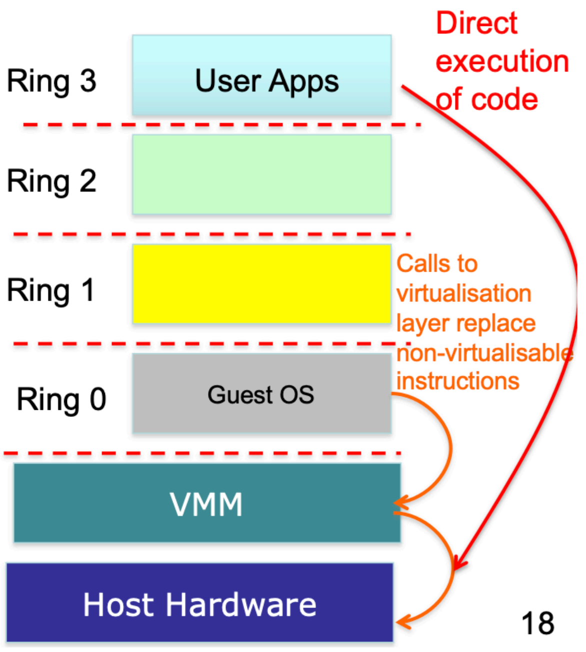
Disadvantages:

- Modification required for the guest OS.
- Can't run arbitrary OS.
- Less portable and compatible.



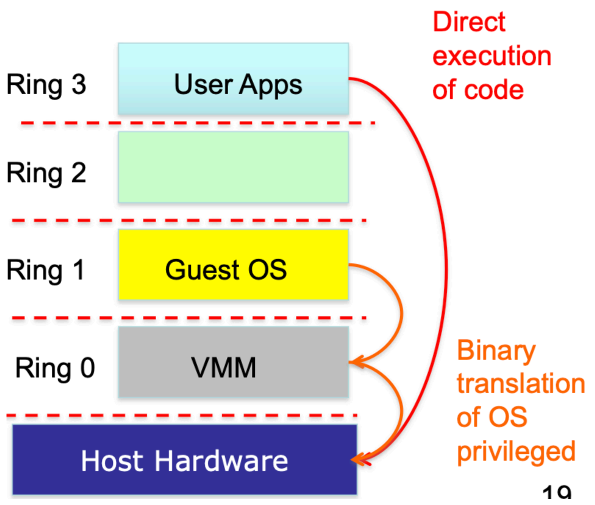
3. **Hardware-assisted Virtualization**
Enhances performance and security of virtual machines with hardware support, providing special CPU instructions for VMM to directly access physical resources. Example includes KVM.
Advantages:
- Good performance.
- Easier to implement.
- More reliable and isolated due to hardware support.

Disadvantages:
- Requires hardware support.
- Deployment configuration can be complex.



4. **Binary Translation**
Intercepts and translates the execution of virtual machines, downgrading privileged and sensitive instructions to instructions that can run on the host machine.
Advantages:
- No modification required for the guest OS.
- Supports multiple OS versions.
- Performance improvement through optimization in the translation process.

Disadvantages:
- Performance overhead due to instruction translation.
- Complex translation process.



5. **Bare Metal Hypervisor**
- VMM runs directly on actual hardware. Example includes VMWare ESX Server.
- The VMM boots up and runs on the actual physical machine, with the requirement of supporting device drivers and all hardware management.

6. **Hosted Virtualization**
- VMM runs on top of another operating system. Examples include VMWare Workstation.

Containers & Operating System Level Virtualization

- Containers are lightweight virtual machines (VMs) created through operating system-level virtualization. They are akin to advanced versions of "chroot", an operation that changes the apparent root directory for current running processes and their subprocesses.
- They are called lightweight because they share the same OS kernel with the host, requiring less resources than traditional VMs.
- Their utility lies in packaging applications and all OS dependencies into one container, ensuring portability and resource efficiency.

Advantages of Containers

- Lightweight and strong portability.
- More VMs can be hosted on the same hardware, saving resources as there is no need for a Virtual Machine Manager (VMM).
- They enable easier packaging and deployment of applications and their dependencies.

