VCS FOR DEVOPS

COMP3104 - DevOps

LEVERAGING VCS FOR DEVOPS

It is a software tool that teams uses to track changes are called **Version Control Systems (VCS)**.

There are some basics, though, that make a VCS what it is.

- As we learn in previous week that VCS should retain a long-term history of changes made on a project including creation, deletion, edits and more. For e.g. author, date and any notes from the changes we made.
- Moreover it should have a solution for branching and merging new code changes to the main project to allow concurrent work from multiple developer of a team.
- The main codebase for a project is called the **Trunk, Master** or **Main**.
- **Branches** are created as independent streams of work that can be merged to the Main.

REASONS WHY VCS IS CRITICAL FOR DEVOPS

1. Avoiding dependency issues in modern containerized applications

- **Microservices** have essentially become the default for the development of new applications.
- With this trend, **dependency issues** have entered centre stage as something that can make or break a project.
- Prevent dependency hell, affectionately called **JAR hell in Java**.
- The **linear path** of building new features and fixing bugs is bound to **clash** with the parallel development of another feature by another team.
- For DevOps, finding the balance between moving quickly and maintaining application reliability is crucial.
- Part of that means cultivating traceability, gaining visibility into the changes made to the code and understanding how those changes affected application performance.

Web Reference Link

REASONS WHY VCS IS CRITICAL FOR DEVOPS

2. Version control is tied to higher DevOps performance

- The “version control was consistently one of the highest predictors of performance.”
- Top performing engineering teams were able to achieve higher throughput .
- The ability to more easily view and understand how changes to one part of the code caused problem across the application.

REASONS WHY VCS IS CRITICAL FOR DEVOPS

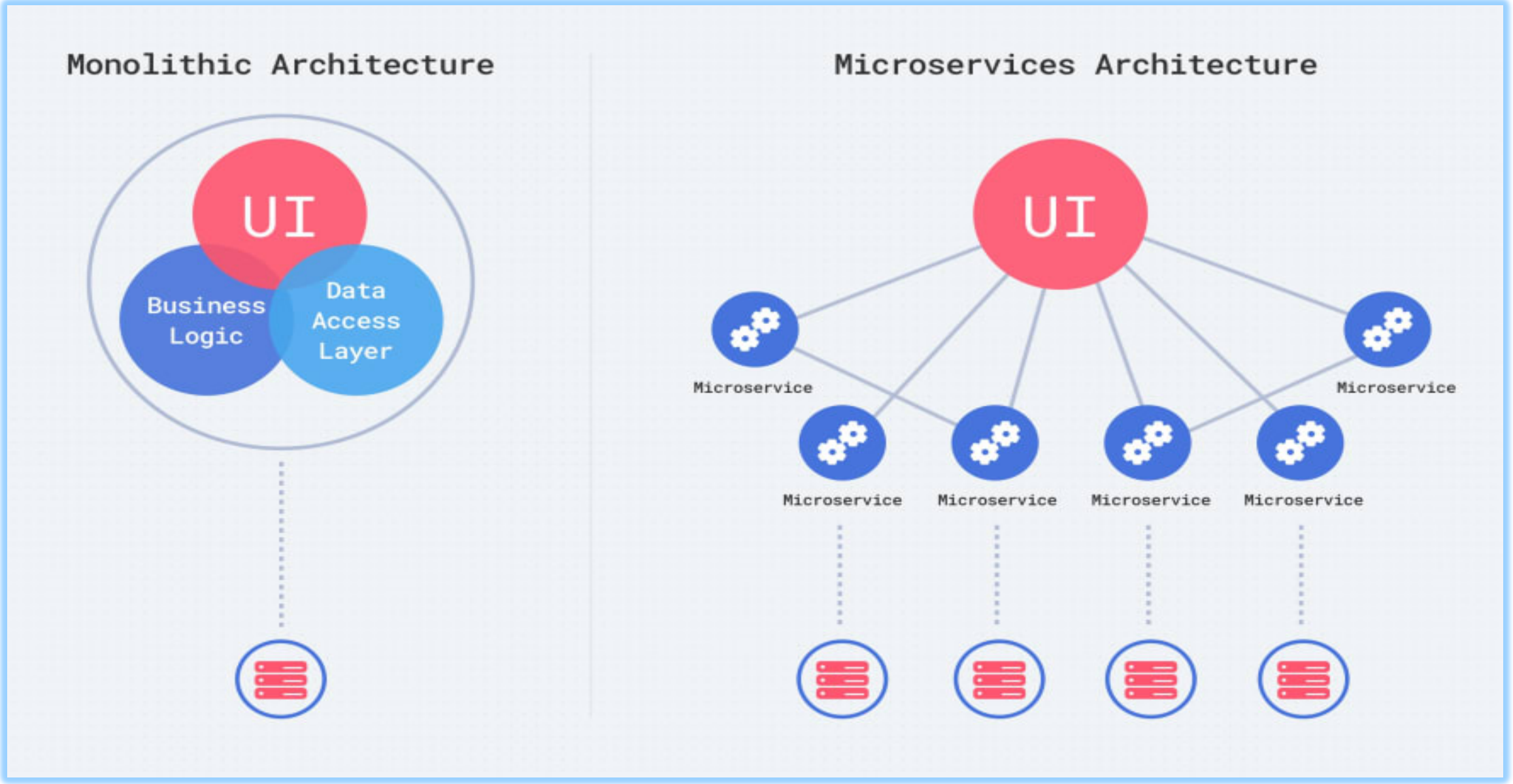
3. Supports building more reliable applications

