



School of SRE

Linux Basics

For basic linux commands  [How Linux Works](#) , and network tools in  [Linux](#) .

Git Basics

```
# initialize a git repo
$ git init
Initialized empty Git repository in
/private/tmp/school-of-sre/.git/
```

Adding More Changes

```
$ echo "I am file 2" > file2.txt
$ git add file2.txt
$ git commit -m "adding file 2"
[master 7f3b00e] adding file 2
1 file changed, 1 insertion(+)
create mode 100644 file2.txt
```

two commits we just made are linked via tree like data structure and we saw how they are linked. But let's actually verify it. Everything in git is an object. Newly created files are stored as an object.

Changes to files are stored as objects and even commits are objects. To view the contents of an object we can use the following command with the object's ID. We will take a look at the contents of the second commit

```
$ git cat-file -p 7f3b00e
tree ebf3af44d253e5328340026e45a9fa9ae3ea1982
parent df2fb7a61f5d40c1191e0fdeb0fc5d6e7969685a
author Sanket Patel <spatel1@linkedin.com> 1603273316 -0700
committer Sanket Patel <spatel1@linkedin.com> 1603273316 -0700
```

References and The Magic

```
$ cat .git/HEAD
ref: refs/heads/master
```

```
$ cat .git/refs/heads/master
7f3b00eaa957815884198e2dfec29361108d6a9
```

So the git head master file is pointing to the commit. Whenever git needs to know where master reference is pointing to, or if git needs to update where master points, it just needs to update the file above.

Creating a branch,

```
$ git branch b1
$ git log --oneline --graph
* 7f3b00e (HEAD -> master, b1) adding file 2
* df2fb7a adding file 1
```

Merges

Option 1: Directly merge the branch.

Merging the branch b1 into master will result in a new merge commit. This will merge changes from two different lines of history and create a new commit of the result.

```
$ git merge b1
Merge made by the 'recursive' strategy.
b1.txt | 1 +
1 file changed, 1 insertion(+)
create mode 100644 b1.txt
$ git log --oneline --graph --all
*   8fc28f9 (HEAD -> master) Merge branch 'b1'
|\
| * 872a38f (b1) adding b1 file
* | 60dc441 adding master.txt file
|/
* 7f3b00e adding file 2
* df2fb7a adding file 1
```

Option 2: Rebase.

Instead of merging two branches which have a similar base (commit), let us rebase branch b1 onto the current master. What this means is take branch b1 from a commit and rebase (put them on top of) master.

```
# Switch to b1
$ git checkout b1
Switched to branch 'b1'
```

```
# Rebase (b1 which is current branch) on master
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: adding b1 file
```

```
# The result
$ git log --oneline --graph --all
* 5372c8f (HEAD -> b1) adding b1 file
* 60dc441 (master) adding master.txt file
* 7f3b00e adding file 2
* df2fb7a adding file 1
```

Hooks

Git has another nice feature called hooks. Hooks are basically scripts which will be called when a certain event happens. It's located in ``.git/hooks/``

These hooks are useful when you want to do certain things when a certain event happens. If you want to run tests before pushing code, you would want to setup pre-push hooks.

```
$ ls .git/hooks/
```

```
applypatch-msg.sample      fsmonitor-watchman.sample
pre-applypatch.sample      pre-push.sample
pre-receive.sample         update.sample
commit-msg.sample          post-update.sample
pre-commit.sample          pre-rebase.sample
prepare-commit-msg.sample
```

Some Python Concepts

Decorators

A decorator is a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure.

```
>>> def deco(func):
...     def inner():
...         print("before")
...         func()
...         print("after")
...     return inner
...
>>> @deco
... def hello_world():
...     print("hello world")
...
>>>
>>> hello_world()
before
```

```
hello world
after
```

Sockets

<https://realpython.com/python-sockets/>

RDBMS

Relational DBs are used for data storage.