Disadvantages of Containers

- Containers can only run applications designed for the same OS.
- They cannot host a different guest OS and can only use native file systems.
- Containers share the same resources as other containers on the host system.

Memory Virtualization

Conventionally, page tables store the logical to physical page number mappings, creating an illusion of more memory than is actually available.

Container vs Virtual Machine (VM)

- *Guest OS*: VMs run on virtual hardware and have their own OS kernels, while containers share the same OS kernel.
- *Communication*: VMs communicate through Ethernet devices while containers use Inter-Process Communication (IPC) mechanisms such as pipes and sockets.
- *Security*: VM security depends on the hypervisor, while container security requires close scrutiny.
- *Performance*: VMs incur a small overhead when translating instructions from guest to host OS, while containers deliver near-native performance.
- *Isolation*: VMs do not share file systems and libraries between guest and host OS, while containers can share these elements.
- *Startup Time*: VM startup is slow, often taking minutes, while containers start up quickly, usually in a few seconds.
- *Storage*: VMs have larger storage requirements, while containers are small in size due to resource re-use.

Containerization vs Virtualization

- Both containerization and virtualization aim for resource sharing and isolation. They can co-exist, with containers often deployed on top of VMs, especially in cloud environments.
- Containerization can significantly reduce resource wastage that occurs in virtualization by sharing the host OS, drivers, binaries, and libraries among containers.
- Whether containers are better than VMs depends on various factors such as the size of the task, the lifespan of the application, and security concerns.

Docker Concepts and Terms

- **Container**: A runtime instance of a Docker image, behaving like an independent machine. It includes the necessary code, environment, system tools, and libraries to run an application.
- **Image**: A read-only blueprint used to create containers. It is a static snapshot of a container that includes the necessary files and configuration information.
- **Dockerfile**: A text file that describes how to build an image. It contains necessary instructions and configuration information such as base image, application code, dependencies, and environment variables.
  - **Layer**: Each modification to an image, represented by an instruction in the Dockerfile.
  - **Build**: The process of constructing Docker images from a Dockerfile.
- **Registry**: A server for storing images, either locally or remotely. This could be a hosted service like Docker Hub.
- **Repository**: A set of Docker images stored in a registry. Repositories contain multiple images distinguished by tags.
- **Tag**: A label applied to a Docker image in a repository, generally including version number and release date.
- **Docker Compose**: A tool for defining, running, and managing multiple Docker containers. Configuration information, dependencies, network settings, etc., are defined in a YAML file.
- **Docker SWARM**: Docker's official container management tool for managing multiple host machines.

Data Storage and Networking in Docker

- **Docker Volume**: Managed by Docker Engine, volumes allow sharing of data between containers. Data stored in a specific directory on the host file system (e.g., /var/lib/docker/volume/) isn't lost when a container is deleted or restarted.
- **Bind mounts**: Allows a directory or file from the host file system to be directly mapped into a container, facilitating sharing of files or directories between container and host without needing to create and manage volumes.
- **Host Networking** (Linux Only): All containers use the host network, sharing the same IP address. Different ports must be assigned for different containers.
- **Bridge Networking** (Default): Creates a local network bridge. By connecting to this network, containers can communicate with other containers or networks using the bridge's IP address.

Working with Docker Registries and Images

- Login to Docker Hub or other public Docker Registry: **docker login -u <username>**
- Login to a private Docker Registry (e.g., AWS ECR, Nexus Server):
  docker login -u AWS https://ecr.ap-southeast-2.amazonaws.com
- Logout from a Docker Registry: **docker logout**
- Pull an image from a Docker Registry: **docker pull <name>[:TAG]**
- List all Docker images: **docker images**
- Tag a Docker image: **docker tag <source_image> <target_image>**
  docker tag nginx alwynpan/comp90024:nginx
- Push an image to a Docker Registry: **docker push <name>[:TAG]**
  docker push alwynpan/comp90024:nginx

Creating and Running Docker Containers

- Create a container: **docker create --name** nginx **-p** 8080:80 nginx
- Start a container: **docker start <container_name>**
- Run a container (creates and starts the container in one command):
  docker run --name nginx **-p** 8080:80 **-d** nginx