- The study found that applications built with comprehensive version control systems were more reliable.
- Using VCS now its not hard to draw the connections between high performing DevOps teams and reliable applications.
- DevOps' role in any organization is to enable successful advancements, and because change is often the source of failures, there's a big focus on mitigating the risk of change.
- The first step is usually making smaller changes more frequently (hence the popularity of CI).
- Just as important, and even more so in an accelerated workflow, is tracking those changes so that everyone is looking at the same thing and so that troubleshooting is easier in the case of failures.

IMPORTANT FEATURES A VCS ENABLES IN DEVOPS

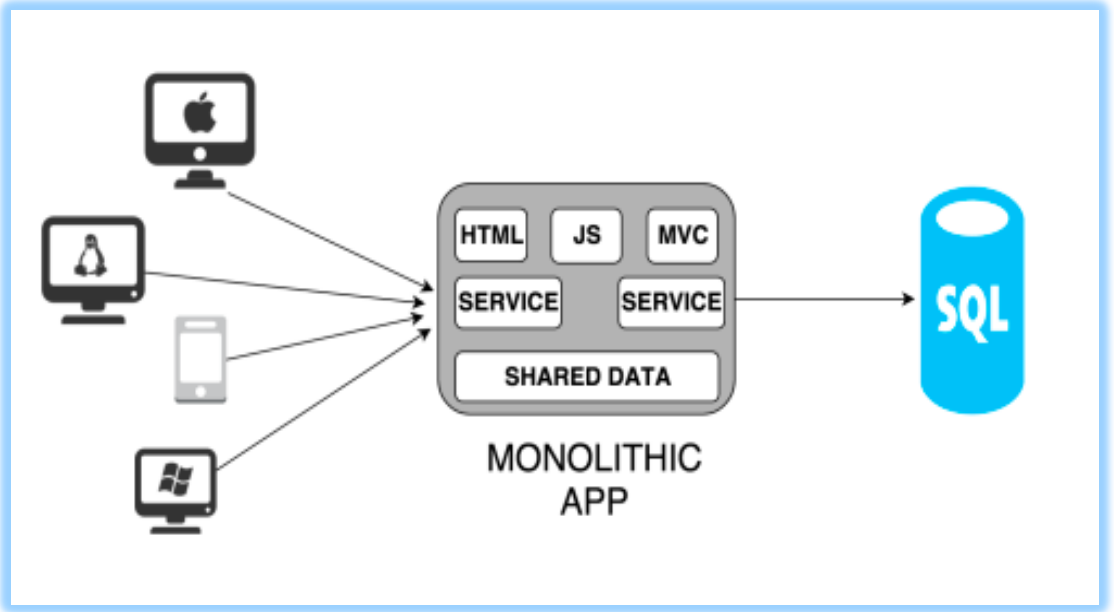
1. Improved collaboration
2. Improved CI
3. Better support for distributed teams working in a single code base
4. Streamlined workflows through the enablement of CI/CD

MONOLITH & MICROSERVICES ARCHITECTURE



MONOLITH ARCHITECTURE

- A monolithic application is built as a single, unified unit.
- Monolith consists of three parts:
  - a database,
  - a client-side user interface (consisting of HTML pages and/or JavaScript running in a browser), and a server-side application.
  - The server-side application will handle HTTP requests, execute domain-specific logic, retrieve and update data from the database, and populate the HTML views to be sent to the browser.
- In a monolith, server side application logic, front end client side logic, background jobs, etc, are all defined in the same massive code base.
- In monolithic architecture if developers want to make any changes or updates, they need to build and deploy the entire stack all at once.



