**GCP services**

* Compute
* storage
* big data
* ML

Cloud computing is cloud computing is a way of using I.T. having traits.

* on-demand and self-service : processing power, storage, and network you need, with no human intervention.
* Broad network access : Global access over the internet
* Resource pooing: provider of resources has a big pool and allocates them to customers. Customers don't have care about location of resources.
* Rapid elasticity: Get more resources or scale back rapidaly
* Measured service: customers pay only for what they use or reserve. No usage -> no payment

**Every company is a data company**

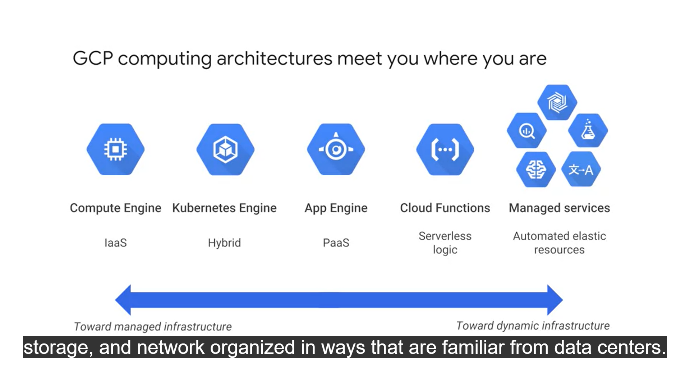
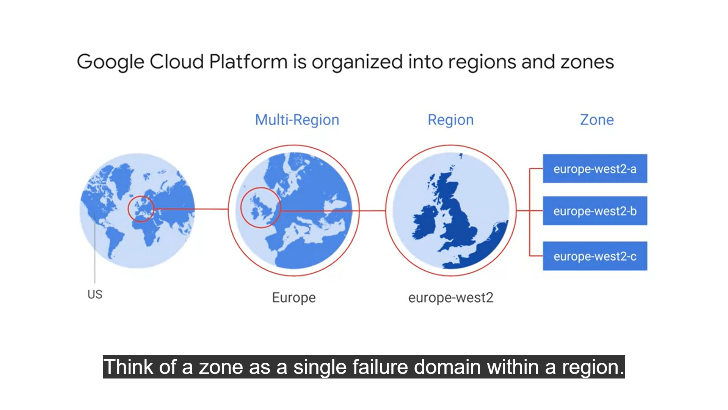
Cloud progressed through Colocation[shared facility] 🡪 virtualization[virtual interface to underlining HW] 🡪serverless[container based architecture]

**IaaS** [pay for allocated] **and PaaS** [Pay for Used]

Iaas : Compute – Storage – Network

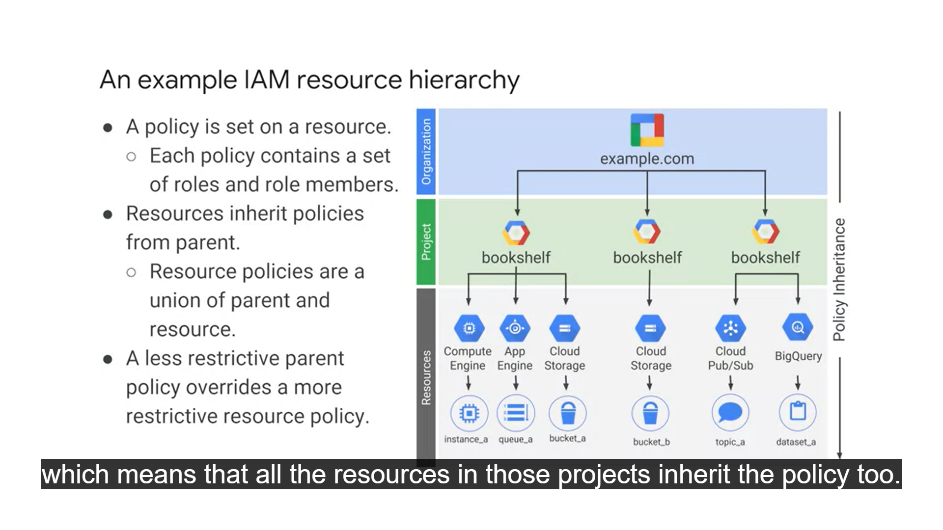
PaaS : Services

SaaS : softwares used online [gSuit]