Listing Docker Containers

- List running Docker containers: **docker ps**
- List all Docker containers (running and stopped): **docker ps -a**

Managing Docker Containers

- Restart a Docker container: **docker restart <container_name>**
- Stop a Docker container: **docker stop <container_name>**
- Remove a Docker container (must be stopped first): **docker rm <container_name>**
- Remove a running Docker container: **docker rm -f <container_name>**
- View logs of a Docker container: **docker log <name or id>**
- Run a shell within a Docker container: **docker exec -ti -w** /usr/share/nginx/html/ nginx sh

Managing Data in Docker

- Create a volume: **docker volume create --nam**e htdocs
- Start a container with a **volume attached**:
  docker run --name nginx-volume **-p** 8080:80 **-v** data:/usr/share/nginx/html **-d** nginx
- Start a container with **bind mount attached**:
  docker run --name nginx-bind **-p** 8081:80 **-v** $(pwd)/data:/usr/share/nginx/html **-d** nginx

Dockerfile Structure and Explanation

- **FROM nginx:latest**: This sets the base image as the latest version of Nginx.
- **ENV WELCOME_STRING "nginx in Docker"**: This sets the environment variable WELCOME_STRING with the value "nginx in Docker".
- **WORKDIR /usr/share/nginx/html**: This sets the working directory in the container to /usr/share/nginx/html.
- **COPY ["./entrypoint.sh", "/"]**: This copies the entrypoint.sh file from the current directory on your machine to the root directory in the container.
- **RUN cp index.html index_backup.html; **
  **chmod +x /entrypoint.sh; **
  apt-get update && apt-get install -qy vim:
  This executes a series of commands: creates a backup of the index.html file, grants execution rights to the entrypoint.sh script, updates the packages and installs vim.
- **ENTRYPOINT ["/entrypoint.sh"]**: This sets the entrypoint of the container to be the entrypoint.sh script, which will be executed every time the container starts.
- **CMD ["nginx", "-g", "daemon off;"]**: This sets the default command for the container when it starts.

Dockerfile Notes

- All the RUN, COPY, FROM, and ENV instructions in the Dockerfile are executed at build time, whereas ENTRYPOINT and CMD instructions are executed at container startup.
- CMD instruction can be overridden by providing command line arguments during docker run. Only the last CMD instruction in a Dockerfile will be effective.
- The ENTRYPOINT and CMD instructions should be specified in JSON array format to avoid parsing errors.

Explanation of ENTRYPOINT

- ENTRYPOINT specifies the script that will be executed when the container starts, and CMD provides the arguments to this script.
- Unless overridden, ENTRYPOINT script will always be executed when the container starts.
- The ENTRYPOINT script in this Dockerfile replaces "Welcome to nginx!" with the value of WELCOME_STRING in the index.html file and makes the ENTRYPOINT a pass-through, meaning it will execute the Docker command provided in CMD by default.

Build with docker file and run container

- **docker build -t** demo2
  This command builds a Docker image from the Dockerfile in the current directory, and tags it as "demo2".
- **docker run --name** demo2 **-e** WELCOME_STRING="COMP90024" **-p** 8081:80 **-d** demo2:
  This command runs the "demo2" Docker image as a container, names it "demo2", sets the environment variable WELCOME_STRING as "COMP90024", maps port 8081 of the host to port 80 of the container, and runs the container in detached mode.

Container Orchestration Tools

- Provide a framework for integrating and managing containers at scale.
- Include tools like Kubernetes and Docker Swarm.
- Simplify container management processes.
- Manage availability and scaling of containers.

Features of Container Orchestration Tools

- Networking: Create secure and isolated network environments.
- Scaling: Auto-scaling to adapt to varying workloads.
- Service Discovery & Load Balancing: Manage traffic and optimize resource utilization.
- Health Check & Self-healing: Detect issues and recover from failures automatically.
- Security: Protect containers and their data.
- Rolling updates: Update without downtime.
- Configuration management: Manage configuration files centrally.

