

Unit -2

APACHE HADOOP :

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

Apache Hadoop and Apache Spark are both open-source frameworks for big data processing with some key differences. Hadoop uses the MapReduce to process data, while Spark uses resilient distributed datasets (RDDs)

HADOOP MAPREDUCE JOB EXECUTION :

In this **Hadoop** blog, we are going to provide you an end to end MapReduce job execution flow. Here we will describe each component which is the part of MapReduce working in detail.

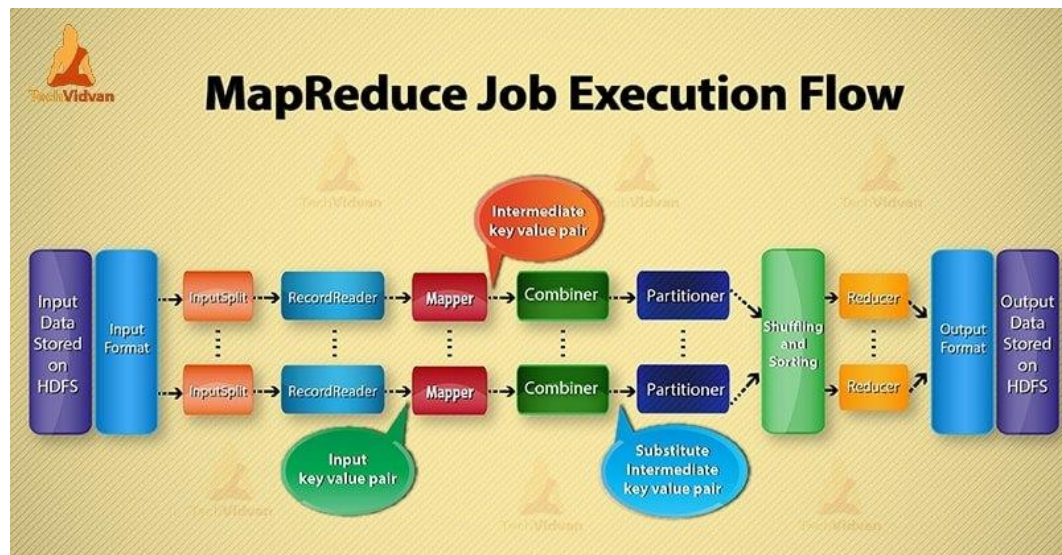
This blog will help you to answer how Hadoop MapReduce work, how data flows in MapReduce, how Mapreduce job is executed in Hadoop?

WHAT IS MAPREDUCE?

Hadoop MapReduce is the data processing layer. It processes the huge amount of structured and unstructured data stored in HDFS. MapReduce processes data in parallel by dividing the job into the set of independent tasks. So, parallel processing improves speed and reliability.

Hadoop MapReduce data processing takes place in 2 phases- Map and Reduce phase.

- **Map phase-** It is the first phase of data processing. In this phase, we specify all the complex logic/business rules/costly code.
- **Reduce phase-** It is the second phase of processing. In this phase, we specify light-weight processing like aggregation/summation.



STEPS OF MAPREDUCE JOB EXECUTION FLOW

MapReduce processes the data in various phases with the help of different components. Let's discuss the steps of job execution in Hadoop.

1. Input Files

In input files data for MapReduce job is stored. In **HDFS**, input files reside. Input files format is arbitrary. Line-based log files and binary format can also be used.

2. InputFormat

After that InputFormat defines how to split and read these input files. It selects the files or other objects for input. InputFormat creates InputSplit.

3. InputSplits

It represents the data which will be processed by an individual **Mapper**. For each split, one map task is created. Thus the number of map tasks is equal to the number of InputSplits. Framework divide split into records, which mapper process.

4. RecordReader

It communicates with the inputSplit. And then converts the data into **key-value pairs** suitable for reading by the Mapper. RecordReader by default uses TextInputFormat to convert data into a key-value pair.

It communicates to the InputSplit until the completion of file reading. It assigns byte offset to each line present in the file. Then, these key-value pairs are further sent to the mapper for further processing.

5. Mapper

It processes input record produced by the RecordReader and generates intermediate key-value pairs. The intermediate output is completely different from the input pair. The output of the mapper is the full collection of key-value pairs.

Hadoop framework doesn't store the output of mapper on HDFS. It doesn't store, as data is temporary and writing on HDFS will create unnecessary multiple copies. Then Mapper passes the output to the combiner for further processing.

4. Combiner

Combiner is Mini-reducer which performs local aggregation on the mapper's output. It minimizes the data transfer between mapper and reducer. So, when the combiner functionality completes, framework passes the output to the partitioner for further processing.

5. Partitioner

Partitioner comes into the existence if we are working with more than one reducer. It takes the output of the combiner and performs partitioning.

Partitioning of output takes place on the basis of the key in MapReduce. By hash function, key (or a subset of the key) derives the partition.

On the basis of key value in MapReduce, partitioning of each combiner output takes place. And then the record having the same key value goes into the same partition. After that, each partition is sent to a reducer.

Partitioning in MapReduce execution allows even distribution of the map output over the reducer.

6. Shuffling and Sorting

After partitioning, the output is shuffled to the reduce node. The shuffling is the physical movement of the data which is done over the network. As all the mappers finish and shuffle the output on the reducer nodes.

Then framework merges this intermediate output and sort. This is then provided as input to reduce phase.

7. Reducer

Reducer then takes set of intermediate key-value pairs produced by the mappers as the input. After that runs a reducer function on each of them to generate the output.

The output of the reducer is the final output. Then framework stores the output on HDFS.

8. RecordWriter

It writes these output key-value pair from the Reducer phase to the output files.

9. OutputFormat

OutputFormat defines the way how RecordReader writes these output key-value pairs in output files. So, its instances provided by the Hadoop write files in HDFS. Thus OutputFormat instances write the final output of reducer on HDFS.

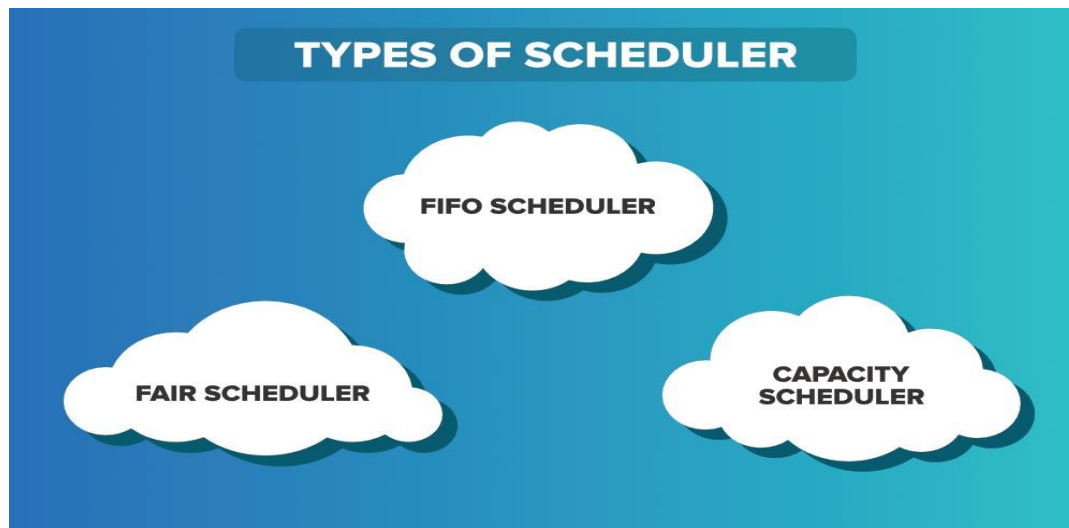
HADOOP SCHEDULERS :

In Hadoop, we can receive multiple jobs from different clients to perform. The Map-Reduce framework is used to perform multiple tasks in parallel in a typical Hadoop cluster to process large size datasets at a fast rate. This Map-Reduce Framework is responsible for scheduling and monitoring the tasks given by different clients in a Hadoop cluster. But this method of scheduling jobs is used prior to **Hadoop 2**.

Now in Hadoop 2, we have YARN (Yet Another Resource Negotiator). In YARN we have separate Daemons for performing Job scheduling, Monitoring, and Resource Management as Application Master, Node Manager, and Resource Manager respectively.

Here, Resource Manager is the Master Daemon responsible for tracking or providing the resources required by any application within the cluster, and Node Manager is the slave Daemon which monitors and keeps track of the resources used by an application and sends the feedback to Resource Manager.

Schedulers and **Applications Manager** are the 2 major components of resource Manager. The Scheduler in YARN is totally dedicated to scheduling the jobs, it can not track the status of the application. On the basis of required resources, the scheduler performs or we can say schedule the Jobs.



There are mainly 3 types of Schedulers in Hadoop:

1. FIFO (First In First Out) Scheduler.
2. Capacity Scheduler.
3. Fair Scheduler.

These Schedulers are actually a kind of algorithm that we use to schedule tasks in a Hadoop cluster when we receive requests from different-different clients.

A **Job queue** is nothing but the collection of various tasks that we have received from our various clients. The tasks are available in the queue and we need to schedule this task on the basis of our requirements.



1. FIFO SCHEDULER

As the name suggests FIFO i.e. First In First Out, so the tasks or application that comes first will be served first. This is the default Scheduler we use in Hadoop. The tasks are placed in a queue and the tasks are performed in their submission order. In this method, once the job is scheduled, no intervention is allowed. So sometimes the high-priority process has to wait for a long time since the priority of the task does not matter in this method.

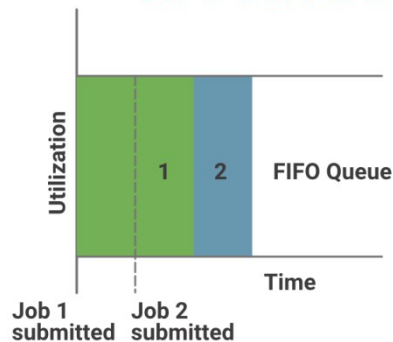
Advantage:

- No need for configuration
- First Come First Serve
- simple to execute

Disadvantage:

- Priority of task doesn't matter, so high priority jobs need to wait
- Not suitable for shared cluster

FIFO SCHEDULER



2. CAPACITY SCHEDULER

In Capacity Scheduler we have multiple job queues for scheduling our tasks. The Capacity Scheduler allows multiple occupants to share a large size Hadoop cluster. In Capacity Scheduler corresponding for each job queue, we provide some slots or cluster resources for performing job operation. Each job queue has its own slots to perform its task. In case we have tasks to perform in only one queue then the tasks of that queue can access the slots of other queues also as they are free to use, and when the new task enters to some other queue then jobs in running in its own slots of the cluster are replaced with its own job.

Capacity Scheduler also provides a level of abstraction to know which occupant is utilizing the more cluster resource or slots, so that the single user or application doesn't take inappropriate or unnecessary slots in the cluster. The capacity Scheduler mainly contains 3 types of the queue that are root, parent, and leaf which are used to represent cluster, organization, or any subgroup, application submission respectively.

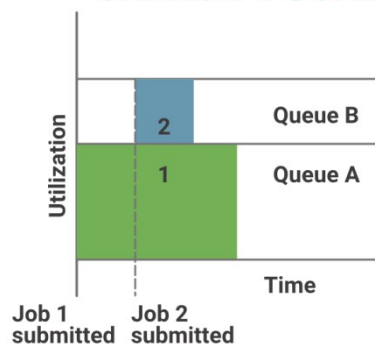
Advantage:

- Best for working with Multiple clients or priority jobs in a Hadoop cluster
- Maximizes throughput in the Hadoop cluster

Disadvantage:

- More complex
- Not easy to configure for everyone

CAPACITY SCHEDULER



3. FAIR SCHEDULER

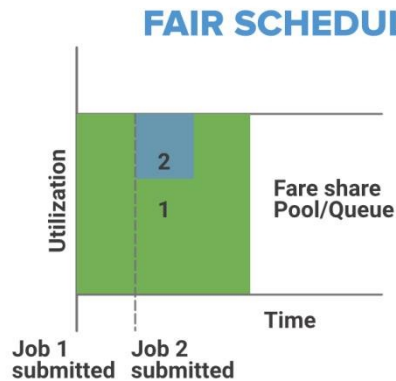
The Fair Scheduler is very much similar to that of the capacity scheduler. The priority of the job is kept in consideration. With the help of Fair Scheduler, the YARN applications can share the resources in the large Hadoop Cluster and these resources are maintained dynamically so no need for prior capacity. The resources are distributed in such a manner that all applications within a cluster get an equal amount of time. Fair Scheduler takes Scheduling decisions on the basis of memory, we can configure it to work with CPU also.