MONOLITH PROS

Fewer Cross-cutting Concerns

It has large number of cross-cutting concerns, such as logging, rate limiting, and security features such audit trails and DOS protection. When everything is running through the same app, it's easy to hook up components to those cross-cutting concerns.

Less Operational Overhead

Having one large application means there's only one application you need to set up logging, monitoring, testing and less complex to deploy.

Performance

There can also be performance advantages, since shared-memory access is faster than inter-process communication (IPC).

MONOLITH CONS

Tightly Coupled

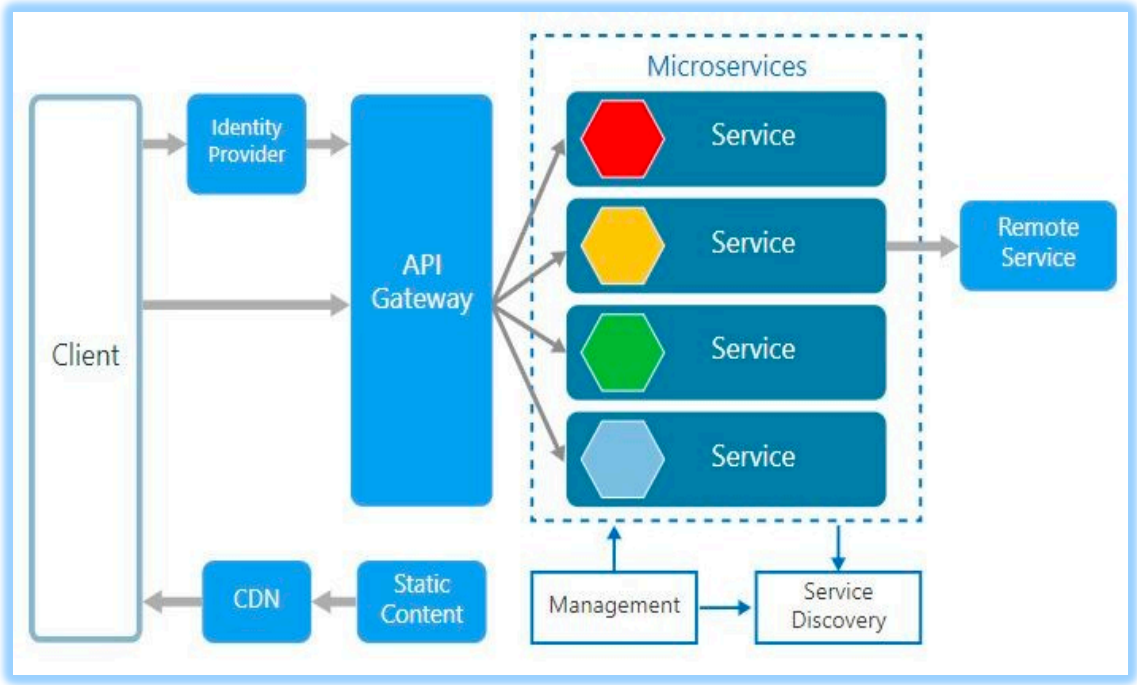
Monolithic app services tend to get tightly coupled and entangled as the application evolves, making it difficult to isolate services for purposes such as independent scaling or code maintainability.

Harder To Understand

Monolithic architectures are also much harder to understand, because there may be dependencies, side-effects, and magic which are not obvious when you're looking at a particular service or controller.

MICROSERVICES ARCHITECTURE

- There is nothing inherently “micro” about microservices.
- While they tend to be smaller than the average monolith, they do not have to be tiny.
- Application size is relative and there’s no standard of unit of measure across organizations.
- The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.
- These services are built around business capabilities and independently deployable by fully automated deployment machinery.
- Bare minimum of centralized management of these services.



Web Reference Link

MICROSERVICES PROS

Better Organization

Each microservice has a very specific job, and is not concerned with the jobs of other components.

Decoupled

Decoupled services are also easier to recompose and reconfigure to serve the purposes of different apps For e.g. serving both the web clients and public API). It allows fast and independent delivery of individual parts within a larger, integrated system.