**Energy**

* Google dataceters are 100% carbon neutral since 2007, have ISO14001 certification
* Google became 100% renewable energy company in 2017

**GCP USP**

* Per sec billing
* API to work out of GCP

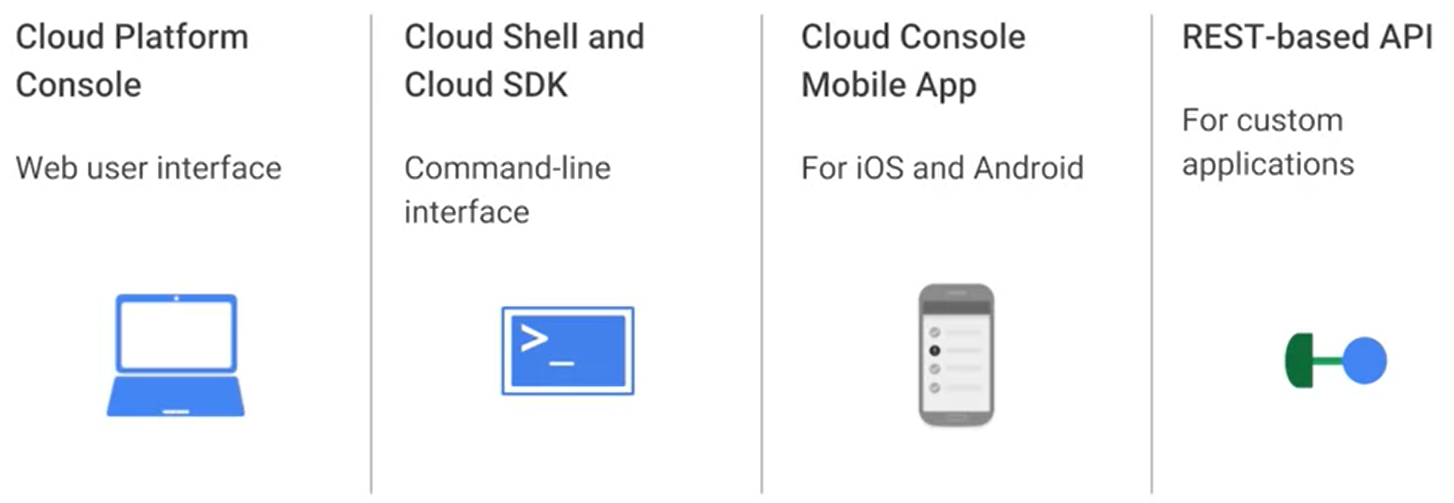
**GCP facilities**

* Budgets and alerts[set in % of billing], reports[visualization], Billing exports [to bigTable for analysis] ,quotas [kubernet API limits 1000 requests per 100 second, GCP limits 5 VPNs per project]

**IAM** [Identity and authorization management]

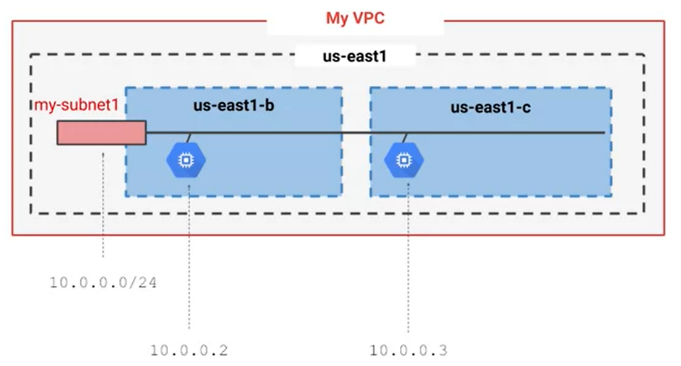
* **Who** [owner – editor – user – billing manager] – **can do what** [services offer predefined roles]– **on what services** [custom rules allow fine graining access to each user]
* **Service accounts** [authenticate a VM running application to GCP, so that only that VM instance can alter data]