As we told you it is similar to Capacity Scheduler but the major thing to notice is that in Fair Scheduler whenever any high priority job arises in the same queue, the task is processed in parallel by replacing some portion from the already dedicated slots.

Advantages:

- Resources assigned to each application depend upon its priority.
- it can limit the concurrent running task in a particular pool or queue.

Disadvantages: The configuration is required.



HADOOP CLUSTER SETUP:

PURPOSE

This document describes how to install and configure Hadoop clusters ranging from a few nodes to extremely large clusters with thousands of nodes. To play with Hadoop, you may first want to install it on a single machine (see [Single Node Setup](#)).

This document does not cover advanced topics such as [Security](#) or High Availability.

PREREQUISITES

- Install Java. See the [Hadoop Wiki](#) for known good versions.
- Download a stable version of Hadoop from Apache mirrors.

INSTALLATION

Installing a Hadoop cluster typically involves unpacking the software on all the machines in the cluster or installing it via a packaging system as appropriate for your operating system. It is important to divide up the hardware into functions.

Typically one machine in the cluster is designated as the NameNode and another machine as the ResourceManager, exclusively. These are the masters. Other services (such as Web App Proxy Server and MapReduce Job History server) are usually run either on dedicated hardware or on shared infrastructure, depending upon the load.

The rest of the machines in the cluster act as both DataNode and NodeManager. These are the workers.

CONFIGURING HADOOP IN NON-SECURE MODE

Hadoop's Java configuration is driven by two types of important configuration files:

- Read-only default configuration - core-default.xml, hdfs-default.xml, yarn-default.xml and mapred-default.xml.
- Site-specific configuration - etc/hadoop/core-site.xml, etc/hadoop/hdfs-site.xml, etc/hadoop/yarn-site.xml and etc/hadoop/mapred-site.xml.

Additionally, you can control the Hadoop scripts found in the bin/ directory of the distribution, by setting site-specific values via the etc/hadoop/hadoop-env.sh and etc/hadoop/yarn-env.sh.

To configure the Hadoop cluster you will need to configure the environment in which the Hadoop daemons execute as well as the configuration parameters for the Hadoop daemons.

HDFS daemons are NameNode, SecondaryNameNode, and DataNode. YARN daemons are ResourceManager, NodeManager, and WebAppProxy. If MapReduce is to be used, then the MapReduce Job History Server will also be running. For large installations, these are generally running on separate hosts.

CONFIGURING ENVIRONMENT OF HADOOP DAEMONS

Administrators should use the etc/hadoop/hadoop-env.sh and optionally the etc/hadoop/mapred-env.sh and etc/hadoop/yarn-env.sh scripts to do site-specific customization of the Hadoop daemons' process environment.

At the very least, you must specify the JAVA_HOME so that it is correctly defined on each remote node.

Administrators can configure individual daemons using the configuration options shown below in the table:

Daemon	Environment Variable
NameNode	HDFS_NAMENODE_OPTS
DataNode	HDFS_DATANODE_OPTS
Secondary NameNode	HDFS_SECONDARYNAMENODE_OPTS
ResourceManager	YARN_RESOURCEMANAGER_OPTS
NodeManager	YARN_NODEMANAGER_OPTS
WebAppProxy	YARN_PROXYSERVER_OPTS
Map Reduce Job History Server	MAPRED_HISTORYSERVER_OPTS

For example, To configure Namenode to use parallelGC and a 4GB Java Heap, the following statement should be added in `hadoop-env.sh` :

```
export HDFS_NAMENODE_OPTS="-XX:+UseParallelGC -Xmx4g"
```

See `etc/hadoop/hadoop-env.sh` for other examples.

Other useful configuration parameters that you can customize include:

- `HADOOP_PID_DIR` - The directory where the daemons' process id files are stored.
- `HADOOP_LOG_DIR` - The directory where the daemons' log files are stored. Log files are automatically created if they don't exist.
- `HADOOP_HEAPSIZE_MAX` - The maximum amount of memory to use for the Java heapsize. Units supported by the JVM are also supported here. If no unit is present, it will be assumed the number is in megabytes. By default, Hadoop will let the JVM determine how much to use. This value can be overridden on a per-daemon basis using the appropriate `_OPTS` variable listed above. For example, setting `HADOOP_HEAPSIZE_MAX=1g` and `HADOOP_NAMENODE_OPTS="-Xmx5g"` will configure the NameNode with 5GB heap.

In most cases, you should specify the `HADOOP_PID_DIR` and `HADOOP_LOG_DIR` directories such that they can only be written to by the users that are going to run the hadoop daemons. Otherwise there is the potential for a symlink attack. It is also traditional to configure `HADOOP_HOME` in the system-wide shell environment configuration. For example, a simple script inside `/etc/profile.d`:

```
HADOOP_HOME=/path/to/hadoop
export HADOOP_HOME
```

CONFIGURING THE HADOOP DAEMONS

This section deals with important parameters to be specified in the given configuration files:

- `etc/hadoop/core-site.xml`

Parameter	Value	Notes
<code>fs.defaultFS</code>	NameNode URI	hdfs://host:port/
<code>io.file.buffer.size</code>	131072	Size of read/write buffer used in SequenceFiles.

- `etc/hadoop/hdfs-site.xml`
- Configurations for NameNode:

Parameter	Value	Notes
<code>dfs.namenode.name.dir</code>	Path on the local filesystem where the NameNode stores the namespace and transactions logs persistently.	If this is a comma-delimited list of directories then the name table is replicated in all of the directories, for redundancy.
<code>dfs.hosts</code> / <code>dfs.hosts.exclude</code>	List of permitted/excluded DataNodes.	If necessary, use these files to control the list of allowable datanodes.
<code>dfs.blocksize</code>	268435456	HDFS blocksize of 256MB for large file-systems.
<code>dfs.namenode.handler.count</code>	100	More NameNode server threads to handle

Parameter	Value	Notes
		RPCs from large number of DataNodes.

- Configurations for DataNode:

Parameter	Value	Notes
dfs.datanode.data.dir	Comma separated list of paths on the local filesystem of a DataNode where it should store its blocks.	If this is a comma-delimited list of directories, then data will be stored in all named directories, typically on different devices.

- etc/hadoop/yarn-site.xml
- Configurations for ResourceManager and NodeManager:

Parameter	Value	Notes
yarn.acl.enable	true / false	Enable ACLs? Defaults to <i>false</i> .
yarn.admin.acl	Admin ACL	ACL to set admins on the cluster. ACLs are of for <i>comma-separated-usersspacecomma-separated-groups</i> . Defaults to special value of * which means <i>anyone</i> . Special value of just <i>space</i> means no one has access.
yarn.log-aggregation-enable	<i>false</i>	Configuration to enable or disable log aggregation

- Configurations for ResourceManager:

Parameter	Value	Notes
yarn.resourcemanager.address	ResourceManager host:port for clients to submit jobs.	<i>host:port</i> If set, overrides the hostname set in yarn.resourcemanager.hostname.
yarn.resourcemanager.scheduler.address	ResourceManager host:port for ApplicationMasters to talk to Scheduler to obtain resources.	<i>host:port</i> If set, overrides the hostname set in yarn.resourcemanager.hostname.
yarn.resourcemanager.resource-tracker.address	ResourceManager host:port for NodeManagers.	<i>host:port</i> If set, overrides the hostname set in yarn.resourcemanager.hostname.
yarn.resourcemanager.admin.address	ResourceManager host:port for administrative commands.	<i>host:port</i> If set, overrides the hostname set in yarn.resourcemanager.hostname.
yarn.resourcemanager.webapp.address	ResourceManager web-ui host:port.	<i>host:port</i> If set, overrides the hostname set in yarn.resourcemanager.hostname.
yarn.resourcemanager.hostname	ResourceManager host.	<i>host</i> Single hostname that can be set in place of setting all yarn.resourcemanager*address resources. Results in default ports for ResourceManager components.
yarn.resourcemanager.scheduler.class	ResourceManager Scheduler class.	CapacityScheduler (recommended), FairScheduler (also recommended), or FifoScheduler. Use a fully qualified class name, e.g., org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler.
yarn.scheduler.minimum-allocation-mb	Minimum limit of memory to allocate to each container request at the Resource Manager.	In MBs
yarn.scheduler.maximum-allocation-mb	Maximum limit of memory to allocate	In MBs

Parameter	Value	Notes
	to each container request at the Resource Manager.	
yarn.resourcemanager.nodes.include-path / yarn.resourcemanager.nodes.exclude-path	List of permitted/excluded NodeManagers.	If necessary, use these files to control the list of allowable NodeManagers.

- Configurations for NodeManager:

Parameter	Value	Notes
yarn.nodemanager.resource.memory-mb	Resource i.e. available physical memory, in MB, for given NodeManager	Defines total available resources on the NodeManager to be made available to running containers
yarn.nodemanager.vmem-pmem-ratio	Maximum ratio by which virtual memory usage of tasks may exceed physical memory	The virtual memory usage of each task may exceed its physical memory limit by this ratio. The total amount of virtual memory used by tasks on the NodeManager may exceed its physical memory usage by this ratio.
yarn.nodemanager.local-dirs	Comma-separated list of paths on the local filesystem where intermediate data is written.	Multiple paths help spread disk i/o.
yarn.nodemanager.log-dirs	Comma-separated list of paths on the	Multiple paths help spread disk i/o.

Parameter	Value	Notes
	local filesystem where logs are written.	
yarn.nodemanager.log.retain-seconds	10800	Default time (in seconds) to retain log files on the NodeManager Only applicable if log-aggregation is disabled.
yarn.nodemanager.remote-app-log-dir	/logs	HDFS directory where the application logs are moved on application completion. Need to set appropriate permissions. Only applicable if log-aggregation is enabled.
yarn.nodemanager.remote-app-log-dir-suffix	logs	Suffix appended to the remote log dir. Logs will be aggregated to \${yarn.nodemanager.remote-app-log-dir}/\${user}/\${thisParam} Only applicable if log-aggregation is enabled.
yarn.nodemanager.aux-services	mapreduce_shuffle	Shuffle service that needs to be set for Map Reduce applications.
yarn.nodemanager.env-whitelist	Environment properties to be inherited by containers from NodeManagers	For mapreduce application in addition to the default values HADOOP_MAPRED_HOME should be added. Property value should JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,CLASSPATH_PREPEND_DISTCACHE,HADOOP_YARN_HOME,HADOOP_HOME,PATh,LANG,TZ,HADOOP_MAPRED_HOME

- Configurations for History Server (Needs to be moved elsewhere):

Parameter	Value	Notes
yarn.log-aggregation.retain-seconds	-1	How long to keep aggregation logs before deleting them. -1 disables. Be careful, set this too small and you will spam the name node.
yarn.log-aggregation.retain-check-interval-seconds	-1	Time between checks for aggregated log retention. If set to 0 or a negative value then the value is computed as one-tenth of the aggregated log retention time. Be careful, set this too small and you will spam the name node.

- etc/hadoop/mapred-site.xml
- Configurations for MapReduce Applications:

Parameter	Value	Notes
mapreduce.framework.name	yarn	Execution framework set to Hadoop YARN.
mapreduce.map.memory.mb	1536	Larger resource limit for maps.
mapreduce.map.java.opts	-Xmx1024M	Larger heap-size for child jvms of maps.
mapreduce.reduce.memory.mb	3072	Larger resource limit for reduces.
mapreduce.reduce.java.opts	-Xmx2560M	Larger heap-size for child jvms of reduces.
mapreduce.task.io.sort.mb	512	Higher memory-limit while sorting data for efficiency.
mapreduce.task.io.sort.factor	100	More streams merged at once while sorting files.
mapreduce.reduce.shuffle.parallelcopies	50	Higher number of parallel copies run by reduces to fetch outputs from very large number of maps.