Performance

Under the right circumstances, microservices can also have performance advantages depending on how they're organized because it's possible to isolate hot services and scale them independent of the rest of the app.

Fewer Mistakes

Microservices enable parallel development by establishing a hard-to-cross boundary between different parts of your system.

MICROSERVICES CONS

Cross-cutting Concerns Across Each Service

When building a new microservice architecture, you're likely to discover lots of cross-cutting concerns that you did not anticipate at design time.

You'll either need to incur the overhead of separate modules for each cross-cutting concern (i.e. testing), or encapsulate cross-cutting concerns in another service layer that all traffic gets routed through.

Even monolithic architectures tend to route traffic through an outer service layer for cross-cutting concerns, but with a monolithic architecture, it's possible to delay the cost of that work until the project is much more mature.

Higher Operational Overhead

Microservices are frequently deployed on their own virtual machines or containers, causing a increase of VM wrangling work.

VERTICAL SCALING

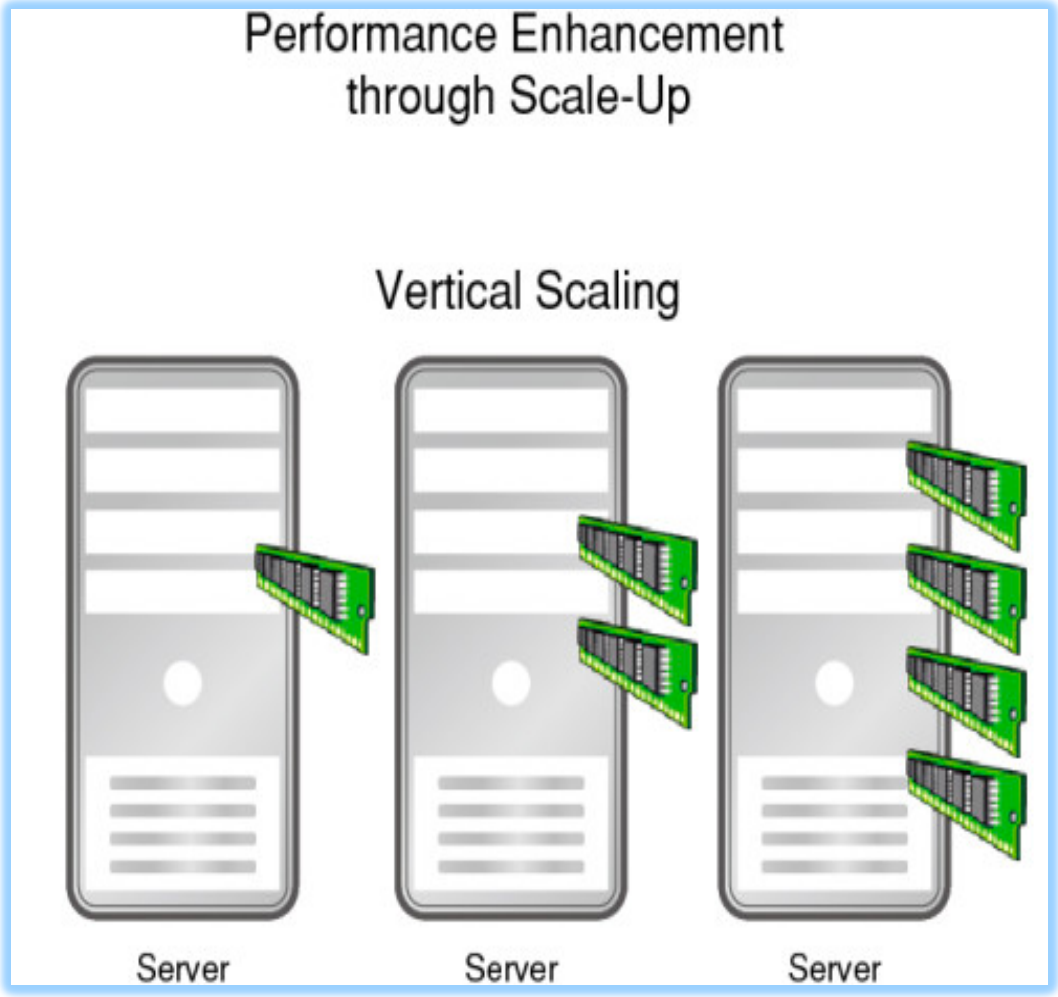
Vertical scaling refers to adding more resources (CPU/RAM/DISK) to your server (database or application server is still remains one) as on demand.