**GCP interaction**



**VM**

Cloud compute instance enables a VM which hosts an OS with other SW resources

**VPC**

* Have resources in same subnet on different zones
* Firewalls [tag webserver VMs and apply central rule at compute instance for that tag], routing tables are auto managed
* Cloud load balancing

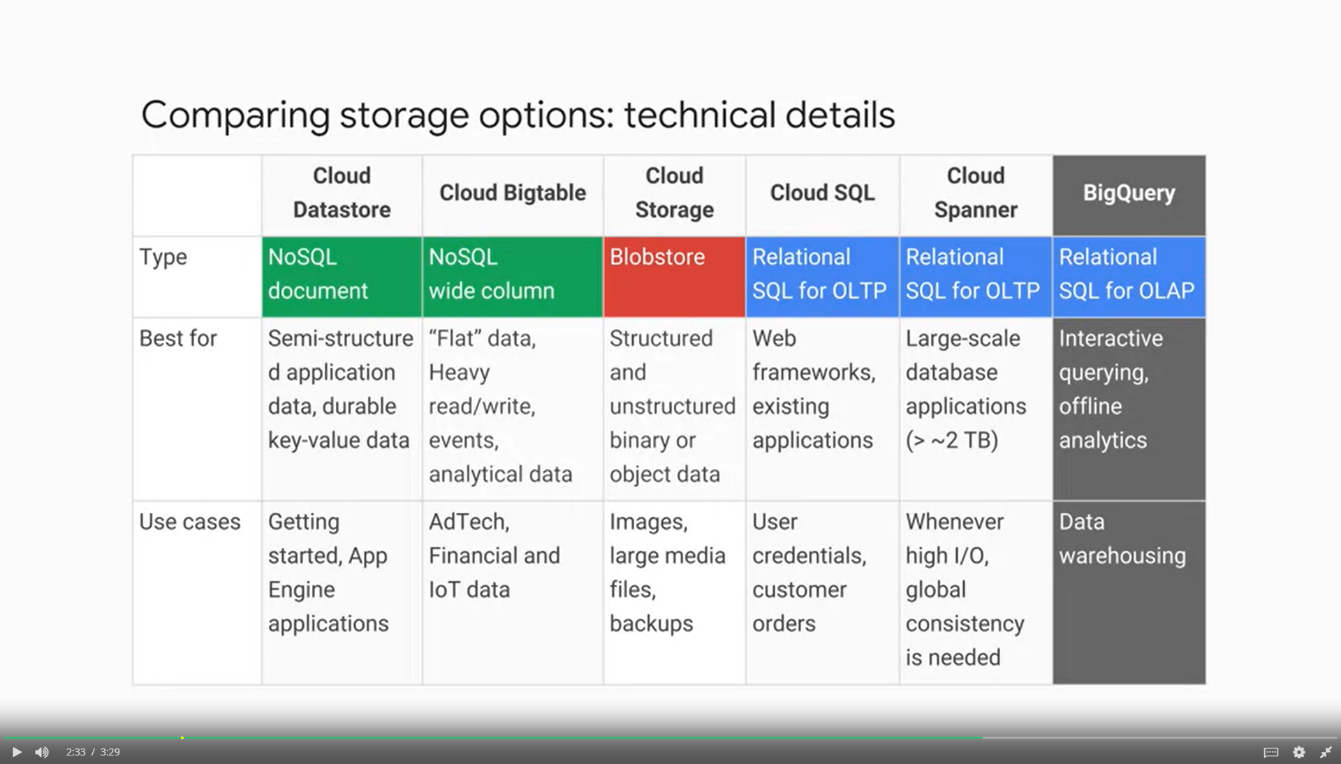
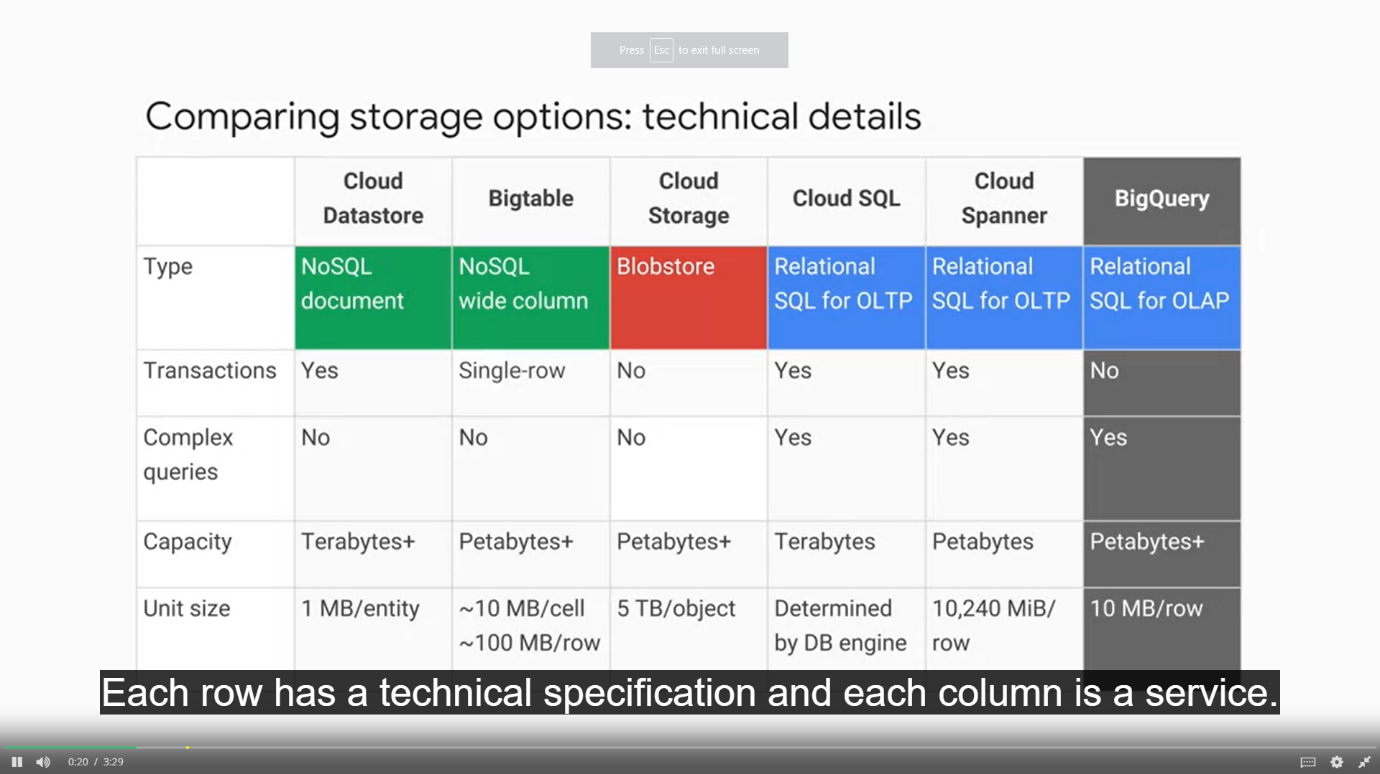
**Cloud CDN**

