Google App Engine

Google App Engine

- It is a **Platform-as-a-Service** implementation providing services for developing and hosting scalable Web Applications.
- Leverage **Google's distributed infrastructure** to scale out the applications .

https://cloud.google.com/appengine/docs/the-appengine-environments

Standard Environment

- Standard environment is a mode where Google Cloud has pre-defined instance classes like B1, B2…F4_1G
- Appengine has pre-defined vm configurations and we will be charged based on **the instance running hours.**

Advantages

- Automatic security patches for managed VMs
- Deployment happens within seconds

Disadvantages

- No SSH access
- Standard environment applications are **single-homed, meaning that all instances of the application live in a single availability zone.**

App Engine Architecture

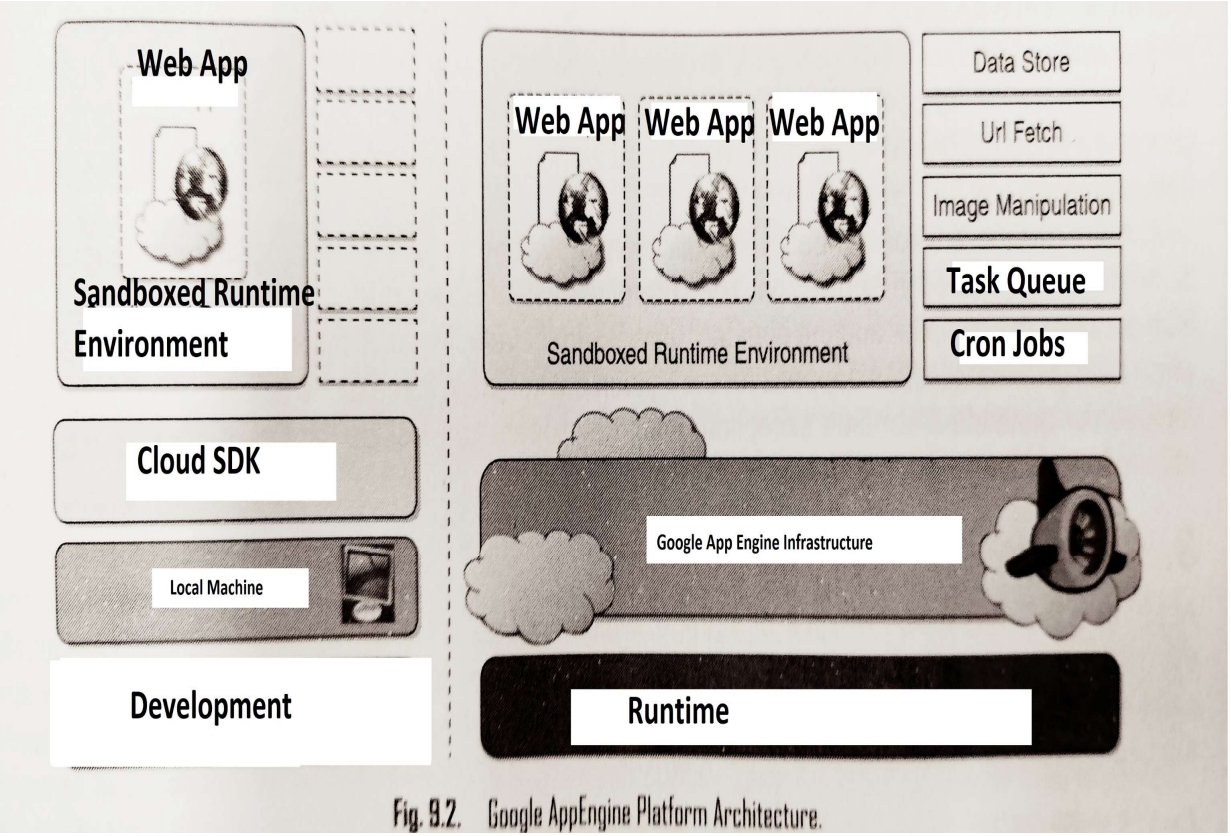

Fig. 9.2. Google AppEngine Platform Architecture.

Architecture and Core Concepts

- **Infrastructure:**

a) AppEngine infrastructure take **advantage of servers available within Google datacenters**

b) For each HTTP request:

i**. Locate the server hosting the application** processing the request.

ii. **Allocates more resources**, if required.

or

iii. **Redirects the request to an existing server.**

c) Responsible **for monitoring the application performance** and for collection of **statistics** for **billing**

Runtime Environment

- It represent **the execution context of applications hosted on AppEngine.**
- The runtime comes into **existence when the request handler starts** to execute and **terminates once** the handler has completed

Sandboxing

- Sandbox provides an application environment with an **isolated and protected context** where **applications can execute without constituting a threat to server** and without being influenced by other applications.
- If an application **try to perform any operation that is considered potentially harmful, an exception is thrown** and execution is interrupted.
- Some operation **not allowed in sandbox are: writing to server file system, executing code outside the scope of a request, processing a request for more than 30s**

Supported Runtimes

When to choose the standard environment

Application instances run in a sandbox, using the runtime environment of a supported language listed below.