- Configurations for MapReduce JobHistory Server:

Parameter	Value	Notes
mapreduce.jobhistory.address	MapReduce JobHistory Server <i>host:port</i>	Default port is 10020.
mapreduce.jobhistory.webapp.address	MapReduce JobHistory Server Web UI <i>host:port</i>	Default port is 19888.
mapreduce.jobhistory.intermediate-done-dir	/mr-history/tmp	Directory where history files are written by MapReduce jobs.
mapreduce.jobhistory.done-dir	/mr-history/done	Directory where history files are managed by the MR JobHistory Server.

MONITORING HEALTH OF NODEMANAGERS

Hadoop provides a mechanism by which administrators can configure the NodeManager to run an administrator supplied script periodically to determine if a node is healthy or not.

Administrators can determine if the node is in a healthy state by performing any checks of their choice in the script. If the script detects the node to be in an unhealthy state, it must print a line to standard output beginning with the string ERROR. The NodeManager spawns the script periodically and checks its output. If the script's output contains the string ERROR, as described above, the node's status is reported as unhealthy and the node is black-listed by the ResourceManager. No further tasks will be assigned to this node. However, the NodeManager continues to run the script, so that if the node becomes healthy again, it will be removed from the blacklisted nodes on the ResourceManager automatically. The node's health along with the output of the script, if it is unhealthy, is available to the administrator in the ResourceManager web interface. The time since the node was healthy is also displayed on the web interface.

The following parameters can be used to control the node health monitoring script in etc/hadoop/yarn-site.xml.

Parameter	Value	Notes
yarn.nodemanager.health-checker.script.path	Node health script	Script to check for node's health status.
yarn.nodemanager.health-checker.script.opts	Node health script options	Options for script to check for node's health status.
yarn.nodemanager.health-checker.interval-ms	Node health script interval	Time interval for running health script.
yarn.nodemanager.health-checker.script.timeout-ms	Node health script timeout interval	Timeout for health script execution.

The health checker script is not supposed to give ERROR if only some of the local disks become bad. NodeManager has the ability to periodically check the health of the local disks (specifically checks nodemanager-local-dirs and nodemanager-log-dirs) and after reaching the threshold of number of bad directories based on the value set for the config property yarn.nodemanager.disk-health-checker.min-healthy-disks, the whole node is marked unhealthy and this info is sent to resource manager also. The boot disk is either raided or a failure in the boot disk is identified by the health checker script.

SLAVES FILE

List all worker hostnames or IP addresses in your etc/hadoop/workers file, one per line. Helper scripts (described below) will use the etc/hadoop/workers file to run commands on many hosts at once. It is not used for any of the Java-based Hadoop configuration. In order to use this functionality, ssh trusts (via either passphraseless ssh or some other means, such as Kerberos) must be established for the accounts used to run Hadoop.

HADOOP RACK AWARENESS

Many Hadoop components are rack-aware and take advantage of the network topology for performance and safety. Hadoop daemons obtain the rack information of the workers in the cluster by invoking an administrator configured module. See the [Rack Awareness](#) documentation for more specific information.

It is highly recommended configuring rack awareness prior to starting HDFS.

LOGGING

Hadoop uses the [Apache log4j](#) via the Apache Commons Logging framework for logging. Edit the etc/hadoop/log4j.properties file to customize the Hadoop daemons' logging configuration (log-formats and so on).

OPERATING THE HADOOP CLUSTER

Once all the necessary configuration is complete, distribute the files to the HADOOP_CONF_DIR directory on all the machines. This should be the same directory on all machines.

In general, it is recommended that HDFS and YARN run as separate users. In the majority of installations, HDFS processes execute as 'hdfs'. YARN is typically using the 'yarn' account.

HADOOP STARTUP

To start a Hadoop cluster you will need to start both the HDFS and YARN cluster.

The first time you bring up HDFS, it must be formatted. Format a new distributed filesystem as *hdfs*:

```
[hdfs]$ $HADOOP_HOME/bin/hdfs namenode -format
```

Start the HDFS NameNode with the following command on the designated node as *hdfs*:

```
[hdfs]$ $HADOOP_HOME/bin/hdfs --daemon start namenode
```

Start a HDFS DataNode with the following command on each designated node as *hdfs*:

```
[hdfs]$ $HADOOP_HOME/bin/hdfs --daemon start datanode
```

If `etc/hadoop/workers` and `ssh` trusted access is configured (see [Single Node Setup](#)), all of the HDFS processes can be started with a utility script. As *hdfs*:

```
[hdfs]$ $HADOOP_HOME/sbin/start-dfs.sh
```

Start the YARN with the following command, run on the designated ResourceManager as *yarn*:

```
[yarn]$ $HADOOP_HOME/bin/yarn --daemon start resourcemanager
```

Run a script to start a NodeManager on each designated host as *yarn*:

```
[yarn]$ $HADOOP_HOME/bin/yarn --daemon start nodemanager
```

Start a standalone WebAppProxy server. Run on the WebAppProxy server as *yarn*. If multiple servers are used with load balancing it should be run on each of them:

```
[yarn]$ $HADOOP_HOME/bin/yarn --daemon start proxyserver
```

If `etc/hadoop/workers` and `ssh` trusted access is configured (see [Single Node Setup](#)), all of the YARN processes can be started with a utility script. As *yarn*:

```
[yarn]$ $HADOOP_HOME/sbin/start-yarn.sh
```

Start the MapReduce JobHistory Server with the following command, run on the designated server as *mapred*:

```
[mapred]$ $HADOOP_HOME/bin/mapred --daemon start historyserver
```

HADOOP SHUTDOWN

Stop the NameNode with the following command, run on the designated NameNode as *hdfs*:

```
[hdfs]$ $HADOOP_HOME/bin/hdfs --daemon stop namenode
```

Run a script to stop a DataNode as *hdfs*:

```
[hdfs]$ $HADOOP_HOME/bin/hdfs --daemon stop datanode
```

If `etc/hadoop/workers` and `ssh` trusted access is configured (see [Single Node Setup](#)), all of the HDFS processes may be stopped with a utility script. As *hdfs*:

```
[hdfs]$ $HADOOP_HOME/sbin/stop-dfs.sh
```

Stop the ResourceManager with the following command, run on the designated ResourceManager as *yarn*:

```
[yarn]$ $HADOOP_HOME/bin/yarn --daemon stop resourcemanager
```

Run a script to stop a NodeManager on a worker as *yarn*:

```
[yarn]$ $HADOOP_HOME/bin/yarn --daemon stop nodemanager
```

If etc/hadoop/workers and ssh trusted access is configured (see [Single Node Setup](#)), all of the YARN processes can be stopped with a utility script. As *yarn*:

```
[yarn]$ $HADOOP_HOME/sbin/stop-yarn.sh
```

Stop the WebAppProxy server. Run on the WebAppProxy server as *yarn*. If multiple servers are used with load balancing it should be run on each of them:

```
[yarn]$ $HADOOP_HOME/bin/yarn stop proxyserver
```

Stop the MapReduce JobHistory Server with the following command, run on the designated server as *mapred*:

```
[mapred]$ $HADOOP_HOME/bin/mapred --daemon stop historyserver
```

INSTALL HADOOP

STEP 1: [CLICK HERE](#) TO DOWNLOAD THE JAVA 8 PACKAGE. SAVE THIS FILE IN YOUR HOME DIRECTORY.

STEP 2: EXTRACT THE JAVA TAR FILE.

Command: tar -xvf jdk-8u101-linux-i586.tar.gz

STEP 3: DOWNLOAD THE HADOOP 2.7.3 PACKAGE.

Command: wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.3/hadoop-2.7.3.tar.gz

STEP 4: EXTRACT THE HADOOP TAR FILE.

Command: tar -xvf hadoop-2.7.3.tar.gz

STEP 5: ADD THE HADOOP AND JAVA PATHS IN THE BASH FILE (.BASHRC).

Open. **bashrc** file. Now, add Hadoop and Java Path as shown below.

Command: vi .bashrc

Then, save the bash file and close it.

For applying all these changes to the current Terminal, execute the source command.

Command: source .bashrc

To make sure that Java and Hadoop have been properly installed on your system and can be accessed through the Terminal, execute the java -version and hadoop version commands.

Command: java -version

Command: hadoop version

STEP 6: EDIT THE [HADOOP CONFIGURATION FILES](#) .

Command: cd hadoop-2.7.3/etc/hadoop/

Command: ls

All the Hadoop configuration files are located in **hadoop-2.7.3/etc/hadoop** directory as you can see in the snapshot below:

STEP 7: OPEN CORE-SITE.XML AND EDIT THE PROPERTY MENTIONED BELOW INSIDE CONFIGURATION TAG :

core-site.xml informs Hadoop daemon where NameNode runs in the cluster. It contains configuration settings of Hadoop core such as I/O settings that are common to HDFS & MapReduce.

Command: vi core-site.xml

STEP 8: EDIT HDFS-SITE.XML AND EDIT THE PROPERTY MENTIONED BELOW INSIDE CONFIGURATION TAG :

hdfs-site.xml contains configuration settings of HDFS daemons (i.e. NameNode, DataNode, Secondary NameNode). It also includes the replication factor and block size of HDFS.

Command: vi hdfs-site.xml

STEP 9: EDIT THE MAPRED -SITE .XML FILE AND EDIT THE PROPERTY MENTIONED BELOW INSIDE CONFIGURATION TAG :

mapred-site.xml contains configuration settings of MapReduce application like number of JVM that can run in parallel, the size of the mapper and the reducer process, CPU cores available for a process, etc.

In some cases, mapred-site.xml file is not available. So, we have to create the mapred-site.xml file using mapred-site.xml template.

Command: cp mapred-site.xml.template mapred-site.xml

Command: vi mapred-site.xml.

STEP 10: EDIT YARN -SITE .XML AND EDIT THE PROPERTY MENTIONED BELOW INSIDE CONFIGURATION TAG :

yarn-site.xml contains configuration settings of ResourceManager and NodeManager like application memory management size, the operation needed on program & algorithm, etc.

Command: vi yarn-site.xml

STEP 11: EDIT HADOOP -ENV .SH AND ADD THE JAVA PATH AS MENTIONED BELOW :

hadoop-env.sh contains the environment variables that are used in the script to run Hadoop like Java home path, etc.

Command: vi hadoop-env.sh

STEP 12: GO TO HADOOP HOME DIRECTORY AND FORMAT THE NAMENode.

Command: cd

Command: cd hadoop-2.7.3

Command: bin/hadoop namenode -format

This formats the HDFS via NameNode. This command is only executed for the first time. Formatting the file system means initializing the directory specified by the dfs.name.dir variable.

Never format, up and running Hadoop filesystem. You will lose all your data stored in the HDFS.

STEP 13: ONCE THE NAMENode IS FORMATTED , GO TO HADOOP -2.7.3/SBIN DIRECTORY AND START ALL THE DAEMONS .

Command: cd hadoop-2.7.3/sbin

Either you can start all daemons with a single command or do it individually.

Command: ./start-all.sh

The above command is a combination of *start-dfs.sh*, *start-yarn.sh* & *mr-jobhistory-daemon.sh*

Or you can run all the services individually as below:

START NAMENode:

The NameNode is the centerpiece of an HDFS file system. It keeps the directory tree of all files stored in the HDFS and tracks all the file stored across the cluster.

Command: ./hadoop-daemon.sh start namenode

START DATANode:

On startup, a DataNode connects to the Namenode and it responds to the requests from the Namenode for different operations.

Command: ./hadoop-daemon.sh start datanode

START RESOURCE MANAGER :

ResourceManager is the master that arbitrates all the available cluster resources and thus helps in managing the distributed applications running on the YARN system. Its work is to manage each NodeManagers and the each application's ApplicationMaster.

Command: ./yarn-daemon.sh start resourcemanager

START NODEMANAGER :

The NodeManager in each machine framework is the agent which is responsible for managing containers, monitoring their resource usage and reporting the same to the ResourceManager.

Command: ./yarn-daemon.sh start nodemanager

START JOB HISTORY SERVER :

JobHistoryServer is responsible for servicing all job history related requests from client.

Command: ./mr-jobhistory-daemon.sh start historyserver

STEP 14: TO CHECK THAT ALL THE HADOOP SERVICES ARE UP AND RUNNING , RUN THE BELOW COMMAND .