Docker Compose
- A tool for defining and running multi-container Docker applications.
- Simplifies configuration, network connections, and volume mounts using YAML.
- Provides command-line tools for starting, stopping, configuring, and managing containers.
- Used primarily on single hosts.
- Start containers: **docker compose up -d**
- Stop containers: **docker compose stop**
- Remove containers: **docker compose down**

Docker Swarm
- Docker orchestration tool used for creating and managing a container cluster across multiple Docker hosts.
- Features include auto-scaling, load balancing, and service discovery.
- Configured and managed through Docker command line or Docker API.
- Structure:
  - Raft consensus group: Includes internal distributed state store and all manager nodes.
  - Manager Node: Conducts orchestration and management tasks.
  - Worker Node: Receives and executes tasks from the manager node.
  - Service: Consists of one or more replica tasks.
  - Task: The combination of a single Docker container and its run commands.

Docker Swarm Commands
- **Initiate Swarm**: sudo docker swarm init --advertise-addr <IP>
- **Join Swarm**: sudo docker swarm join --token <token> <IP>:2377
- **Create a service**: sudo docker service create --replicas 3 -p 8080:80 --name nginx nginx:alpine
- **List services**: sudo docker node ls
- **Check a service**: sudo docker service ps nginx
- **Scale up/down a service**:
  - docker service scale SERVICE=REPLICAS
  - docker service scale SERVICE=REPLICAS sudo docker service scale nginx=6
  - sudo docker service scale nginx=1
- **Update a service**: sudo docker service update --image alwynpan/comp90024:demo1 nginx

Big Data

Challenges of Big Data

- **Volume**: Managing and analyzing large quantities of data
- **Velocity**: Need for faster processing due to increasing data generation
- **Variety**: Managing diverse and complex data formats
- **Veracity**: Ensuring data authenticity and reliability
- Distributed processing and fault tolerance in system design
- Limitations of traditional DBMS for big data, necessitating the use of NoSQL databases

Big Data Analytics

- Big Data Analytics is a broad concept that includes distributed computing, machine learning, data mining, data aggregation, clustering, sentiment analysis, recommendation systems, etc.
- It involves the accumulation and analysis of vast amounts of data, which has led to the development of large-scale business intelligence and advanced machine learning algorithms.
- Apache Hadoop is a commonly used framework for distributed computing in Big Data Analytics.

Challenges of Big Data Analytics

A Big Data Analytics framework needs to distribute both data and processing over multiple nodes/servers. This implies:

- Reading and writing distributed datasets
- Preserving data in the event of node failures
- Executing MapReduce tasks
- Being fault-tolerant (a few failing compute nodes may slow down processing, but not stop it)
- Coordinating task execution across a cluster

CAP Theorem

A distributed data store can't provide more than two out of these three guarantees:

- **Consistency**: Every read from the data store receives the most recent write or an error.
- **Availability**: Every request to the data store receives a (non-error) response, without guarantee that it contains the most recent write.
- **Partition Tolerance**:The system can sustain any amount of network failure that doesn't result in a failure of the entire network.

Consistency and Availability: Two Phase Commit

- The Two Phase Commit protocol is typically used in relational DBMS to enforce consistency.
- It locks data within the transaction scope and performs transactions on write-ahead logs.
- Transactions only complete when all nodes in the cluster have executed the transaction.
- If a partition is detected, transactions are aborted.
- Best in co-located cluster systems or scenarios where strong consistency is prioritized over availability, such as in banking transactions where data consistency is of utmost importance.

Consistency and Partition-Tolerance: Paxos

- Paxos is a consensus-based family of algorithms that is partition-tolerant and consistent.
- Nodes in Paxos either propose a value or accept/refuse a proposal. A consensus is reached when a quorum of acceptances is attained, and an agreed value is sent to the acceptors.
- Paxos can recover from partitions while maintaining consistency, but the smaller part of a partition (the part not in the quorum) will not send responses to clients, reducing availability.
- Best for environments that require fault tolerance, like distributed databases or cloud computing infrastructures.

Availability and Partition-tolerance: Multi-Version Concurrency Control (MVCC)

- MVCC is a method that ensures availability and allows for some level of recovery from a partition. It allows concurrent updates without distributed locks.
- Every node in a cluster always accepts requests. Each update is assigned a unique revision number. The transaction that completes last has the higher revision number, hence is considered the current value.
- In case of partition and concurrent requests with the same revision number, conflicts are left to be resolved by the application after the partition is solved.
- Best for systems prioritizing high availability, like large-scale web applications.