Applications that need to deal with rapid scaling.

The standard environment is optimal for applications with the following characteristics:

- Source code is written in **specific versions of the supported programming languages**:
  - Python 2.7, Python 3.7, Python 3.8, Python 3.9, Python 3.10, and Python 3.11
  - Java 8, Java 11, and Java 17
  - Node.js 10, Node.js 12, Node.js 16, Node.js 18, and Node.js 20 (preview)
  - PHP 7.2, PHP 7.3, PHP 7.4, PHP 8.1, and PHP 8.2
  - Ruby 2.5, Ruby 2.6, Ruby 2.7, Ruby 3.0, and Ruby 3.2
  - Go 1.12, Go 1.13, Go 1.14, Go 1.15, Go 1.16, Go 1.18, Go 1.19, and Go 1.20
- Intended to **run for free or at very low cost**, where you pay only for what you need and when you need it. For example, your application can scale to 0 instances when there is no traffic.
- Experiences **sudden and extreme spikes of traffic** which require immediate scaling.

When to choose the flexible environment

Application instances run within Docker containers on Compute Engine virtual machines (VM).

Applications that receive consistent traffic, experience regular traffic fluctuations, or meet the parameters for scaling up and down gradually.

The flexible environment is optimal for applications with the following characteristics:

- Source code that is written in a version of any of the supported programming languages: **Python**, **Java**, **Node.js**, **Go**, **Ruby**, **PHP**, or **.NET**
- Runs in a Docker container that includes a custom runtime or source code written in **other programming languages**.
- Uses or depends on frameworks that include **native code**.
- Accesses the resources or services of your Google Cloud project that reside in the **Compute Engine network**.

https://cloud.google.com/appengine/docs/the-appengine-environments

Storage

- Three different levels of storage:
  a) In-memory cache - **MemCache**
  b) storage for semi-structured data - **DataStore**
  c) long term storage for static data - **Static File Servers**

Storage: Static File Servers

- **Static data** is mostly constituted by components that define the **graphical layout of the application (css file, plain html files, javascript, images, icons, sound files) or data files.**

- These files can be hosted **on static file servers since they are not frequently modified.**

- These servers are **optimized for serving static content.**

Storage: DataStore

- It is a **service allowing developers to store semi-structured data**.
- Service is designed **to scale and quickly access the data.**
- DataStore can be **considered as a large object database where objects can be stored and retrieved by a specific key.**
- It also provide **facility to create index on data and update data within context of transaction.**
- **Data is persisted to disk redundantly** in order to ensure reliability and availability

Storage: MemCache

- AppEngine caching services.
- **Distributed in-memory cache that is optimized for fast access** and provide developer with a volatile store for objects that are frequently accessed.
- Caching algorithm automatically **remove the objects used rarely.**

Each object is first looked in MemCache and if there is a miss, it will be retrieved from DataStore and put into the Cache for future look-ups.

Application Services

- These services **simplify most of common operations that are performed in Web applications:**
  - ✓ Access to data
  - ✓ Account Management
  - ✓ Integration of external resources
  - ✓ Messaging and communication
  - ✓ Image manipulation
  - ✓ Asynchronous computation

Application Services

- **UrlFetch:**

✓ The **sandbox environment** don't allow applications to open arbitrary connections through sockets but **it provides developers with the capability of retrieving the remote resource through HTTP/HTTPS by means of UrlFetch service.**

✓ Deadlines can be set for a request to ensure timely completion

✓**Asynchronous Request:** Applications continue with the logic while resource is retrieved in background

- **MemCache: App Engine Cache Services**

Application Services: Mail and Instant Messaging

Mail

- AppEngine provide **developer with the ability to send and receive the** mails through *Mail service.*
- The service **allow to send email on behalf of application of specific user account.**
- It is also possible to **include several types of different attachments and target recipients.**

Application Services: Mail and Instant Messaging

Instant Messaging: XMPP

(Google Talk)

Extensible Messaging Presence Protocol. It's protocol for streaming XML elements over a network in order to exchange messages and presence information in close to real time.

X : It means eXtensible. XMPP is a **open source project which can be changed or extended according to the need.**

M : XMPP **is designed for sending messages in real time**.