Command: jps

STEP 15: NOW OPEN THE MOZILLA BROWSER AND GO TO LOCALHOST :50070/DFSHEALTH .HTML TO CHECK THE NAME NODE INTERFACE .

CONSIDERATIONS FOR CLOUD APPLICATION DESIGN

DESIGN FOR THE CLOUD

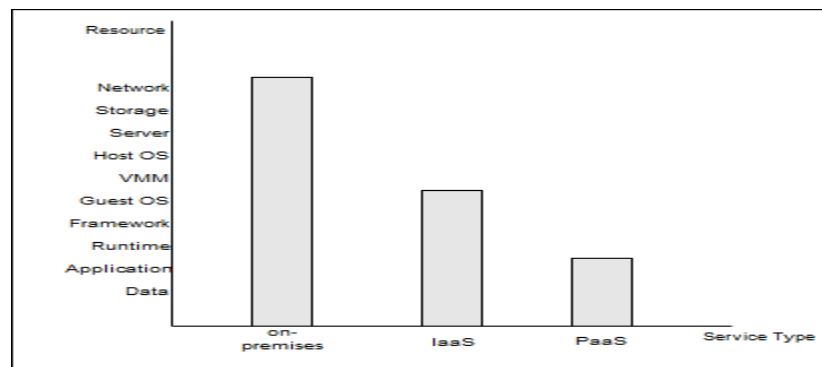
Jane is a developer, and she is working on a software project for a client who is planning to host the project with one of the cloud computing service providers. So, Jane needs advice on the considerations for cloud applications. Her friend Teresa, a specialist in the cloud computing environment, is trying to help her.

CLOUD APPLICATIONS, NOT APPLICATIONS IN THE CLOUD

Jane asks, "Are you saying that I should consider certain points when I design for the cloud environment?"

"Yes, you are right," Teresa responds. "Developing for the cloud is different from developing for the on-premises environment. The cloud environment has some constraints but offers additional features to enable autoscaling, securing, enhancing availability, and elevating performance. If you want to benefit from these features, you should plan the application design for the cloud." "Please tell me more about that," Jane responds.

"Let's start from a simple but essential matter, that is, access to the machine resources in the application. You should be considering the type of service you are renting from the provider when designing your application. Take a look at Figure-1 which shows a comparison of the **control level** over resources for local, Infrastructure as a Service (IaaS), and Platform as a Service (PaaS) environments. You can see that if you go for a PaaS environment, you will not be able to access the file system of the machine your application is running on! So, if the client is planning for PaaS, then you should expect to use a storage service for the files your application generates or uses."



When designing applications for the cloud, irrespective of the chosen platform, I have often found it useful to consider four specific topics during my initial discussions; scalability, availability, manageability and feasibility.

It is important to remember that the items presented under each topic within this article are not an exhaustive list and are aimed only at presenting a starting point for a series of long and detailed conversations with the stakeholders of your project, always the most important part of the design of any application. The aim of these conversations should be to produce an initial high-level design and architecture. This is achieved by considering these four key elements holistically within the domain of the customers project requirements, always remembering to consider the side-effects and trade-offs of any design decision (i.e. what we gain vs. what we lose, or what we make more difficult).

SCALABILITY

Conversations about scalability should focus on any requirement to add additional capacity to the application and related services to handle increases in load and demand. It is particularly important to consider each application tier when designing for scalability, how they should scale individually and how we can avoid contention issues and bottlenecks.

Key areas to consider include:

CAPACITY

- Will we need to scale individual application layers and, if so, how can we achieve this without affecting availability?
- How quickly will we need to scale individual services?
- How do we add additional capacity to the application or any part of it?
- Will the application need to run at scale 24x7, or can we scale-down outside business hours or at weekends for example?

PLATFORM / DATA

- Can we work within the constraints of our chosen persistence services while working at scale (database size, transaction throughput, etc.)?
- How can we partition our data to aid scalability within persistence platform constraints (e.g. maximum database sizes, concurrent request limits, etc.)?
- How can we ensure we are making efficient and effective use of platform resources? As a rule of thumb, I generally tend towards a design based on many small instances, rather than fewer large ones.
- Can we collapse tiers to minimise internal network traffic and use of resources, whilst maintaining efficient scalability and future code maintainability?

LOAD

- How can we improve the design to avoid contention issues and bottlenecks? For example, can we use queues or a service bus between services in a co-operating producer, competing consumer pattern?
- Which operations could be handled asynchronously to help balance load at peak times?
- How could we use the platform features for rate-leveling (e.g. Azure Queues, Service Bus, etc.)?
- How could we use the platform features for load-balancing (e.g. Azure Traffic Manager, Load Balancer, etc.)?

AVAILABILITY

Availability describes the ability of the solution to operate in a manner useful to the consumer in spite of transient and enduring faults in the application and underlying operating system, network and hardware dependencies. In reality, there is often some crossover between items useful for availability and scalability. Conversations should cover at least the following items:

UPTIME GUARANTEES

- What Service Level Agreements (SLA's) are the products required to meet?
- Can these SLA's be met? Do the different cloud services we are planning to use all conform to the levels required? Remember that [SLA's are composite](#).

REPLICATION AND FAILOVER

- Which parts of the application are most at risk from failure?
- In which parts of the system would a failure have the most impact?
- Which parts of the application could benefit from redundancy and failover options?
- Will data replication services be required?
- Are we restricted to specific geopolitical areas? If so, are all the services we are planning to use available in those areas?
- How do we prevent corrupt data from being replicated?
- Will recovery from a failure put excess pressure on the system? Do we need to implement retry policies and/or a circuit-breaker?

DISASTER RECOVERY

- In the event of a catastrophic failure, how do we rebuild the system?
- How much data, if any, is it acceptable to lose in a disaster recovery scenario?
- How are we handling backups? Do we have a need for backups in addition to data-replication?
- How do we handle "in-flight" messages and queues in the event of a failure?
- Are we idempotent? Can we replay messages?
- Where are we storing our VM images? Do we have a backup?

PERFORMANCE

- What are the acceptable levels of performance? How can we measure that? What happens if we drop below this level?
- Can we make any parts of the system asynchronous as an aid to performance?
- Which parts of the system are the mostly highly contended, and therefore more likely to cause performance issues?
- Are we likely to hit traffic spikes which may cause performance issues? Can we auto-scale or use queue-centric design to cover for this?

SECURITY

This is clearly a huge topic in itself, but a few interesting items to explore which relate directly to cloud-computing include:

- What is the local law and jurisdiction where data is held? Remember to include the countries where failover and metrics data are held too.
- Is there a requirement for federated security (e.g. ADFS with Azure Active Directory)?
- Is this to be a hybrid-cloud application? How are we securing the link between our corporate and cloud networks?
- How do we control access to the administration portal of the cloud provider?
- How do we restrict access to databases, etc. from other services (e.g. IP Address white-lists, etc.)?
- How do we handle regular password changes?
- How does service-decoupling and multi-tenancy affect security?
- How we will deal with operating system and vendor security patches and updates?

MANAGEABILITY

This topic of conversation covers our ability to understand the health and performance of the live system and manage site operations. Some useful cloud specific considerations include:

MONITORING

- How are we planning to monitor the application?
- Are we going to use off-the-shelf monitoring services or write our own?
- Where will the monitoring/metrics data be physically stored? Is this in line with data protection policies?
- How much data will our plans for monitoring produce?
- How will we access metrics data and logs? Do we have a plan to make this data useable as volumes increase?
- Is there a requirement for auditing as well as logging?
- Can we afford to lose some metrics/logging/audit data (i.e. can we use an asynchronous design to “fire and forget” to help aid performance)?
- Will we need to alter the level of monitoring at runtime?
- Do we need automated exception reporting?

DEPLOYMENT

- How do we automate the deployment?
- How do we patch and/or redeploy without disrupting the live system? Can we still meet the SLA's?
- How do we check that a deployment was successful?
- How do we roll-back an unsuccessful deployment?
- How many environments will we need (e.g. development, test, staging, production) and how will deploy to each of them?
- Will each environment need separate data storage?
- Will each environment need to be available 24x7?

FEASIBILITY

When discussing feasibility we consider the ability to deliver and maintain the system, within budgetary and time constraints. Items worth investigating include:

- Can the SLA's ever be met (i.e. is there a cloud service provider that can give the uptime guarantees that we need to provide to our customer)?
- Do we have the necessary skills and experience in-house to design and build cloud applications?
- Can we build the application to the design we have within budgetary constraints and a timeframe that makes sense to the business?
- How much will we need to spend on operational costs (cloud providers often have very complex pricing structures)?
- What can we sensibly reduce (scope, SLAs, resilience)?
- What trade-offs are we willing to accept?

Application design consideration

- Persistence.
- Model-view-controller pattern.
- Statelessness.
- Caching.
- Asynchronous considerations.
- Third-party libraries

The most important criteria for a cloud application architecture design?

The system should be architected in such a way that **continuous deployment is possible from testing to staging and then staging to production**. The applications should be designed in a modular way so that continuous deployment is possible.

PLANNING THE APPLICATION FOR THE CLOUD

Jane states, "I see your point, so how do I plan my application for the cloud?"

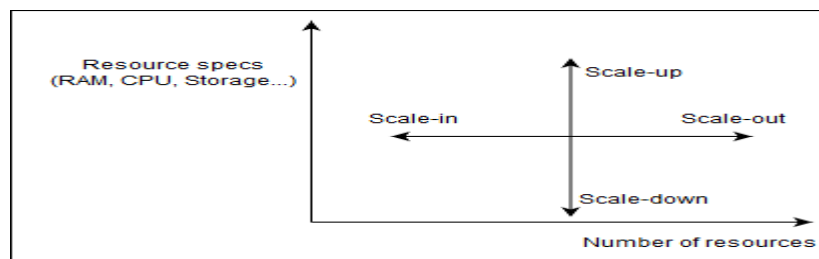
Teresa explains, "There are some concepts that help developers design for the cloud.

1. The composition of the cloud and more specifically virtualization and elasticity.
2. Loose coupling of components (as in the case of web services).
3. Fault tolerance and high availability.
4. Multitenancy.
5. Platform-agnosticism.
6. Performance enhancement."

THE CLOUD COMPOSITION

"I know the cloud is based on virtualization technology, but do I have to understand how the technology works in detail?" Jane asks.

Teresa responds, "No, you don't have to. However, it is necessary to have a basic idea of the virtualization technology and rapid elasticity since they enable auto-scaling and dynamically networking resources as the four directions in Figure-2 demonstrate. Scaling out/in is about increasing/decreasing the number of resources while scaling up/down is about enhancing/degrading the specs of a resource."



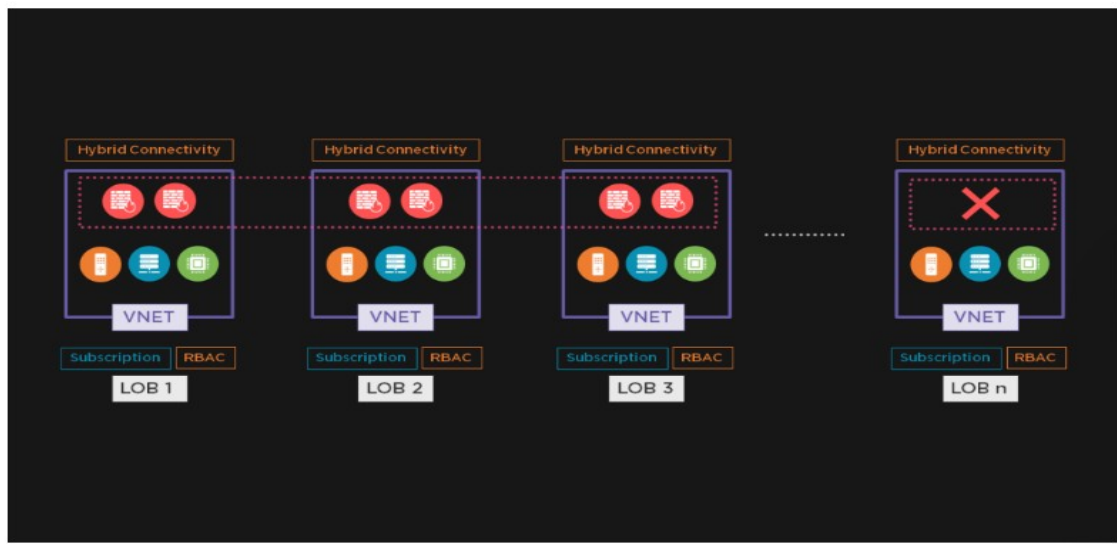
CLOUD REFERENCE ARCHITECTURE (CRA)

Before digging into the definition of the Cloud Reference Architecture (CRA) and its benefits, it is better to look at how things can go wrong without having one. You will quickly realize that it is better to spend some time before migration to plan your cloud migration journey with security and governance in mind. Doing that will not only save you time and money but will help you meet your security and governance needs. So let's get started.