For e.g. to buy an expensive hardware and use it as a Virtual Machine hypervisor (VMWare ESX).

Vertical Scaling usually means upgrade of server hardware.

Sometime to scale vertically includes increasing IOPS (Input / Output Operations), amplifying CPU/RAM capacity, as well as disk capacity.

However, even after using virtualization, whenever an improved performance is targeted, the risk for downtimes with it is much higher.



HORIZONTAL SCALING

Horizontal Scaling is a must use technology – whenever a high availability of (server) services are required

Scaling horizontally involves adding more processing units or physical machines to your server or database.

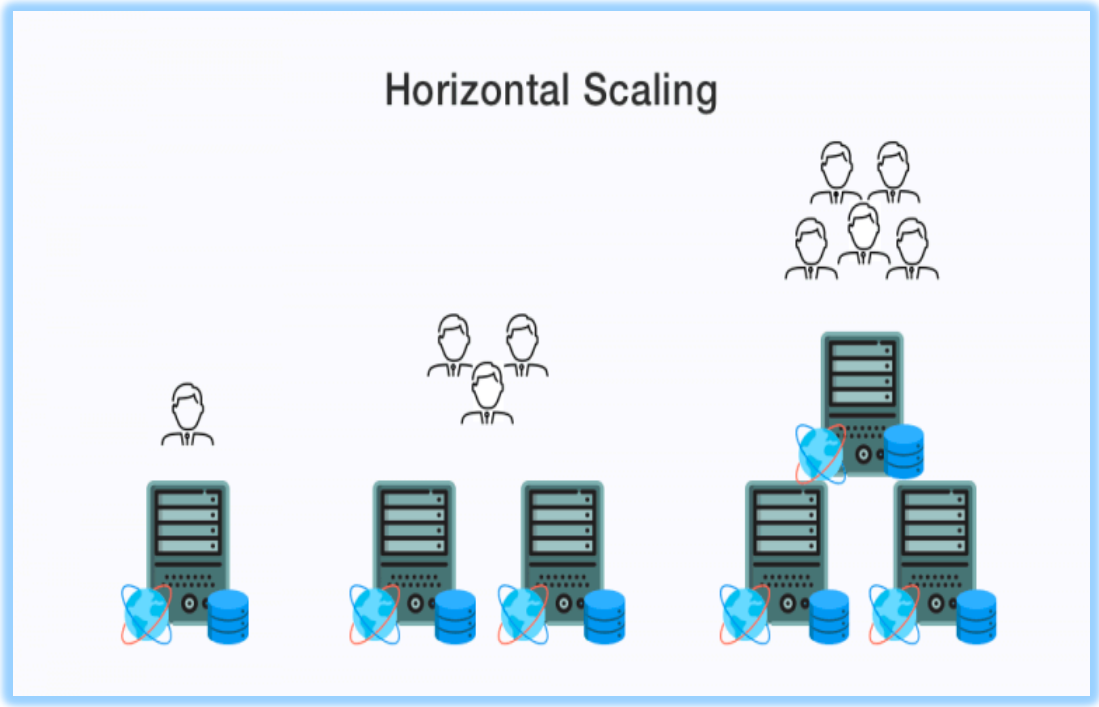
For e.g. growing the number of nodes in the cluster or buying new machines.

Although this does not alter the capacity of each individual node, the load is decreased due to the distribution between separate server nodes.

The response why organizations should choose to scale horizontally include increasing I/O concurrency, reducing the load on existing nodes, and increasing disk capacity.

The Internet and particular **web services** have boosted the use of **Horizontal Scaling**.

Most giant companies that provide well known web services like Google (Gmail, YouTube), Yahoo, Facebook, eBay, Amazon etc. are using heavily horizontal scaling.