RDMS are designed to meet these Goals,

- Efficiency
- Ease of access and management
- Organized
- Handle relations between data (represented as tables)

Some common properties and tools used in RDBMS to improve efficiency and integrity of data are listed below.

ACID: Set of properties that guarantee data integrity of DB transactions

- **Atomicity:** Each transaction is atomic (succeeds or fails completely)
- **Consistency:** Transactions only result in valid state (which includes rules, constraints, triggers etc.)
- **Isolation:** Each transaction is executed independently of others safely within a concurrent system
- **Durability:** Completed transactions will not be lost due to any later failures

SQL: A query language to interact with and manage data.

- CRUD operations - create, read, update, delete queries
- Management operations - create DBs/tables/indexes etc, backup, import/export, users, access controls

Constraints: Rules for data that can be stored. Query fails if you violate any of these defined on a table.

- **Primary key:** one or more columns that contain UNIQUE values, and cannot contain NULL values. A table can have only ONE primary key. An index on it is created by default.
- **Foreign key:** links two tables together. Its value(s) match a primary key in a different table \ **Not null:** Does not allow null values \ **Unique:** Value of column must be unique across all rows \ **Default:** Provides a default value for a column if none is specified during insert
- **Check:** Allows only particular values (like Balance >= 0)

Indexes: Most indexes use B+ tree structure. Its Speeds up queries by creating pointers to where data is stored within a database.

- Types of indexes: unique, primary key, fulltext, secondary

Write-heavy loads, mostly full table scans or accessing large number of rows etc. do not benefit from indexes

Joins: Allows you to fetch related data from multiple tables, linking them together with some common field. Powerful but also resource-intensive and makes scaling databases difficult.

- This is the cause of many slow performing queries when run at scale, and the solution is almost always to find ways to reduce the joins.

Access control: DBs have privileged accounts for admin tasks, and regular accounts for clients. There are fine grained controls on what actions(DDL, DML etc. discussed earlier)are allowed for these accounts.

MySQL Replication

Replication enables data from one MySQL host (termed as Primary) to be copied to another MySQL host (termed as Replica). MySQL Replication is asynchronous in nature by default, but it can be changed to semi-synchronous with some configurations.

Some common applications of MySQL replication are:-

- Read-scaling - as multiple hosts can replicate the data from a single primary host, we can set up as many replicas as we need and scale reads through them, i.e. application writes will go to a single primary host and the reads can balance between all the replicas that are there. Such a setup can improve the write performance as well, as the primary is dedicated to only updates and not reads.
- Backups using replicas - the backup process can sometimes be a little heavy. But if we have replicas configured, then we can use one of them to get the backup without affecting the primary data at all.
- Disaster Recovery - a replica in some other geographical region paves a proper path to configure disaster recovery.

NoSQL

There are many types of NoSQL DBs.

- Document databases: They store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values.
- Key-Value databases: These are a simpler type of databases where each item contains keys and values. A value can typically only be retrieved by referencing its key, so learning how to query for a specific key-value pair is typically simple.
- Wide-Column stores: They store data in tables, rows, and dynamic columns. Wide-column stores provide a lot of flexibility over relational databases because each row is not required to have the same columns. Many consider wide-column stores to be two-dimensional key-value databases.
- Graph Databases: These databases store data in nodes and edges. Nodes typically store information about people, places, and things while edges store information about the relationships between the nodes.

Advantages:

- Flexible Data Models
- Horizontal Scaling
- Fast Queries
- Developer productivity

Big Data

Big Data is a collection of large datasets that cannot be processed using traditional computing techniques. It is not a single technique or a tool, rather it has become a complete subject, which involves various tools, techniques, and frameworks.

Systems Design

Scalability

A service is said to be scalable if, as resources are added to the system, it results in increased performance in a manner proportional to resources added.