When organizations start planning their cloud migration, and like anything else new, they start by trying and testing some capabilities. Perhaps they start hosting their development environment in the cloud while keeping their production one on-premises.

It is also common to see small and isolated applications being migrated first, perhaps because of their size, low criticality and to give the cloud a chance to prove it is trust worthy. After all, migration to the cloud is a journey and doesn't happen overnight.

Then the benefits of cloud solutions became apparent and companies started to migrate multiple large-scale workloads. As more and more workloads move to the cloud, many organizations find themselves dealing with workload islands that are managed separately with different security models and independent data flows.



Even worse, with the pressure to quickly get new applications deployed in the cloud with strict deadlines, developers find themselves rushing to consume new cloud services without reasonable consideration to organization's security and governance needs.

The unfortunate result in most cases is to end up with a cloud infrastructure that is hard to manage and maintain. Each application could end up deployed in a separate island with its own connectivity infrastructure and with poor access management.



Managing cost of running workloads in the cloud becomes also challenge. There is no clear governance and accountability model which leads to a lot of management overhead and security concerns.

The lack of governance, automation, naming convention and security models are also even hard to achieve afterwards. In fact, it is nightmare to look at a poorly managed cloud infrastructure and then trying to apply security and governance afterward because these need to be planned a head before even deploying any cloud resources.

Even worse, data can be hosted in geographies that violates corporate's compliance requirements, which is a big concern for most organizations. I remember once asking one of my customers if they knew where their cloud data is hosted, and most of them just don't know.

THE BENEFITS OF CLOUD REFERENCE ARCHITECTURE (CRA)

Simply put, the Cloud Reference Architecture (CRA) **helps organizations address the need for detailed, modular and current architecture guidance for building solutions in the cloud.**

The Cloud Reference Architecture (CRA) serves as a collection of *design guidance* and *design patterns* to support structured approach to deploy services and applications in the cloud. This means that every workload is deployed with security, governance and compliance in mind from day one.

CRA

Cloud reference architecture (CRA) helps organizations address the need for detailed, modular and current architecture guidance for building solutions in the cloud

The ISO/IEC 17789 Cloud Computing Reference Architecture defines four different views for the Cloud Reference Architecture (CRA) :

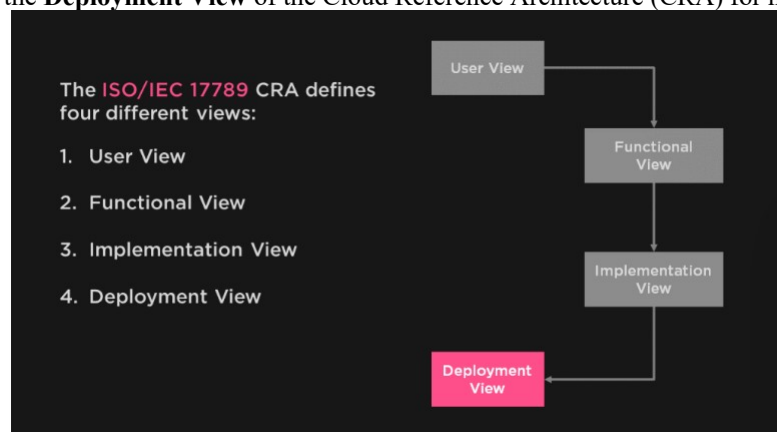
User View

Functional View

Implementation View

Deployment View.

We will be focusing on the **Deployment View** of the Cloud Reference Architecture (CRA) for now.



The Cloud Reference Architecture (CRA) **Deployment View** provides a framework to be used for all cloud deployment projects, which reduces the effort during design and provides an upfront guidance for a deployment aligned to architecture, security and compliance.

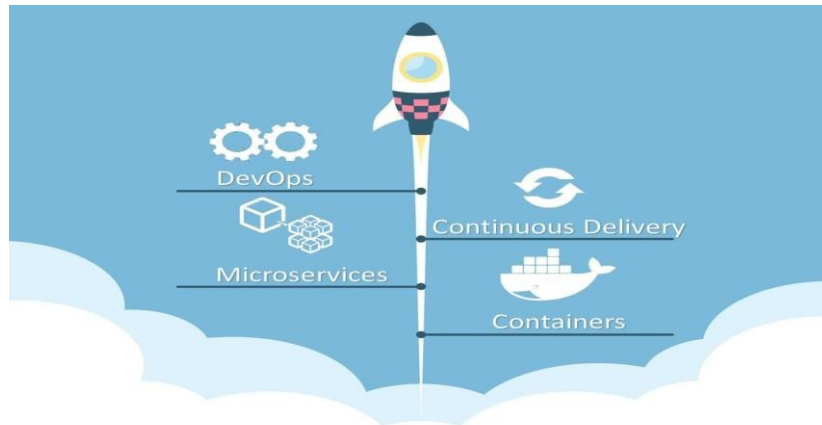
You can think of the Cloud Reference Architecture (CRA) **Deployment View** as the blueprint for all cloud projects.

What you get from this blueprint, the end goal if you are wondering, is to help you quickly develop and implement cloud-based solutions, while reducing complexity and risk.

Therefore, having a **foundation architecture** not only helps you ensure security, manageability and compliance but also consistency for deploying resources. It includes network, security, management infrastructure, naming convention, hybrid connectivity and more.

I know what you might be thinking right now, how does one blueprint fit the need for organizations with different sizes? Since not all organizations are the same, the Cloud Reference Architecture (CRA) **Deployment View** does not outline a single design that fits all sizes. Rather, it provides a framework for decisions based on core cloud services, features and capabilities.

CloudApplication Design Methodologies:



12 Factor Design Methodology and Cloud-Native Applications

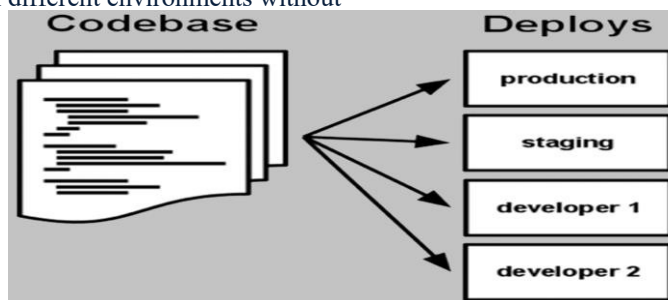
It is a design methodology which has been introduced to manage cloud-based or software as a Service ([SaaS](#)) apps. Some of the key features or applications of this design methodology are as follows:

- Use declarative formats for setup automation, to minimize time and cost for new developers joining the project
- Have a clean contract with the underlying operating system, offering maximum portability between execution environments
- Are suitable for deployment on [modern cloud platforms](#) ([Google cloud](#), Heroku, [AWS](#) etc..), obviating the need for servers and systems administration
- Minimize divergence between development and production, enabling continuous deployment for maximum

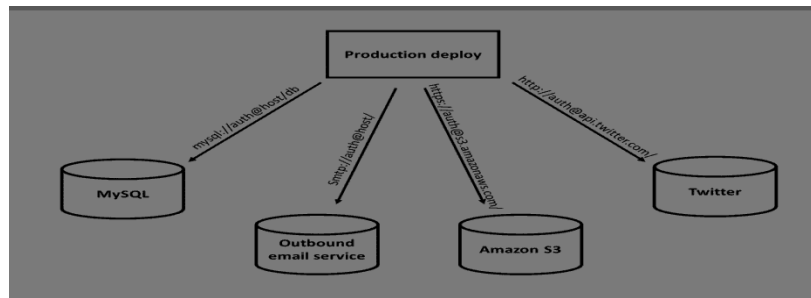
If we simplify this term further, 12 Factor App design methodology is nothing but a collection of 12 factors which act as building blocks for deploying or developing an app in the cloud.

Listed below are the 12 Factors:

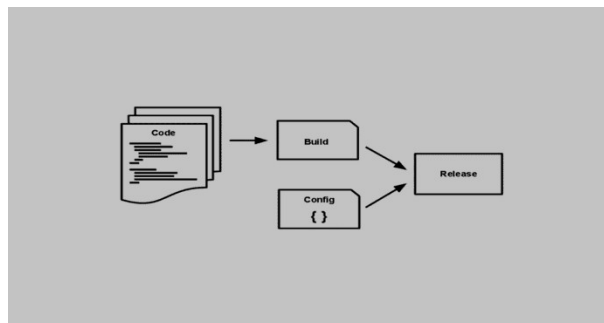
1. **Codebase:** A 12 Factor App is always tracked in a version control system such as Git or Apache Subversion (SVN) in the form of code repository. This will essentially help you to build your code on top of one codebase, fully backed up with many deployments and revision control. As there is a one to one relationship between a 12 factor app and the codebase repository, so in case of multiple repositories, there is always a need to consider this relationship as a distributed system consisting of multiple 12 factored apps. Deployments should be automatic, so everything can run in different environments without



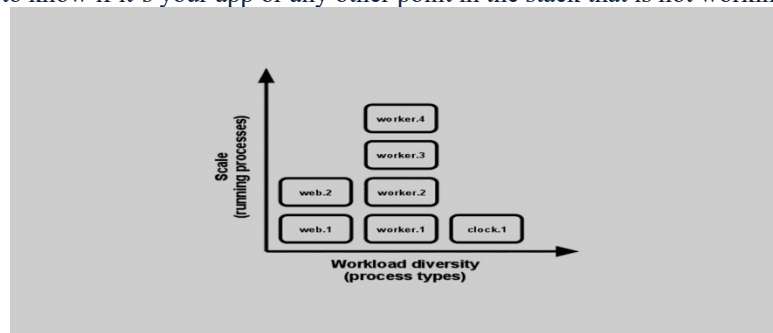
2. **Dependencies:** As the app is standalone and needs to install dependencies, it is important to explicitly declare and isolate dependencies. Moreover, it is always recommended to keep your development, production and QA identical to 12 Factor apps. This will help you to build applications in order to scale web and other such applications that do not have any room for error. As a solution to this, you can use a dependency isolation tool such as ex VirtualEnv for Python uniformly to remove several explicit dependency specifications to both production and development phases and environments.
3. **Config:** This factor manages the configuration information for the app. Here you store your configuration files in the environment. This factor focuses on how you store your data – the database Uniform Resource Identifier (URI) will be different in development, QA and production.



4. **Backing Services:** This includes backing service management services (local database service or any third party service) which depends on over a network connection. In case of a 12 factor app, the interface to connect these services should be defined in a standard way. You need to treat backing services like attached resources because you may want different databases depending on which team you are working with. Sometimes developers will want a lot of logs, while QA will want less. With this method, even each developer can have their own config file



5. **Build, Run, Release:** It is important to run separately all the build and run stages making sure everything has the right libraries. For this, you can make use of required automation and tools to generate build and release packages with proper tags. This is further backed up by running the app in the execution environment while using proper release management tools like Capistrano for ensuring timely rollback.
6. **Stateless Processes:** This factor is about making sure the app is executed in the execution environment as one or more processes. In other words, you want to make sure that all your data is stored in a backing store, which gives you the right to scale out anything and do what you need to do. During stateless processes, you do not want to have a state that you need to pass along as you scale up and out.
7. **Port Binding:** Twelve factor apps are self-contained and do not rely on runtime injection of a web server into the execution environment to create a web-facing service. With the help of port binding, you can directly access your app via a port to know if it's your app or any other point in the stack that is not working properly.