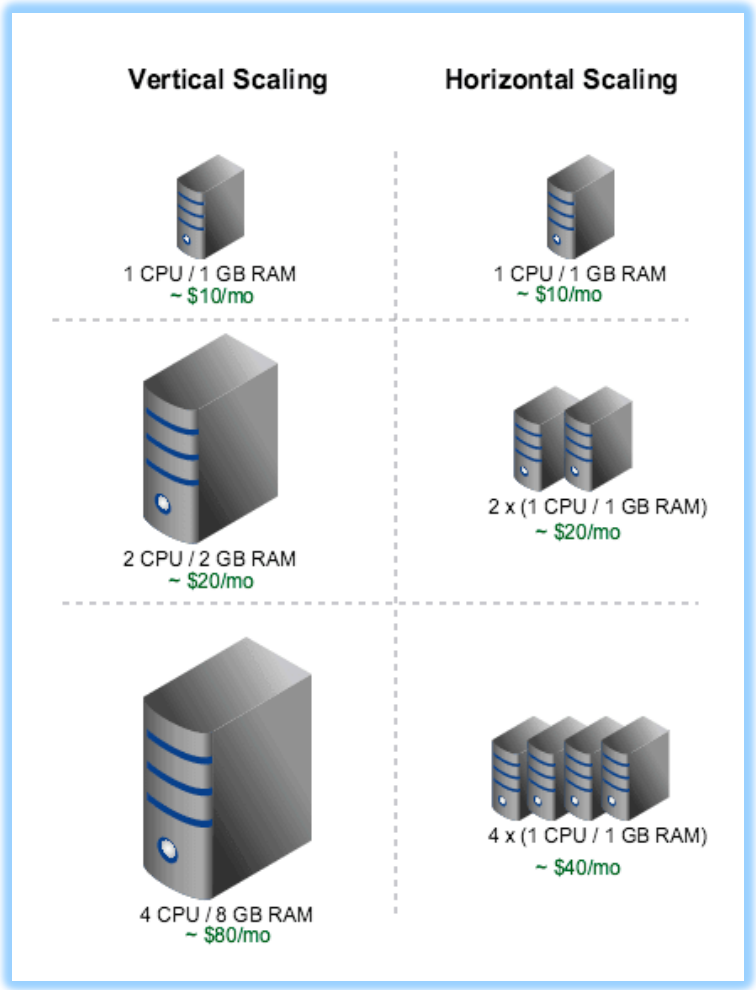
THE DIFFERENCE

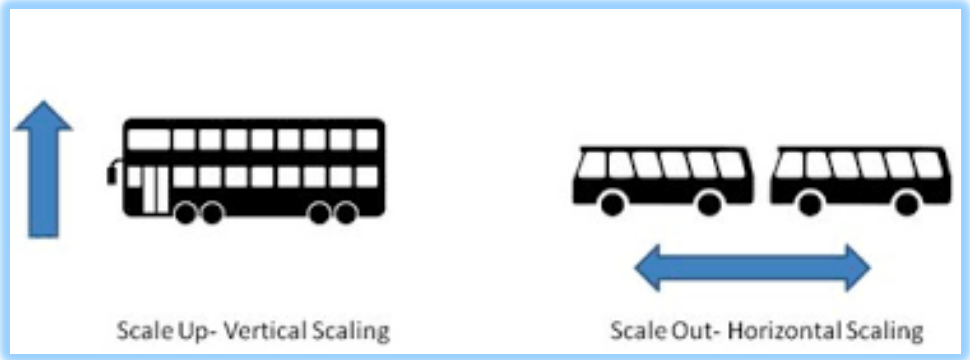
1. **Horizontal-scaling** is often based on partitioning of the data in which each node contains only part of the data while on **vertical-scaling**, the data resides on a single node.
2. **Horizontal-scaling** or scale dynamically is quite easy as you can add more machines into the existing pool. **Vertical-scaling** on the contrary is often limited to the capacity of a single machine.

For example of horizontal scaling is Cassandra, MongoDB and that of vertical scaling is MySQL.

3. **Scaling vertically** can be achieved easily by switching from small to bigger machines. But this involves downtime.

If you need to achieve superior performance issues you can use either **vertical scaling** or horizontal scaling or both in **cloud environments**.





Web Reference Link

WHAT IS A TECH STACK?

A **technology stack**, also called a solutions **stack**, **technology** infrastructure, or a data ecosystem, is a list of all the **technology** services used to build and run one single application.

It is a combination of software applications, frameworks, and programming languages that realize some aspects of the program.

People sometime refer as the combination of programming languages and software underneath a development project in question.

Picking the right combination of underlying development tools is very important in the early stages of a project. When building a skyscraper, you don't start with the marble facade or the fountain in the lobby. You start with a deep foundation and girders to hang everything else on. Your tech stack is like this skeleton and you should consider hiring a cloud architect to help ensure your stack does not fall apart.