Google edge caches

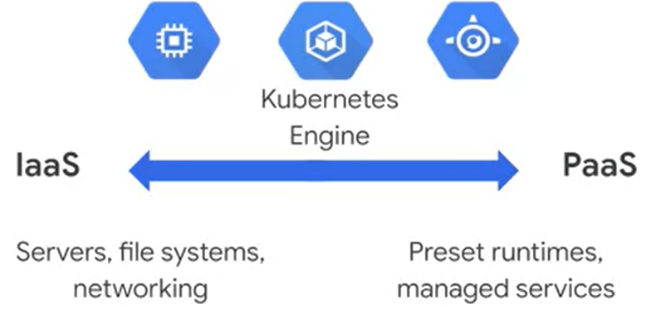
**STORAGE** [media streaming, sensor data storage]

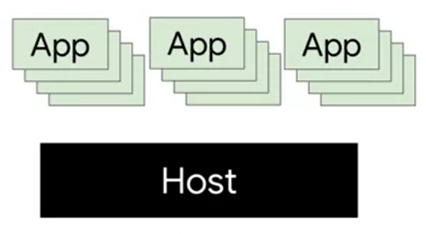
Apart from **VM persistent disc**, google provides following PaaS options for storage:

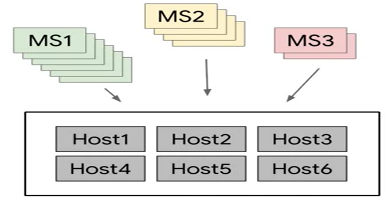
* **Storage [immutable large objects : movies, images]**
  + Object storage [Not file[hirarchical], Block[chunk of discs]]
  + Arbitrary data bytes; saved into “**Buckets**”. Buckets are accessible through unique key[URL].
  + Buckets have unique name, location and default storage class, access control list.
  + Buckets are Immutable, though versioning and lifecycle management is allowed. Web contents, achieving, direct dl
  + Storage transfer service allowes scheduling batch transfers for Petabytes of data stored
* **SQL [Online transaction]**
  + MySQL / PostgreSQL implementation
  + Read / failover/ external replica ensuing data persistane | Managed backups
  + NW firewalls, encryption
  + Vertical[R/W] scaling inflight
* **Spanner [online transactions using horizonal scaling]**
  + Horizontal scaling [outgrowth on flight, sharding database]
  + Transaction consistency at global scale
* **Datastore [transaction, structured objects]**
  + Application backend
  + Auto handling sharding and replication
  + SQL like queries across multiple tables in single line
  + Daily free quota of read write delete and small operation
* **Google Bigtable [large structured objects without SQL / multiline transaction]**
  + Fully managed NoSQL [All rows under a table may not need to have same column]
  + Persistent hash table, single look up key
  + Low latency high throughput [IoT, financial]
  + HBase API : easily portable[default for Hadoop]
    - Scalability without downtime
    - Encryption inflight and at rest
    - Streaming / batch processing [spark – Hadoop]



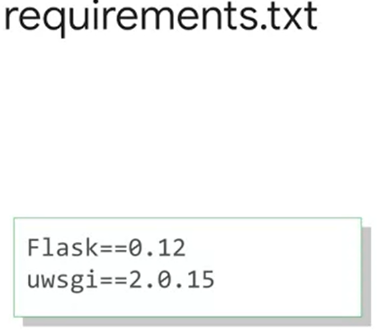
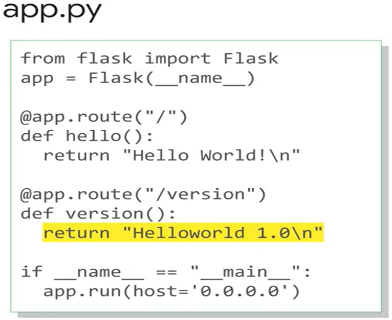
**Cloud blocks**

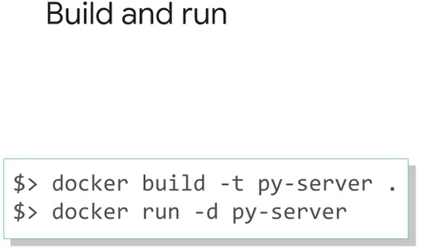
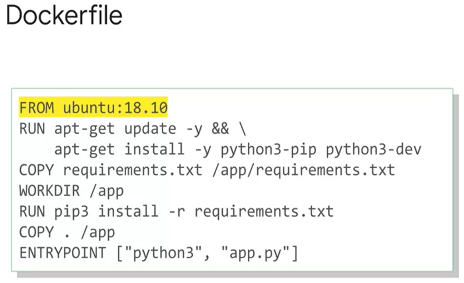
* Mix and match microservices running across clouds
  + **IaaS** : Compute Engine (Shared HW, flexible)
    - **Smallest unit is a Compute engine [in GBs]**
    - **Scaling out is creating new Compute instance with new OS!!!**
  + **PaaS** : App engine
    - Flexibility in server architecture is sacrificed, but apps are scaled independently
  + **Containers**
    - **Provide both OS HW layer abstraction[IaaS] and workload independence [PaaS]**