8. **Concurrency:** This factor looks into the best practices for scaling the app. These practices are used to manage each process in the app independently i.e. start/stop, clone to different machines etc. The factor also deals with breaking your app into much smaller pieces and then look for services out there that you either have to write or can consume.
9. **Disposability:** Your app might have different multiple processes handling different tasks. So, the ninth factor looks into the robustness of the app with fast startup and shutdown methods. Disposability is about making sure your app can startup and takes down fast and can handle any crash anytime. You can use some high quality robust queuing backend (Beanstalk, RabbitMQ etc.) that would help return unfinished jobs back to the queue in the case of a failure.
10. **Dev/Prod Parity:** Development, staging and production should be as similar as possible. In case of continuous deployment, you need to have continuous integration based on matching environments to limit deviation and errors

- . Some of the features of keeping the gap between development and production small are as follows:
1. **Make the time gap small:** a developer may write code and have it deployed hours or even just minutes later.
 2. **Make the personnel gap small:** developers who wrote code are closely involved in deploying it and watching its behavior in production.
 3. **Make the tools gap small:** keep development and production as similar as possible.
11. **Logs:** Logging mechanisms are critical for debugging. Having proper logging mechanisms allows you to output the log info as a continuous stream rather than managing the entire database of log files. Then, depending on the configuration, you can decide where that log will publish.
12. **Admin Processes:** One-off admin processes help in collecting data from the running application. In order to avoid any synchronization issues, you need to ensure that all these processes are a part of all deploys.

Data Storage Approaches:

DATA STORAGE DEVICES

Storage devices can be broadly classified into two categories:

- Block Storage Devices
- File Storage Devices
- **object storage**

BLOCK STORAGE DEVICES

The **block storage devices** offer raw storage to the clients. These raw storage are partitioned to create volumes.

FILE STORAGE DEVICES

The **file Storage Devices** offer storage to clients in the form of files, maintaining its own file system. This storage is in the form of Network Attached Storage (NAS).

CLOUD STORAGE CLASSES

Cloud storage can be broadly classified into two categories:

- Unmanaged Cloud Storage
- Managed Cloud Storage

UNMANAGED CLOUD STORAGE

Unmanaged cloud storage means the storage is preconfigured for the customer. The customer can neither format, nor install his own file system or change drive properties.

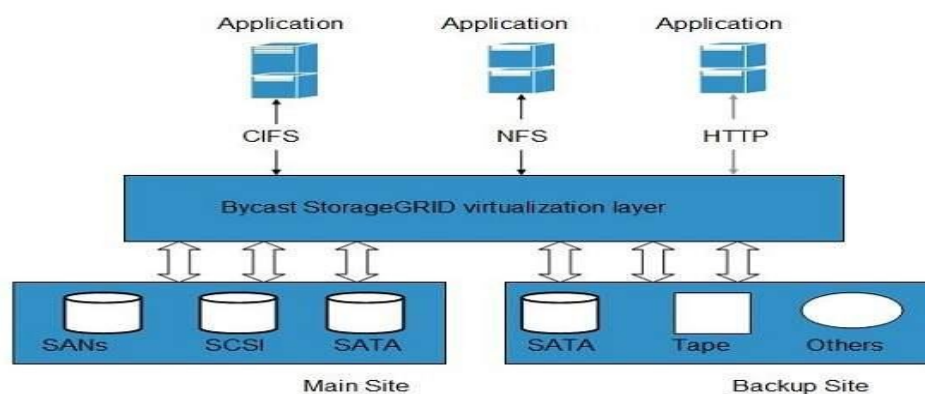
MANAGED CLOUD STORAGE

Managed cloud storage offers online storage space on-demand. The managed cloud storage system appears to the user to be a raw disk that the user can partition and format.

CREATING CLOUD STORAGE SYSTEM

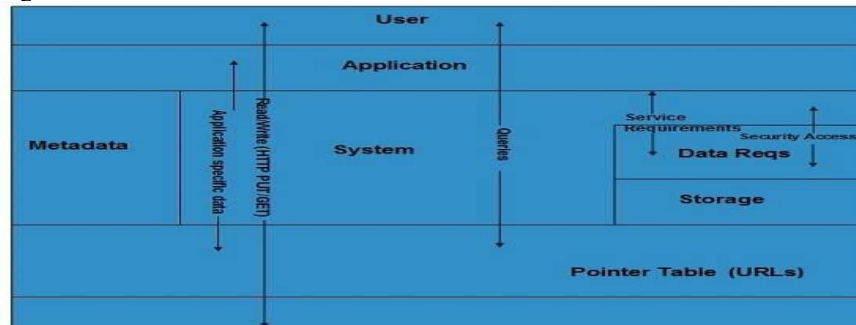
The cloud storage system stores multiple copies of data on multiple servers, at multiple locations. If one system fails, then it is required only to change the pointer to the location, where the object is stored.

To aggregate the storage assets into cloud storage systems, the cloud provider can use storage virtualization software known as **StorageGRID**. It creates a virtualization layer that fetches storage from different storage devices into a single management system. It can also manage data from **CIFS** and **NFS** file systems over the Internet. The following diagram shows how StorageGRID virtualizes the storage into storage clouds:



VIRTUAL STORAGE CONTAINERS

The **virtual storage containers** offer high performance cloud storage systems. **Logical Unit Number (LUN)** of device, files and other objects are created in virtual storage containers. Following diagram shows a virtual storage container, defining a cloud storage domain:



CHALLENGES

Storing the data in cloud is not that simple task. Apart from its flexibility and convenience, it also has several challenges faced by the customers. The customers must be able to:

- Get provision for additional storage on-demand.
- Know and restrict the physical location of the stored data.
- Verify how data was erased.
- Have access to a documented process for disposing of data storage hardware.
- Have administrator access control over data.

Storage Systems in the Cloud :

There are 3 types of storage systems in the Cloud as follows.

- Block-Based Storage System
- File-Based Storage System
- Object-Based Storage System

Type-1 :

Block-Based Storage System –

Hard drives are block-based storage systems. Your operating system like Windows or Linux actually sees a hard disk drive. So, it sees a drive on which you can create a volume, and then you can partition that volume and format them.

For example, If a system has 1000 GB of volume, then we can partition it into 800 GB and 200 GB for local C and local D drive respectively.

Remember with a block-based storage system, your computer would see a drive, and then you can create volumes and partitions

Type-2 :

File-Based Storage System –

In this, you are actually connecting through a [Network Interface Card \(NIC\)](#). You are going over a network, and then you can access the network-attached storage server (NAS). NAS devices are file-based storage systems.

This storage server is another computing device that has another disk in it. It is already created a file system so that it's already formatted its partitions, and it will share its file systems over the network. Here, you can actually map the drive to its network location.

In this, like the previous one, there is no need to partition and format the volume by the user. It's already done

Type-3 :

Object-Based Storage System –

In this, a user uploads objects using a web browser and uploading an object to a container i.e, Object Storage Container. This uses the HTTP Protocols with the rest of the APIs (example: GET, PUT, POST, SELECT, DELETE).

For example, when you connect to any website, and you need to download some images, text, or anything that the website contains. For that, it is a code HTTP GET request. If you want to review any product then you can use PUT and POST requests.

Also, there is no hierarchy of objects in the container. Every file is on the same level in an Object-Based storage system.

Advantages:**Scalability –**

Capacity and storage can be expanded and performance can be enhanced.

Flexibility –

Data can be manipulated and scaled according to the rules.

Simpler Data Migrations –**Disadvantages:**

Data centers require electricity and proper internet facility to operate their work, failing in which system will not work properly.

Python

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991. It is used for:

web development (server-side),
software development,
mathematics,
system scripting.

Python can do

Python can be used on a server to create web applications.
Python can be used alongside software to create workflows.
Python can connect to database systems. It can also read and modify files.
Python can be used to handle big data and perform complex mathematics.
Python can be used for rapid prototyping, or for production-ready software development.

INSTALL PYTHON IN WINDOWS

Python is a widely used high-level programming language. To write and execute code in python, we first need to install Python on our system.

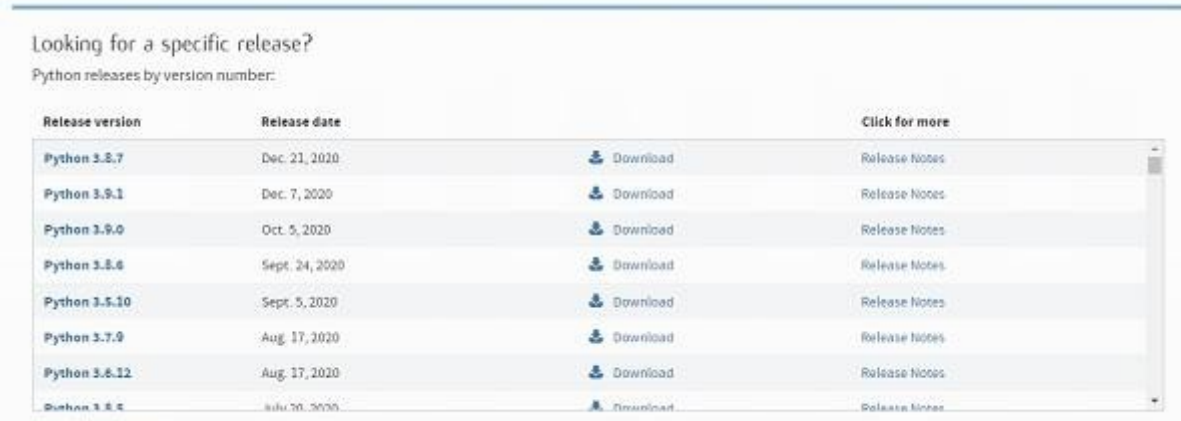
Installing Python on Windows takes a series of few easy steps.

STEP 1 – SELECT VERSION OF PYTHON TO INSTALL

Python has various versions available with differences between the syntax and working of different versions of the language. We need to choose the version which we want to use or need. There are different versions of Python 2 and Python 3 available.

STEP 2 – DOWNLOAD PYTHON EXECUTABLE INSTALLER

On the web browser, in the official site of python (www.python.org), move to the Download for Windows section. All the available versions of Python will be listed. Select the version required by you and click on Download. Let suppose, we chose the Python 3.9.1 version.



Looking for a specific release?
Python releases by version number:

Release version	Release date		Click for more
Python 3.8.7	Dec. 21, 2020	Download	Release Notes
Python 3.9.1	Dec. 7, 2020	Download	Release Notes
Python 3.9.0	Oct. 5, 2020	Download	Release Notes
Python 3.8.6	Sept. 24, 2020	Download	Release Notes
Python 3.8.10	Sept. 5, 2020	Download	Release Notes
Python 3.7.9	Aug. 17, 2020	Download	Release Notes
Python 3.8.12	Aug. 17, 2020	Download	Release Notes
Python 3.8.2	July 20, 2020	Download	Release Notes

[View older releases](#)

On clicking download, various available executable installers shall be visible with different operating system specifications. Choose the installer which suits your system operating system and download the installer. Let suppose, we select the Windows installer(64 bits). The download size is less than 30MB.

Version	Operating System	Description	MDS Sum	File Size	PGP
Gzipped source tarball	Source release		429ae95d24227f8fa1560684ad6fca7	25372998	SIG
XZ compressed source tarball	Source release		61981498e75ac8f00adcb908281adb6	18897104	SIG
macOS 64-bit intel installer	Mac OS X	for macOS 10.9 and later	74f5cc5b5783ce8fb2ca55f11f3f0699	29795899	SIG
macOS 64-bit universal2 installer	Mac OS X	for macOS 10.9 and later, including macOS 11 Big Sur on Apple Silicon (experimental)	8b19748473609241e60aa3618bbaf3ed	37451735	SIG
Windows embeddable package (32-bit)	Windows		96c6fa81fe8b650e68c3dd41258ae317	7571141	SIG
Windows embeddable package (64-bit)	Windows		e70e5c22432d8f57a497cde5ec2e5ce2	8402333	SIG
Windows help file	Windows		c49d9b6ef88c0831ed0e2d39bc42b316	8787443	SIG
Windows installer (32-bit)	Windows		dde210ea04a31c27488605a9e7cd297a	27126136	SIG
Windows installer (64-bit)	Windows	Recommended	b3fce2ed8bc315ad2bc49eae48a94487	28204528	SIG

STEP 3 – RUN EXECUTABLE INSTALLER

We downloaded the Python 3.9.1 Windows 64 bit installer.