A tech stack is the underlying elements of a web or mobile application. These are the frameworks, languages, and software products that everything else is built on. For example, you might have created your web application with Ruby on Rails—that's the language and framework. That might access a database created with PostgreSQL. You'll need to host that on a server, say, an Apache server. You'll need Phusion Passenger to make that happen. Those are all elements of the server-side stack.

The client-side tech stack includes HTML, CSS, and JavaScript. That's what translates your application to a readable format for the user's browser. If you've created a mobile application, the stack is very small: it's usually only a native application, created with something like Xcode or Android Studio.

EXAMPLE - TECH STACK FOR WEB PROJECT DEVELOPMENT

Back-end technologies stack

The backend stack is required to operate smoothly, especially if your project has any features other than simple HTML-coded static pages.

Structure-wise, the backend side consists of the following elements:

1. Programming languages (for e.g. Python, PHP, JavaScript)
2. Frameworks (for e.g. Ruby on Rails, Flask, Django, Swift or Objective-C)
3. Databases (for e.g., MongoDB and MySQL)
4. Server providers (for e.g. Apache, Nginx, etc.)

Front-end technology stack

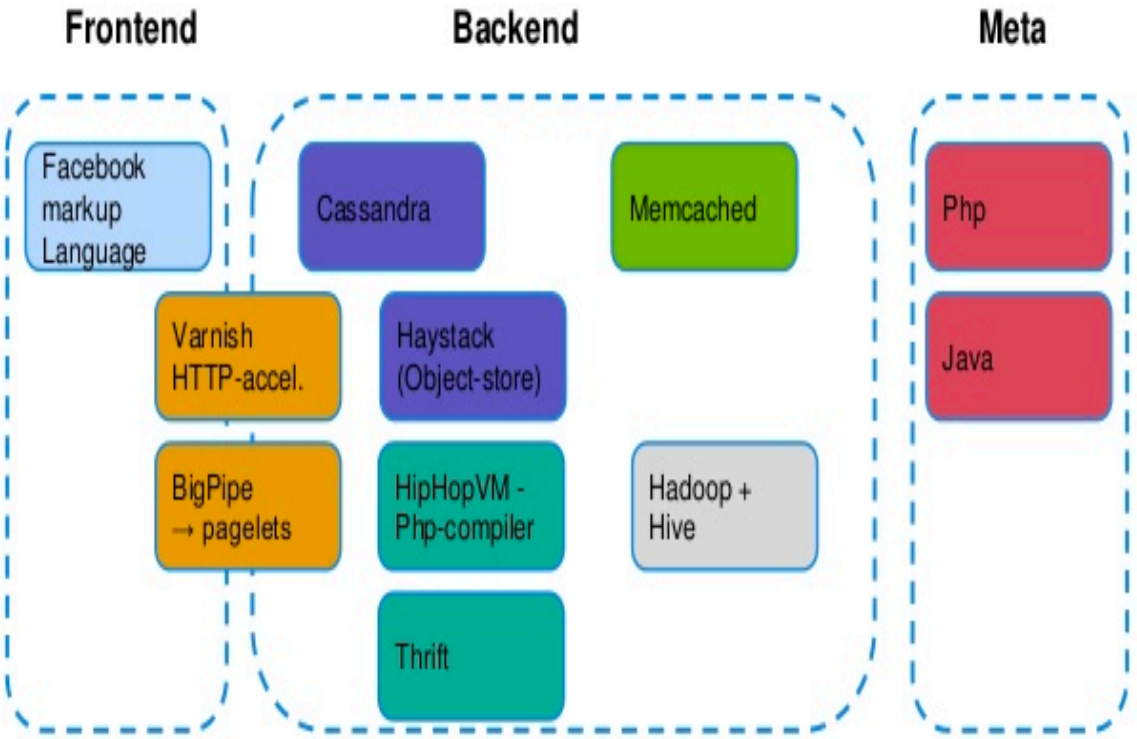
The front-end technology stack is what users see when they interact with the program.

Usually, there are two significant elements in the frontend stack.

1. One of them (HTML) is responsible for the structuring and placement of the content.
2. The other element (CSS) is responsible for the representation of the content. It includes colors, fonts, background stuff, peculiarities of a layout, etc.

If you want to add some interactivity, use JavaScript. These things are controlled via libraries (such as jQuery, React.js, or Zepto.js) that are fitted into frameworks (for example, Ember, Backbone, or Angular).