CouchDB

What and Why Document-oriented DBMS?

A Document-oriented DBMS (Database Management System) is a type of NoSQL DBMS where data is stored, retrieved, and managed in a semi-structured manner, usually as documents.

Advantages:

- Coarse-grained data: Unlike relational DBMS, document-oriented DBMS stores data in a coarse-grained format reducing complex relationships among data. This leads to improved fault-tolerance in distributed scenarios.
- Efficient data access: Due to data being stored and accessed within the same partition, data access and modifications are more efficient and reliable, reducing partition access and modification overheads.
- Scalability and flexibility: Document-oriented DBMS can easily accommodate new fields without altering the entire database, supporting data expansion and updates, thus providing greater scalability and flexibility.

Replication and Sharding

- Replication is the process of duplicating and storing the same data on multiple nodes to enhance the database's fault-tolerance.
- Sharding is the practice of partitioning data into smaller subsets or "buckets," each stored on different servers to distribute computational load and enhance performance.

Partitioning in CouchDB

- Grouping logically-related rows into the same shard.
- It enhances performance by narrowing the scope of queries to a limited set of documents within one shard.
- Partitioning is useful when the database is relatively small, ideally smaller than a shard.
- A database must be declared as 'partitioned' during its creation in CouchDB.
- Example: Suppose you are building a Twitter-like application and using CouchDB as your database. A partition in this case could represent a single user's tweets. When you want to load a user's timeline, instead of querying the entire database, you would just need to query that specific user's partition, significantly improving the query speed.

MapReduce in CouchDB

- In CouchDB, the "Map" stage transforms data into key-value pairs, where each pair represents a document or a part of it. The "Reduce" stage then merges values that share the same key, producing a new set of key-value pairs.
- **Pros:** ability to process large datasets, run in a distributed environment to enhance performance.
- Example: Suppose you're again building a Twitter-like application and you want to count the total number of tweets each user has made. The Map step could go through each tweet in the database and emit a key-value pair, where the key is the user ID and the value is 1 (representing one tweet). The Reduce step would then take all these key-value pairs and sum up the values for each user, giving you the total number of tweets for each user.

Databases in CouchDB:

- A CouchDB instance can have many databases, each containing its unique set of functions and sharded differently.
- Databases can be added, deleted, and listed using HTTP calls (PUT, DELETE, and GET respectively).
- There are system databases prefixed with an underscore (e.g., _users).

Documents in CouchDB:

- A document can be added using the POST method and retrieved with the GET method.
- Documents have system properties including _id (ID of a document, generated by CouchDB by default) and _rev (revision number, ensuring version control).
- The ID of a document can be self-defined using PUT instead of POST, though it may lead to update errors.
- Documents can be updated using PUT with the _rev returned when creating the document.
- Deletion of a document uses DELETE method, though the deleted version can still be retrieved using its id and rev.
- Documents can be permanently deleted using the _purge command.
- Multiple documents can be managed in bulk using the _bulk_docs API.
- Documents can have one or more attachments of any MIME-type.

Error 409 in Document Updates:

- Document update conflict may result in Error 409. This conflict usually arises if the correct _rev is not included during the update process.
- In a clustered database, such conflict could occur even when the revision number is provided (e.g., during network partition).

Multiversion Concurrency Control (MVCC):

- MVCC relies on increasing revision numbers and preserving old versions for availability.
- Conflicts in MVCC can be avoided by stating the revision the update refers to.
- In a CouchDB cluster, conflicting revisions may happen due to network partitioning. The "winning" revision is determined deterministically and "losing" revisions are stored for conflict resolution.

Document Deletion and Compaction:

- Documents are not truly deleted until purged. A deleted document can be retrieved unless purged.
- Accumulation of old revisions can be managed through automatic deletion and compaction.
- Compaction can be customized by setting a limit on the number of revisions stored before deletion.