- **Horizontal scaling**
Horizontal scaling stands for cloning of an application or service such that work can easily be distributed across instances with absolutely no bias.
- **Microservices**
This pattern represents the separation of work by service or function within the application. Microservices are meant to address the issues associated with growth and complexity in the code base and data sets. The intent is to create fault isolation as well as to reduce response times.
- **Sharding**
This pattern represents the separation of work based on attributes that are looked up to or determined at the time of the transaction. Most often, these are implemented as splits by requestor, customer, or client.

Load Balancing

Improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives. A commonly used technique is load balancing traffic across identical server clusters

LB Tasks

Service discovery:

LB acts as a single endpoint that clients can use transparently to reach one of the 4 servers.

Health checking:

What backends are currently healthy and available to accept requests? If one out of the 4 App servers turns bad, LB should automatically short circuit the path so that clients don't sense any application downtime

Load balancing:

What algorithm should be used to balance individual requests across the healthy backends? There are many algorithms to distribute traffic across one of the four servers.

LB Methods

Least Connection Method

directs traffic to the server with the fewest active connections.

Least Response Time Method

directs traffic to the server with the fewest active connections and the lowest average response time. Here response time is used to provide feedback of the server's health

Round Robin Method

rotates servers by directing traffic to the first available server and then moves that server to the bottom of the queue.

IP Hash

the IP address of the client determines which server receives the request. This can sometimes cause skewness in distribution but is useful if apps store some state locally and need some stickiness

Content Delivery Networks (CDN)

CDNs are added closer to the client's location. If the app has static data like images, Javascript, CSS which don't change very often, they can be cached.

Fault Tolerance

Common failure metrics that get measured and tracked for any system.

Mean time to repair (MTTR), Mean time between failures (MTBF) etc..

https://www.splunk.com/en_us/data-insider/what-is-mean-time-to-repair.html

Swimlane is one of the commonly used fault isolation methodologies. Swimlane adds a barrier to the service from other services so that failure on either of them won't affect the other.

Swimlane Principles

Principle 1: Nothing is shared (also known as "share as little as possible"). The less that is shared within a swim lane, the more fault isolative the swim lane becomes. (as shown in Enterprise use-case)

Principle 2: Nothing crosses a swim lane boundary. Synchronous (defined by expecting a request—not the transfer protocol) communication never crosses a swim lane boundary; if it does, the boundary is drawn incorrectly.

Introduction To Containers

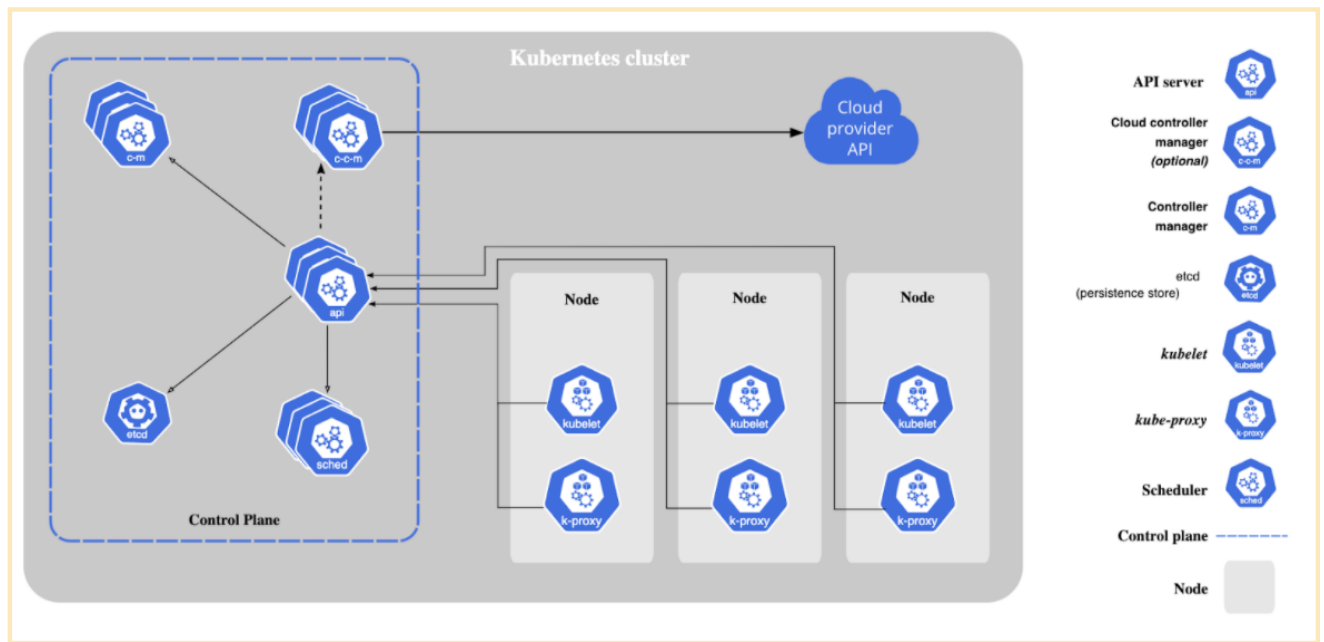
A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another