Facebook



For more tech stack : https://stackshare.io/stacks

WHY YOUR TECHNOLOGY STACK IS IMPORTANT

So your app is built on your tech stack.

But technology used in web application development can have a big effect on how your app works and how it will behave in the future.

> *For example, some server systems are made for high-read operations, but are less efficient when it comes to high-write traffic. That's an important thing to know if you're transitioning your app from local file storage to cloud file storage.*

Your tech stack can also influence the scalability of your product.

Certain stacks will better serve different projects. Because so many different combinations are possible for your tech stack, it's difficult (if not impossible) to generalize.

But getting familiar with the strengths and weaknesses of your tech stack before you start building your product will help you take advantage of the strengths and mitigate the weaknesses.

Web Reference Link

WHAT IS GIT CLI ?

At its core, Git is a set of **command line utility programs** that are designed to execute on a Unix style command-line environment.

Modern operating systems like **Linux** and **macOS** both include built-in Unix command line terminals. This makes Linux and macOS complementary operating systems when working with Git.

In **Windows** environments, Git is often packaged as part of higher level GUI applications. GUIs for Git may attempt to abstract and hide the underlying version control system primitives.

This can be a great aid for Git beginners to rapidly contribute to a project. Once a project's collaboration requirements grow with other team members, ***it is critical to be aware of how the actual raw Git methods work***.

This is when it can be beneficial to drop a GUI version for the command line tools. **Git Bash is offered to provide a terminal Git experience.**

REMOTE VCS REPO

Adding an existing project to GitHub using the command line

Create a new repository on GitHub. To avoid errors, do not initialize the new repository with README, license, or gitignore files. You can add these files after your project has been pushed to GitHub.
Open Git Bash.
Change the current working directory to your local project.

Initialize the local directory as a Git repository.
$ git init

Add the files in your new local repository. This stages them for the first commit.
$ git add

Commit the files that you've staged in your local repository.
$ git commit -m "First commit“

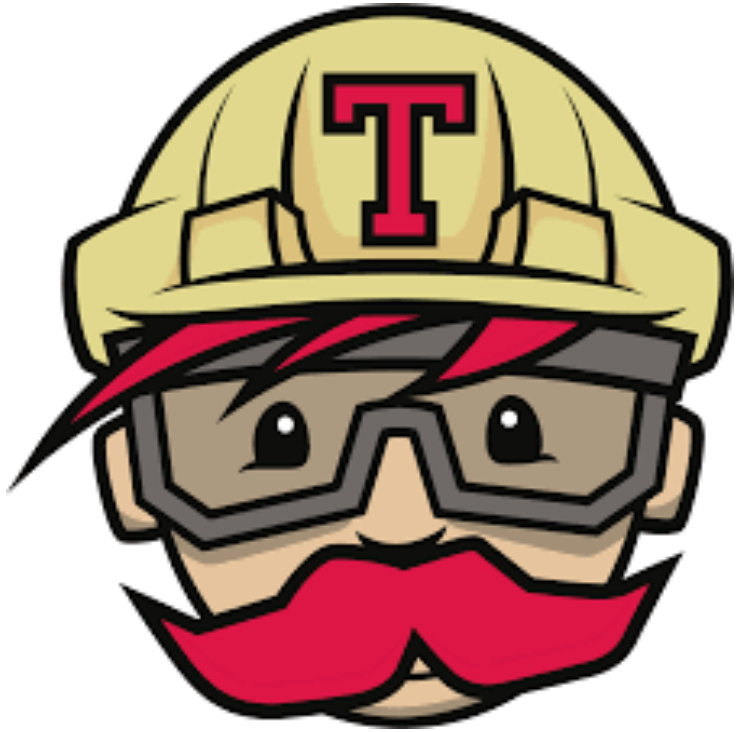
Sets the new remote**.** In the Command prompt, add the URL for the remote repository where your local repository will be pushed.
$ git remote add origin remote repository URL

Verifies the new remote URL
$ git remote -v

Pushes the changes in your local repository up to the remote repository you specified as the origin
$ git push origin master

Web Reference Link

TRAVIS CI



- Continuous integration (CI) is a software development method where members of the team can integrate their work at least once in a day.
- In this , every integration is checked by an automated build to search the error.

- Travis CI is free for open source projects. For commercial projects, you need to purchase an enterprise plan.

Click to see Travis CI documentation

THANK YOU