**Container**

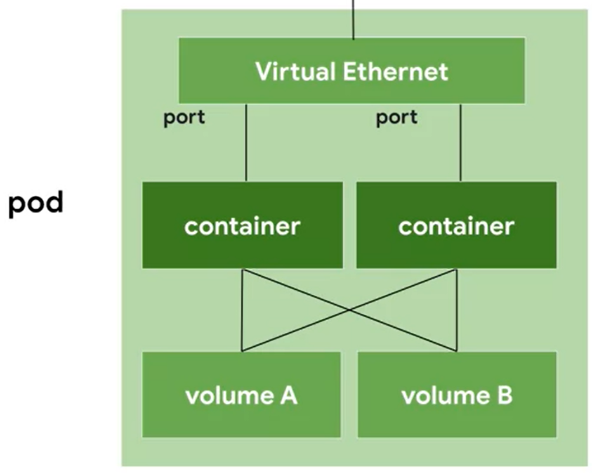
* **CONTAINER** is like a new process which runs on already booted infrastructure running apps [container run time]!!
  + **It virtualize OS rather than HW**
  + **It bundles app + dependencies into an entity**
* **KUBERNETICS**
  + Suppose multiple microservices need to run within containers to make up an app; apps can be made modular 🡪 deployed and scaled across group of hosts using kubernetics
  + Containers are made using formats defined by docker, google cloud build

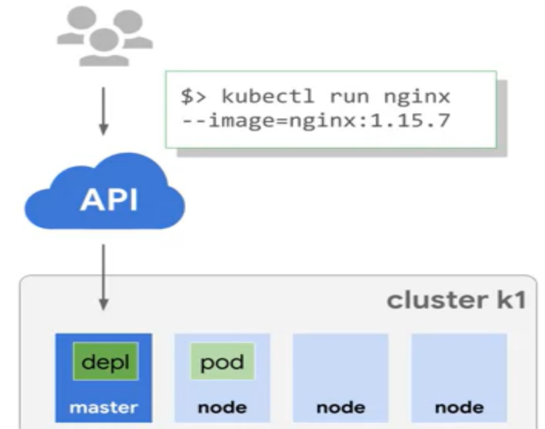
**Example of a container**





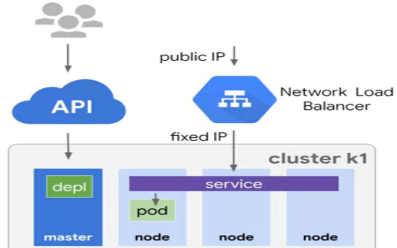
**+**  **=**

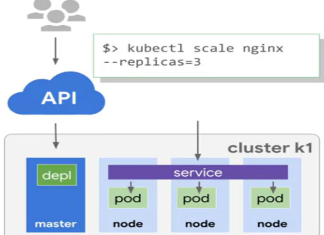
**KUBERNETICS is open source orchestrator for containers**

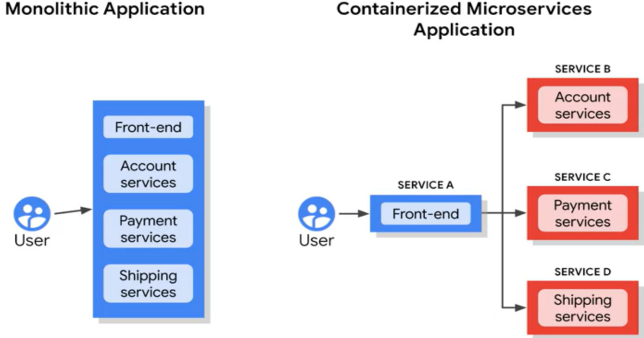
* **Let user deploy containers on cluster : set of nodes [compute instances]**
* Building and maintaining cluster is tedious
* **GKE clusters** let you build complete configured cluster
* **Pod :** single unit representing a cloud process
  + **Get unique IP and set of ports for individual containers**
  + **Mostly a single container / multiple containers with hard dependency**
  + **have shared hard disk storage and network access**