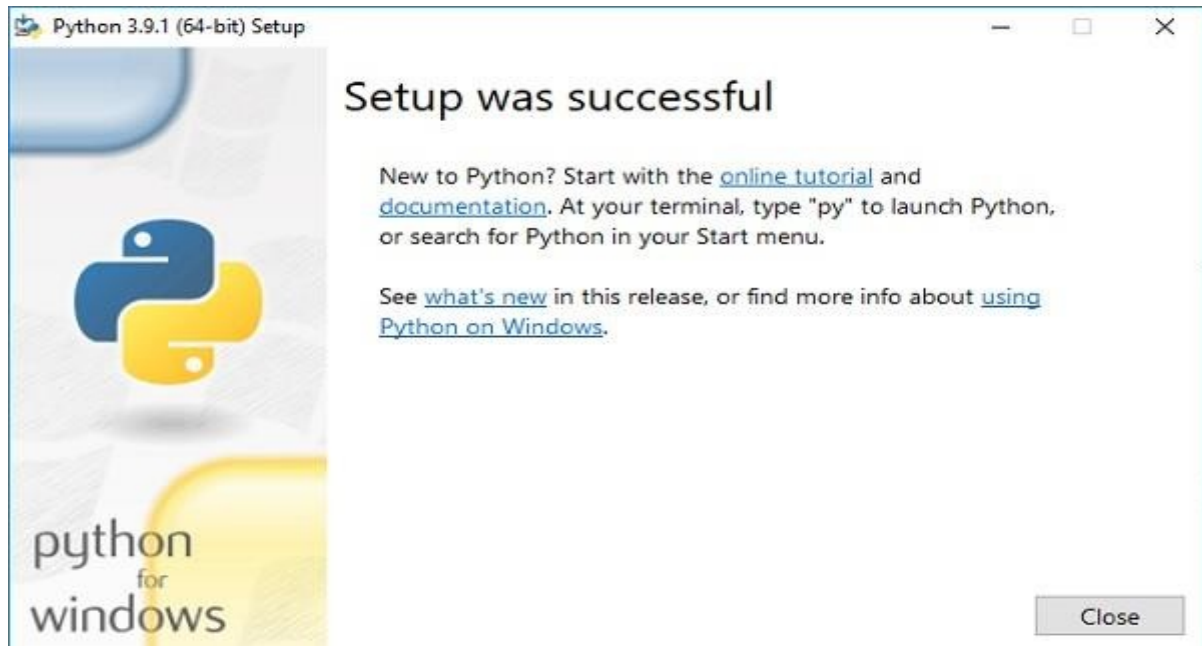
Run the installer. Make sure to select both the checkboxes at the bottom and then click Install New.



On clicking the Install Now, The installation process starts.



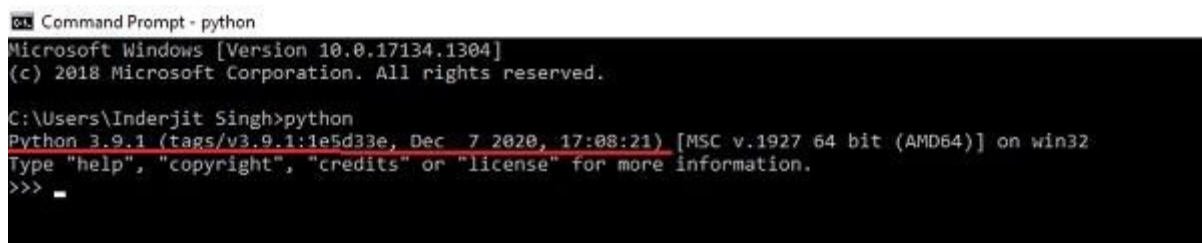
The installation process will take few minutes to complete and once the installation is successful, the following screen is displayed.



STEP 4 – VERIFY PYTHON IS INSTALLED ON WINDOWS

To ensure if Python is successfully installed on your system. Follow the given steps –

- Open the command prompt.
- Type 'python' and press enter.
- The version of the python which you have installed will be displayed if the python is successfully installed on your windows.
-

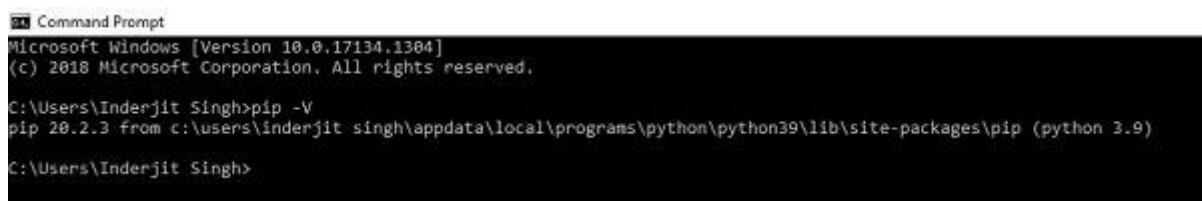


STEP 5 – VERIFY PIP WAS INSTALLED

Pip is a powerful package management system for Python software packages. Thus, make sure that you have it installed.

To verify if pip was installed, follow the given steps –

- Open the command prompt.
- Enter pip -V to check if pip was installed.
- The following output appears if pip is installed successfully.
-



We have successfully installed python and pip on our Windows system.

PYTHON DATA TYPES

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:

Numeric Types:

Sequence Types:

Mapping Type: `dict`

Set Types: `set`, `frozenset`

Boolean Type: `bool`

Binary Types: `bytes`, `bytearray`, `memoryview`

None Type: `NoneType`

SETTING THE DATA TYPE

In Python, the data type is set when you assign a value to a variable:

Example	Data Type
<code>x = "Hello World"</code>	
<code>x = 20</code>	<code>int</code>
<code>x = 20.5</code>	
<code>x = 1j</code>	<code>complex</code>
<code>x = ["apple", "banana", "cherry"]</code>	
<code>x = ("apple", "banana", "cherry")</code>	<code>tuple</code>
<code>x = range(6)</code>	
<code>x = {"name" : "John", "age" : 36}</code>	<code>dict</code>
<code>x = {"apple", "banana", "cherry"}</code>	<code>set</code>


```
x = frozenset({"apple", "banana", "cherry"})
```

```
x = True
```

bool

```
x = b"Hello"
```

bytes

```
x = bytearray(5)
```

bytearray

```
x = memoryview(bytes(5))
```

```
x = None
```

NoneType

SETTING THE SPECIFIC DATA TYPE

If you want to specify the data type, you can use the following constructor functions:

```
x = str("Hello World")
```

str

```
x = int(20)
```

int

```
x = float(20.5)
```

float

```
x = complex(1j)
```

```
x = list(("apple", "banana", "cherry"))
```

```
x = tuple(("apple", "banana", "cherry"))
```

tuple

```
x = range(6)
```

range

```
x = dict(name="John", age=36)
```

dict

```
x = set(("apple", "banana", "cherry"))
```

```
x = frozenset(("apple", "banana", "cherry"))
```

```
x = bool(5)
```

bool

```
x = bytes(5)
```

bytes

```
x = bytearray(5)
```

bytearray

```
x = memoryview(bytes(5))
```

Control Flow in Python

There comes situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code. Decision-making statements in programming languages decide the direction (Control Flow) of the flow of program execution.

Types of Control Flow in Python

In [Python programming language](#), the type of control flow statements is as follows:

1. [The if statement](#)
2. [The if-else statement](#)
3. [The nested-if statement](#)
4. [The if-elif-else ladder](#)

Python if statement

The if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not.

Syntax: if condition:

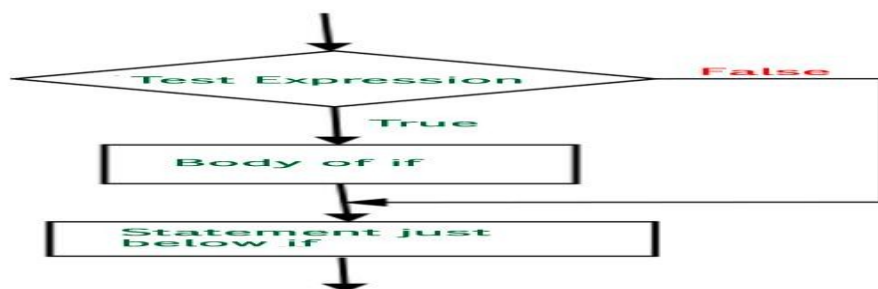
 statement1

statement2

Here if the condition is true, if block

will consider only statement1 to be inside

its block.



Python If-Else Statement

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But if we want to do something else if the condition is false, we can use the *else* statement with *if* statement to execute a block of code when the if condition is false.

Syntax:

if (condition):

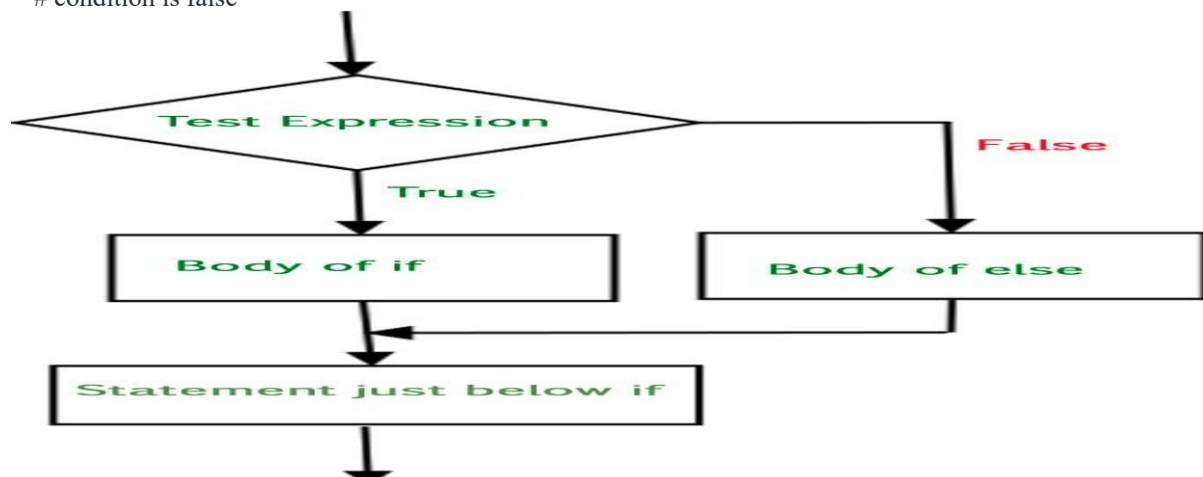
 # Executes this block if

 # condition is true

else:

 # Executes this block if

 # condition is false

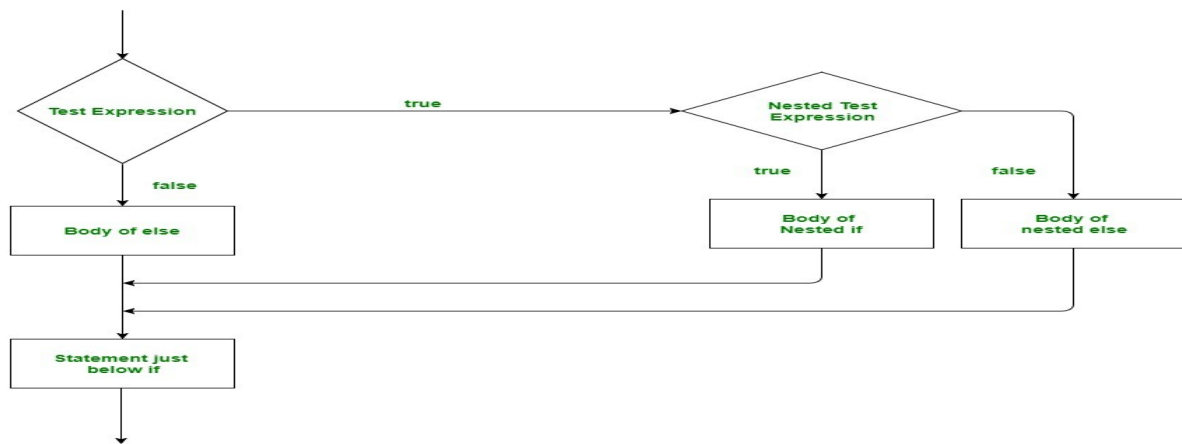


Nested-If Statement in Python

A nested if is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, Python allows us to nest if statements within if statements. i.e., we can place an if statement inside another if statement.