P : It determines **whether we are online/offline/busy. It indicates the state.**

P : XMPP is a **protocol, that is, a set of standard**s that allow systems to communicate with each other

Account Management

- Developers **can make use of Google Account Management by means of *Google Accounts.***

- ***The data of interaction* with the user normally** goes under the name of **user profile and are attached to an account.**

- *Integration* also allows web applications to **offload the implementation of authentication capability to Google's authentication system**

Image Manipulation

- AppEngine allows **applications to perform resizing, mirroring, rotation and enhancement** by means of Image Manipulation, a service designed for **light-weight image processing and optimized for speed.**

Compute Services

- Most of the interaction is performed synchronously: users navigate the web pages and instantaneous feedback is received as response.

- The **feedback is result of some computation happening** on Web Application.

- AppEngine also provides **services that simplify the execution of computations that are off-bandwidth** or **cannot be contained within the time frame of Web request** handling: **Task Queues and Cron Jobs**

Compute Services: Task Queues

- It allow **applications to submit a task for later execution**.
- Useful for services **that cannot complete within the maximum response time of a request handler.**
- Service allows users **to have up to 10 queues that** can execute tasks at a configurable rate.
- Queue is **designed to re-execute the task in case of failure.**

Compute Services: Cron Jobs

- Required operation **needs to be performed at a specific time of the day, which does not coincide with the time of web request.**

- In this case, it is possible to schedule the required operation at desired time, by **using the Cron Jobs service.**

- ***The service operates similar to TaskQueues but invokes the request handler specified in the task at a given time and does not re-execute the task in case of failure.***

Application Life Cycle

- Support for all phases characterizing the life cycle of application:

✓ Development and Testing

✓ Deployment

✓ Monitoring

Application Development and Testing

- Developers can **build their Web Applications on a local development server.**

- This is a self contained environment that help developers in simulating AppEngine **runtime environment by providing a mock implementation of DataStore, MemCache, UrlFetch and other services leveraged by Web applications**.

- Development server also **feature complete set of monitoring features helpful to profile the behaviour of applications**

- Indexes are also built to speed the access on relevant data.

Application Development and Testing: Cloud SDK

- Google Cloud SDK **contains tools and libraries that enable you to easily create and manage resources on Google Cloud Platform**, including App Engine , Compute Engine , Cloud Storage , BigQuery , Cloud SQL , and Cloud DNS.

Key features include

- **Cloud Client Libraries** - These are available in multiple languages for **programmatically integrating Google Cloud API into the code**

  These are **available for Java, Python, Node.js, Go, Ruby, C#, PHP**
- **Google Cloud CLI- Command line tools** that we can **install to access and manage the cloud from command line**
- **Tools and Documentation**
- **Cloud Code: Extension** that helps to **write, debug, deplo**y cloud-native apps from **VS Code or IntelliJ IDE**.

https://cloud.google.com/sdk#all-features

Application Deployment and Management

- After application development and testing, **it can be deployed on AppEngine.**
- An application identifier is mandatory **to allow unique identification of application while interacting with AppEngine**.
- Once an application identifier has been created, it is possible to deploy an application on AppEngine. This can be done either by using **the respective development environment (GoogleAppEngineLauncher and Google AppEngine Plugin) or command line tools.**
- Developers can then manage the application by using the administrative console.

Cost Model

- Pricing is different for **apps in the standard environment and the flexible environment.**

App Engine standard environment pricing

- Apps in the **standard environment have a free tier for App Engine resources**. Any use of App Engine resources beyond the free tier incurs charges.
- Charges **are specified per hour per instance.**

App Engine flexible environment pricing

- App Engine **does not provide a free tier** in the flexible environment.
- Apps **running in the flexible environment are deployed to virtual machine types that you specify.** These virtual machine resources are billed on **a per-second basis with a 1 minute minimum usage cost.**
- Billing for the memory resource **includes the memory your app uses plus the memory that the runtime** itself needs to run your app.

Quotas

- Google app engine uses **quotas to ensure that users don't spend more than the allocated budget and that application run without being influenced by each other from a performance perspective.**

- Application is measured **against billable quotas, fixed quotas and per-minute quotas**.

- **Billable quotas identify the daily quotas** set by the application administrator and are defined by the **daily budget allocated for the application**. Free quotas are a part of billable quotas and identify portions of the quota for which users are not charged.

- **Fixed quotas are internal quotas set by the AppEngine that identify the infrastructure boundarie**s and define the operations that applicants can carry out on infrastructure. These quotas are set up to avoid impact of applications on each other performance and overloading the infrastructure.
- **Per-minute quotas are defined in order to avoid applications consume all the credit in a limited time** period**, monopolize a** resource and create service interrupts for other applications.

Once an application reaches the quotas for a given resource, the resource is **depleted and subsequent access to application will result in an error or exception.**

References

- MASTERING CLOUD COMPUTING, Rajkumar Buyya , Christian Vecchiola, S. Thamarai Selvi, McGraw Hill Education