**RUNNING KUBERNET starts deployment of a container in a pod.**

* **Deployment**  is **running group of replicas of a pod**
  + **Pod RUN though some nodes FAIL**
  + A deployment may contain component of/ whole application
  + Pods are accessible only within cluster
    - **To make them visible to internet, they are “Exposed”; which attaches a LOAD BALANCER to pods.**
    - **This is done though creating a fixed IP service for each pod**
    - Service provide a stable interface for external world to pods since pods are dynamically created and destryed in runtime
    - A serivce proxies all pods under it umbrella



* + - **Thus Kubernetes creates a load balancer with public IP address to access the service**
    - **GKE providers NW load balancer as a service provided by compute engine to VMs**
    - **Replicas of pod can be made**
    - **This eases scaling applications**
    - Config files have facility to define **“Update strategy”**
    - **“Rolling update” inducts new version of pods one by one and preserves old pods till new ones are available and functioning. This enables downtime free update**

**Hybrid and multi-cloud**

* **Scaling traditional servers** need: procurement – installment – reconfiguration – dependencies
* This is **painful** since**:**
  + **Old server with old tech need resell**
  + **New server is always situated in face of bottlenecks**
* **Hybrid system allows:**
  + **Keep enterprise apps on premise and move other parts on cloud**
  + **Cloud specialized services [ML, DA, IoT, Long term storage] can be bought on fly**
  + **Migration is self paced**

**App engine**

* PaaS with built in webApp engines [NoSQL, in memory cache, load balancing, health check, logs, user authenication]
* Autoscale : Users pay for what they use : No user administration
* App engine SDK enable simple deployment and local debugging

**Development, deployment and monitoring**

* **Cloud source repository**
  + Git hosted and managed on cloud
  + Infinite private repo controlled via IAM
* **Cloud function**
  + Single purpose programs to respond server runtimes
  + Eg. Before ingesting an image uploaded to website, it needs preprocessing. For that, resources need to be allocated and later released on fly. This is unmanageable since opened requests may be 1 per day or 1 lakh in a millisecond
  + This event based program is fired when a request happens [cloud storage event/ pub-sub/ http call]
* **Deployment manager** [Infrastructure as code]
  + Setting storage and configuring GCP environment can be daunting
  + **Deployment manager** intakes configuration in [yaml, python] and generate declarative Template to automate cloud infrastructure creation
  + These can be version controlled in cloud source repo
* **Stackdriver [monitoring]**
  + Monitoring[uptime checks, alerts, dashboards]
  + Log [filer and view, export to bigquery]
  + Debug [connects production data to error line in source code [in repo]🡪 no need to follow though logs]
  + Traces [sample per url stats]
  + error reporting [groups error and report new error detected]
  + Deployment manager intakes configuration in [yaml, python] and generate declarative Template to automate cloud infrastructure creation
  + These can be version controlled in cloud source rep

**Serverless Big data services**

* **Dataproc** [per sec billing, 1 min min billing]

Build requested Hadoop cluster within 90 sec, on top of Compute engine VM as stated in config file

* + Preemptible instances are available for bath processing
  + Preferable when data size is known
* **Dataflow** [general purpose ETL tool for data anaylsis]
  + ETL, batch and continuous computation pipeline is built
  + No need to launch/ manage a cluster for pipeline ops
  + IoT, healthcare imaging, fraud detection
* **BigQuery** [pay as go]
  + Petabyte analytics werehouse
  + 100k rows per second ingress / load from cloud storage
  + Dataflow, Hadoop, spark integration
* **PubSub** [many to many async messaging]
  + Decouple publisher and subscribers
  + 10,00k messages / sec and beyond
  + IoT, streaming
* **Datalab**
  + **Jupyter notebook environment to implement cloud based apps**
  + **Integrated google services**