Syntax:

```
if (condition1):  
    # Executes when condition1 is true  
    if (condition2):  
        # Executes when condition2 is true  
    # if Block is end here  
# if Block is end here
```



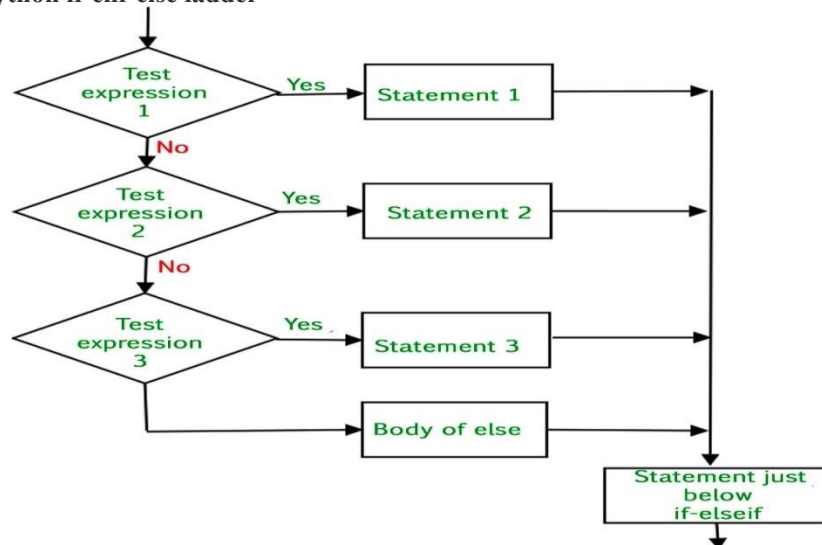
Python if-elif-else Ladder

Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:

```
if (condition):  
    statement  
elif (condition):  
    statement  
.  
.  
.  
else:  
    statement
```

Flowchart of Python if-elif-else ladder



Python Functions

Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.

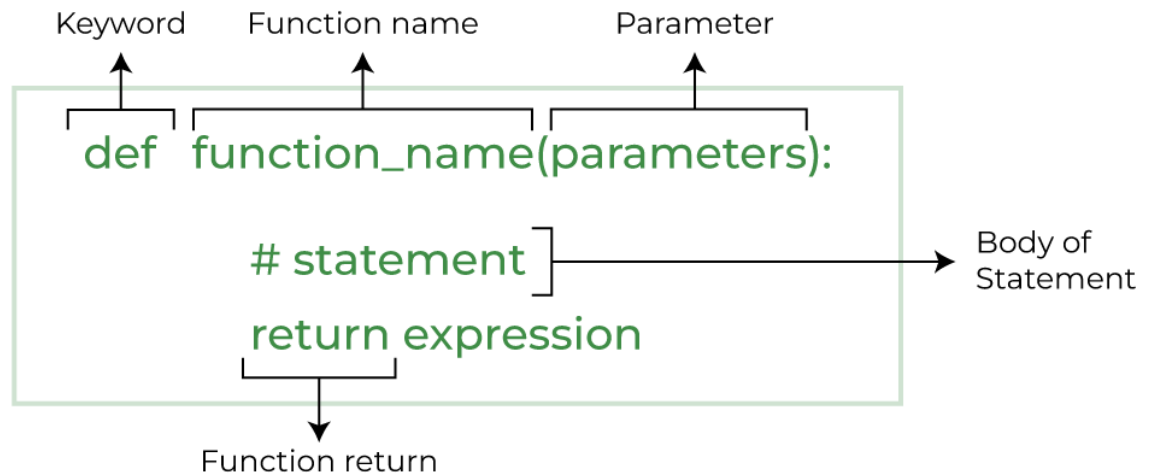
Some Benefits of Using Functions

Increase Code Readability

Increase Code Reusability

Python Function Declaration

The syntax to declare a function is:



Syntax of Python Function Declaration

Types of Functions in Python

There are mainly two types of functions in Python: _____

Built-in library function: These are [Standard functions](#) in Python that are available to use.

User-defined function: We can create our own functions based on our requirements.

Creating a Function in Python

We can create a user-defined function in Python, using the **def** keyword. We can add any type of functionalities and properties to it as we require.

deffun():

print("Welcome to GFG")

Types of Python Function Arguments

Python supports various types of arguments that can be passed at the time of the function call. In Python, we have the following 4 types of function arguments.

Default argument

Keyword arguments (named arguments)

Positional arguments

Arbitrary arguments (variable-length arguments *args and **kwargs)

Python Modules

A [Python](#) module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized.

Create a Python Module

Let's create a simple `calc.py` in which we define two functions, one **add** and another **subtract**.

A simple module, `calc.py`

```
def add(x, y):  
    return(x+y)
```

```
def subtract(x, y):  
    return(x-y)
```

Import module in Python

We can import the functions, and classes defined in a module to another module using the [import statement](#) in some other Python source file.

Syntax of Python Import

```
import module
```

Python Packages

[Python modules](#)

modules. In simpler terms, Package in Python is a folder that contains various modules as files.

Creating Package

Let's create a package in Python named `mypckg` that will contain two modules `mod1` and `mod2`. To create this module follow the below steps:

Create a folder named `mypckg`.

Inside this folder create an empty Python file i.e. `__init__.py`

Then create two modules `mod1` and `mod2` in this folder.

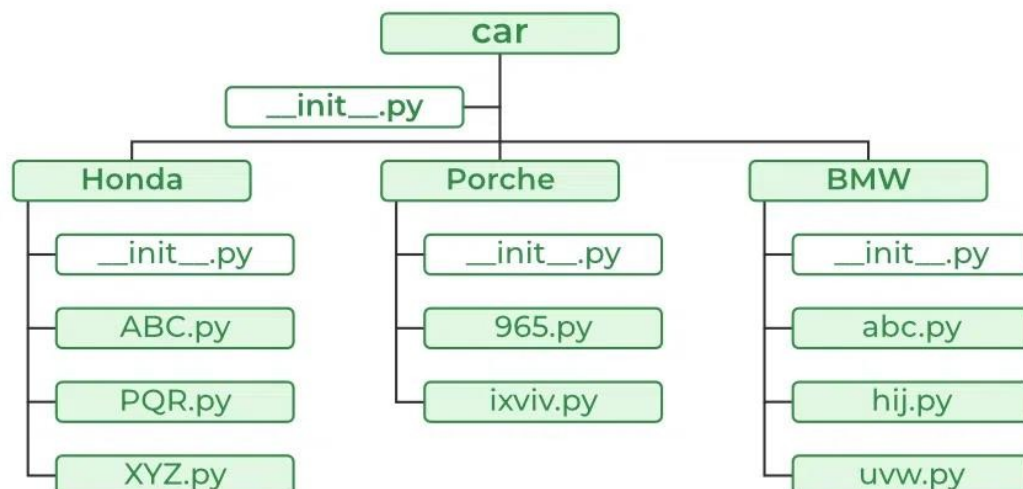
```
def gfg():  
    print("Welcome to GFG")
```

Mod2.py

- Python3

```
def sum(a, b):  
    return a+b
```

The Hierarchy of our Python package looks like this:



Import Modules from a Package

We can import these Python modules using the [from...import statement](#) and the dot(.) operator.

Syntax:

```
import package_name.module_name
```

Amp is an open-source package designed to easily bring machine-learning to atomistic calculations. This project is being developed at Brown University in the School of Engineering, primarily by Andrew Peterson and Alireza Khorshidi, and is released under the GNU General Public License.

The latest stable release of Amp is version 1.0.1, released on January 25, 2023; see the Release notes page for a download link. Please see the project's git repository for the latest development version or a place to report an issue.

You can read about Amp in the below paper; if you find this project useful, we would appreciate if you cite this work:

Khorshidi & Peterson, "Amp: A modular approach to machine learning in atomistic simulations", Computer Physics Communications 207:310-324, 2016.

An amp-users mailing list exists for general discussions about the use and development of Amp. You can subscribe via listserv at:

<https://listserv.brown.edu/?SUBED1=AMP-USERS&A=1>

Amp is now part of the Debian archives! This means it will soon be available via your package manager in linux releases like Ubuntu.

Amp is now installable via pip! This means you should be able to install with just:

```
$ pip3 install amp-atomistics
```

File Handling in Python

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but like other concepts of Python, this concept here is also easy and short. [Python](#) treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun. Let's start with the reading and writing files.

Advantages of File Handling

Versatility: File handling in Python allows you to perform a wide range of operations, such as creating, reading, writing, appending, renaming, and deleting files.

Flexibility: File handling in Python is highly flexible, as it allows you to work with different file types (e.g. text files, binary files, CSV files, etc.), and to perform different operations on files (e.g. read, write, append, etc.).

User-friendly: Python provides a user-friendly interface for file handling, making it easy to create, read, and manipulate files.

Cross-platform: Python file-handling functions work across different platforms (e.g. Windows, Mac, Linux), allowing for seamless integration and compatibility.

Disadvantages of File Handling

Error-prone: File handling operations in Python can be prone to errors, especially if the code is not carefully written or if there are issues with the file system (e.g. file permissions, file locks, etc.).

Security risks: File handling in Python can also pose security risks, especially if the program accepts user input that can be used to access or modify sensitive files on the system.

Complexity: File handling in Python can be complex, especially when working with more advanced file formats or operations. Careful attention must be paid to the code to ensure that files are handled properly and securely.

Performance: File handling operations in Python can be slower than other programming languages, especially when dealing with large files or performing complex operations.

Working of open() Function in Python

Before performing any operation on the file like reading or writing, first, we have to open that file. For this, we should use Python's inbuilt function [open\(\)](#) but at the time of opening, we have to specify the mode, which represents the purpose of the opening file.

```
f = open(filename, mode)
```

Where the following mode is supported:

1. **r:** open an existing file for a read operation.
2. **w:** open an existing file for a write operation. If the file already contains some data then it will be overridden but if the file is not present then it creates the file as well.
3. **a:** open an existing file for append operation. It won't override existing data.
4. **r+:** To read and write data into the file. The previous data in the file will be overridden.
5. **w+:** To write and read data. It will override existing data.
6. **a+:** To append and read data from the file. It won't override existing data.

Python datetime module

Python Datetime module supplies classes to work with date and time. These classes provide a number of functions to deal with dates, times, and time intervals. Date and DateTime are an object in Python, so when you manipulate them, you are actually manipulating objects and not strings or timestamps.

The DateTime module is categorized into 6 main classes –

[**date**](#) – An idealized naive date, assuming the current Gregorian calendar always was, and always will be, in effect. Its attributes are year, month, and day.

[**time**](#) – An idealized time, independent of any particular day, assuming that every day has exactly 24*60*60 seconds. Its attributes are hour, minute, second, microsecond, and tzinfo.

[**datetime**](#) – Its a combination of date and time along with the attributes year, month, day, hour, minute, second, microsecond, and tzinfo.

[**timedelta**](#) – A duration expressing the difference between two date, time, or datetime instances to microsecond resolution.

[**tzinfo**](#) – It provides time zone information objects.

[**timezone**](#) – A class that implements the tzinfo abstract base class as a fixed offset from the UTC (New in

Python Date Class

The [**date class**](#) is used to instantiate date objects in Python. When an object of this class is instantiated, it represents a date in the format YYYY-MM-DD. The constructor of this class needs three mandatory arguments year, month, and date.

```
class datetime.date(year, month, day)
```

The arguments must be in the following range –

```
MINYEAR <= year <= MAXYEAR
```

```
1 <= month <= 12
```

```
1 <= day <= number of days in the given month and year
```

Python Classes

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Example

Create a class named MyClass, with a property named x:

```
class MyClass:
```

```
    x = 5
```

```
class Person:
```

```
    def __init__(self, name, age):
```

```
        self.name = name
```

```
        self.age = age
```

```
p1 = Person("John", 36)
```

```
print(p1.name)
```

```
print(p1.age)
```