How are containers implemented

See: [📺 Docker: A Deep Dive](#)

Orchestration With Kubernetes

[Architecture of Kubernetes](#)



The brain of Kubernetes which responds to events from the node plane (e.g. create a pod, replicas mismatch) and does the main orchestration is called the control plane. All control plane components are typically installed in a master node. This master node does not run any user containers.

The Kubernetes components themselves are run as containers wrapped in Pods (which is the most basic Kubernetes resource object).

- Control plane components:
- kube-apiserver
- etcd
- kube-scheduler

- kube-controller-manager
- Node plane components
- kubelet
- kube-proxy

Large System Design

However, when it is time to implement the system, especially as an SRE we have no other choice but to think in specific terms. Servers have a fixed amount of memory, storage capacity and processing power. So we need to think about the realistic expectations from our system, assess the requirements, translate them into specific requirements from each component of the system like network, storage and compute.

Google has formalized this approach to designing systems as 'Non abstract large system design' (NALSD).

Read the whole thing,

https://linkedin.github.io/school-of-sre/level102/system_design/large-system-design/

Performance Improvements

Profiling is an important part of performance analysis of the service. There are various profiler tools available,

- FlameGraph: Flame graphs are a visualization of profiled software, allowing the most frequent code-paths to be identified quickly and accurately.
- Valgrind: It is a programming tool for memory debugging, memory leak detection, and profiling.
- Gprof: GNU profiler tool uses a hybrid of instrumentation and sampling. Instrumentation is used to collect function call information, and sampling is used to gather runtime profiling information.

Benchmarking

It is a process of measuring the best performance of the service. Like how much QPS service can handle, its latency when load is increasing, host resource utilization, loadavg etc etc.

Some of known tools,

- Apache Benchmark Tool, ab: It simulate a high load on webapp and gather data for analysis
- Httpperf: It sends requests to the web server at a specified rate and gathers stats. Increase till one finds the saturation point.
- Apache JMeter: It is a popular open-source tool to measure web application performance. JMeter is a java based application and not only a web server, but you can use it against PHP, Java, REST, etc.
- Wrk: It is another modern performance measurement tool to put a load on your web server and give you latency, request per second, transfer per second, etc. details.
- Locust: Easy to use, scriptable and scalable performance testing tool.

Above tools help in synthetic load or stress testing, but such does not measure actual end user experience, It can't see how end user resources will affect application performance.

CI/CD

Continuous Integration and Continuous Delivery, also known as CI/CD, is a set of processes that helps in faster integration of software code changes and deployment to the end user in a reliable manner.

Continuous Integration is a software development practice where members of a team integrate their work frequently. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.

Continuous Delivery means deploying the application builds more frequently in the non-production environments such as dev, UAT, QA and performing the integration tests and the acceptance tests automatically.



Setup Jenkins based CI/CD Pipeline,

<https://github.com/neeraj9194/devops-roadmap#cicd-with-jenkins>