Views

- Views in CouchDB are definitions of MapReduce jobs that are updated as new data come in.
- They allow CouchDB to organize data according to certain key-value pairs for efficient querying and aggregation operations like calculating sum, average, maximum, minimum etc.
- They are essentially B-tree indexes. Each view is actually an index based on the B-tree data structure.
- Keys for views can be composite, allowing for data aggregation at different levels.
- When we use it, we need to balance resource usage and performance.
- Reduce functions in views must be referentially transparent, associative, and commutative.
- They can be accessed through HTTP and are grouped into design documents.
- **Pros**: views are very suitable for querying and analyzing large amounts of data due to their pre-organized data structure.
- **Cons:** the main drawback is that they require planning and prediction of all usage scenarios and query needs during the design phase. In an unbalanced design, views may use a lot of storage.

Apache Hadoop: a distributed computing framework

Apache Hadoop Distributed File System (HDFS)

Description: A fault-tolerant file system explicitly designed for distributing data across multiple nodes in a cluster. It uses large block sizes (128MB) for improved efficiency and reduced network operations compared to conventional file systems. Data is divided into large blocks that are stored across various datanodes, with block location metadata stored in a separate namenode.

Advantages

- High reliability due to its fault-tolerant design.
- Reduced metadata and efficient network usage thanks to larger block sizes.
- Fewer seek operations on large files, leading to improved processing efficiency.
- Suitable for high-volume data, capable of alleviating the single server bottleneck problem.

Disadvantages

- Not optimized for low-latency data access, more suited for batch processing of large files.
- Certain aspects of data management, such as data consistency, might need to be managed by the program developer.

Hadoop Shell

Used to manage files on HDFS as typical operating system shells are inadequate for this purpose. It offers command replication and dedicated commands for moving files between the local file system and the cluster.

Yet Another Resource Negotiator (YARN)

It is used for task scheduling and resource management in the Hadoop ecosystem. It comprises a central Resource Manager (master node), several Node Managers (slave nodes), and starts an Application Master for each job to negotiate resources and coordinate operations on the slave nodes.

Apache Spark

Description: A powerful computing framework that operates within the Apache Hadoop ecosystem, using YARN and Zookeeper for resource management. Apache Spark is capable of handling a variety of tasks, including complex jobs like machine learning or graph-based algorithms. It interacts with HDFS for data storage, allowing it to read/write data in various formats and from various sources.

Advantages:

- Designed to reduce the latency inherent in Hadoop MapReduce jobs and support memory-intensive analytics.
- Provides a more sophisticated and powerful programming environment.
- Efficient for complex jobs that benefit from caching data in memory and requires finer-grained control over the execution.

Disadvantages:

- Can be resource-intensive when dealing with very large datasets, even with in-memory processing.
- Primarily a batch system, not optimized for real-time processing requirements.

Spark Runtime Architecture

Core components include:

- Job: Operations on a dataset.
- Task: Single operation in a job; multiple tasks can exist in one job.
- Executor: Process where tasks are executed.
- Cluster Manager: Process that assigns tasks to executors.
- Driver Program: Main application logic.
- Spark Context: Configuration info of the job.

Deployment modes:

- **Local Mode**: All components run in the same JVM, suitable for development and testing.
- **Client Mode**: Used for interactive applications; machine hosting the driver program should be connected to the cluster until job completion.
- **Cluster Mode**: All components, including driver program, are executed on the cluster; job can run autonomously. Ideal for non-interactive Spark jobs.

Resilient Distributed Dataset (RDD)

RDD is a fundamental data structure of Spark, a data storage method in Spark during computation.

Key aspects:

- Resilient: Data partitions can be rebuilt if a node fails; node failure doesn't affect data integrity.
- Distributed: Data is divided into chunks and sent to different nodes.
- Dataset: Collection of objects, generic term.

Features of RDDs:

- Immutability: Once defined, cannot be changed. Simplifies parallel computations. However, every time a transformation is performed, a new RDD must be created, having extra overhead.
- Transiency: RDDs are designed to be used once and then discarded. However, they can be cached during processing for performance improvement. Not suitable for use cases requiring persistent data.
- Lazy Evaluation: Computations on RDDs are not immediately executed. Optimizes the execution order and increases efficiency. However, this may result in unexpected performance behavior because computations are delayed until necessary.

Hadoop ecosystem

1. Data is ingested into HDFS, either directly or via Hadoop Shell commands.
2. YARN manages the resources of the Hadoop cluster and schedules tasks.
3. Spark process this data, pulling it directly from HDFS.
4. Spark uses YARN for resource management, ensuring efficient utilization of resources in the cluster.
5. Spark can use various modules to perform complex data processing tasks. The processed data can then be written back to HDFS or other databases.

Authentication

- Authentication is the process of verifying a user's identity in a system, but it does **not check what the user is permitted to do**.
- It is necessary for preventing identity forgery. Hence, a secure solution is required.
- Traditional methods like password authentication are limited by complexity and scalability issues.

Public Key Infrastructures (PKI)

- PKI is based on public key cryptography (also called **asymmetric** cryptography), which uses a pair of keys – a public key and a private key.
- The public key is openly available and used to encrypt data.
- The private key is known only to the key owner and used to decrypt data.
- The public and private keys are mathematically related but one cannot infer the private key from the public key.
- **Pros**:
  - Avoids the need for key transmission, making key management easier and safer.
  - **Only the recipient's public key is needed for encryption, and only the sender's private key needs to be securely stored**.

Public Key Cryptography

- PKC uses two distinct keys, one that can be made public (public key) and one that must be kept private (private key).
- With private keys, one can digitally sign messages and validate them with associated public keys.
- PKC simplifies key management as public keys can be used for a short time and then discarded, and only the private key needs to be kept long term and securely.

Public Key Certificates

- Public Key Certificates are used to **identify public keys and the users who own the corresponding private keys**, similar to business cards.
- They resolve the issue of trust when using asymmetric encryption by associating a public key with a trusted user.
- These certificates are issued by a trusted Certification Authority (CA).

Certification Authority

- CA is the core component of PKI, responsible for issuing and managing digital certificates.
- Duties include:
  - developing regulations for digital certificates,
  - issuing certificates,
  - revoking certificates when they expire or are misused,
  - maintaining records of issued digital certificates,
  - ensuring the security of certificates to avoid malicious attacks and data leaks.
- While not completely secure, CAs are more difficult to attack and forge due to their high costs and maintenance complexity.

PKI and Cloud

- Key-pair is extensively used in Infrastructure as a Service (IaaS) to ensure secure access when creating instances.
- With cloud service providers across different regions following various security standards and certification mechanisms, there's no universally present Certification Authority (CA). This implies that cloud computing users must use different authentication methods to prove their identity and gain access to cloud resources.
- Access to Melbourne Research Cloud (MRC) VMs and SPARTAN cluster requires proof of identity as a member of the University of Melbourne and enrollment in COMP90024 respectively.
- The necessity for single sign-on (SSO) is critical. SSO is an identity verification and access control technology that allows users to access multiple applications, services, or websites after just one login. This enhances user efficiency and experience while reducing security risks.

Authentication & Authorization

- Authentication validates the user's identity without inspecting the user's actions
- Authorization is the process of granting permissions based on the authenticated identity, controlling resource access.
- Authorization is typically realized through **Virtual Organisations (VO)**, allowing different organizations and members to share and utilize resources effectively.
- Various approaches to authorization include Group-Based Access Control, Role-Based Access Control (RBAC), Identity-Based Access Control (IBAC), and Attribute-Based Access Control (ABAC).

Other Cloud Security Challenges

- Single Sign-On (SSO): A system that improves user efficiency and experience by managing authentication and token passing.
- Auditing: Involves processes such as logging, intrusion detection, and security auditing to monitor and evaluate security.
- Deletion & Encryption: Data deletion can be time-consuming due to data location and scale. Loss or leakage of keys can render data inaccessible.
- Liability: Determining the responsibilities and obligations of cloud service providers to protect user data and provide secure services.
- Licensing: Management of licenses becomes complex due to the elasticity and scalability of cloud services.
- Workflows: Ensuring efficient workflow execution using appropriate security policies and technological measures.
- Changing Technical/Legal Landscape: Navigating the constantly evolving technology and legal regulations in the cloud domain.