

# Acknowledgement

This group project that we were assigned with gave us a tremendous impact towards enhancing our knowledge. In performing our project, we had to take the help and guideline of some respected persons to whom we would like to express our deepest gratitude. We are thankful to Prof. D.B. Phatak and IIT-Bombay for providing us with this internship opportunity and the necessary resources for our project. We would like to thank our mentors Ms. Aparna Pansare and Ms. Supriya Tambe whose guidance and constant encouragement was indispensable in the completion of this project within the stipulated time. We are grateful to all the system administrators in Software Lab, New Computer Science Department for always coming to our aid when we needed. We express our foremost gratitude to our program coordinator Mrs. Aruna Adil for organizing and managing the internship program in such a student friendly way. We would also thank Mr. Rahul Kharat and Mr. Bikas Chhatri for all the administrative help, making our work environment comfortable. Last but not the least, we thank our fellow interns whose ideas and valuable suggestions were very helpful to solve many challenges that we faced and for making our internship experience an enjoyable and memorable one.

# **Summer Internship 2018**

## **Project Approval Certificate**

### **Department of Computer Science and Engineering Indian Institute of Technology Bombay**

The project entitled **Implementation of content repository for Project Sunbird** submitted by Ms. Arushi Jain, Mr. Prakash Kumar, Mr. Tanmoy Ghosh and Mr. Utkarsh Shukla is approved for Summer Internship 2018 Ekalavya programme from 21th May 2018 to 6th July 2018, at Department of Computer Science and Engineering, IIT Bombay.

Prof. Deepak B. Phatak

Ms. Aparna Pansare  
(Mentor in charge)

Place: IIT Bombay, Mumbai

Date: 6 July, 2018

# Declaration

I declare that this written submission represents my ideas in my own words and whenever others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Ms. Arushi Jain  
IIT Indore

Mr. Prakash Kumar  
NIT Delhi

Mr. Tanmoy Ghosh  
NIT Durgapur

Mr. Utkarsh Shukla  
NIT Durgapur

## Table of contents

<b>LIST OF FIGURES</b>	<b>6</b>
<b>ABSTRACT</b>	<b>8</b>
<b>INTRODUCTION</b>	<b>9</b>
Platform	9
Sunbird Platform Architecture	9
Motivation	12
Problem Statement	12
<b>PROJECT REQUIREMENTS</b>	<b>12</b>
Software Requirements	12
Hardware Requirements	12
<b>PROJECT EXECUTION APPROACH</b>	<b>13</b>
<b>STORAGE ARCHITECTURE</b>	<b>13</b>
Cloud Storage	14
Fig. 5.1.1 : Cloud Computing deployment models	15
OpenStack Object Storage and Identity Service	15
<b>PROJECT DETAILS</b>	<b>19</b>
<b>IMPLEMENTATION DETAILS</b>	<b>20</b>
User and Organization Management APIs	20
Encapsulating Swift APIs in Django APIs	30
Understanding the Content Flow	38
Distributing Django Cloud Storage APIs	42
Kong API: Onboarding APIs on Sunbird Kong API container	43
New Feature: IITBX Course on Sunbird	45
Storing Content to Swift	48
Build process for deploying source code changes using local registry	50
<b>DELIVERABLES</b>	<b>50</b>
<b>CONCLUSION</b>	<b>50</b>
<b>FUTURE SCOPE</b>	<b>51</b>
REFERENCES	51

<b>APPENDIX</b>	<b>52</b>
Appendix A: Angular.js	52
Appendix B: Ansible	55
Appendix C: Docker	59
Appendix D: Kong	68
Appendix E: Node.js	71
Appendix F: Introduction of REST API and Django Rest Framework	73
Appendix G: Sunbird Installation	76
Appendix H: Swift Installation	80

## **LIST OF FIGURES**

**Fig. 2.1.1 :** Sunbird Request Flow Diagram

**Fig. 5.1.1 :** Cloud Computing deployment models

**Fig. 5.2.1 :** Storage Concepts ( Block Storage vs Object Storage )

**Fig. 5.2.2 :** Swift Architecture Overview

**Fig. 7.1.1 :** Request Description for creating a new Organisation

**Fig. 7.1.2 :** Request Description for adding a User to a Organisation

**Fig. 7.1.3 :** Entity-Relationship Diagram in Cassandra Query Language

**Fig. 7.1.4 :** Swift API : Retrieve container

**Fig. 7.1.5 :** Swift API : Create new container

**Fig. 7.1.6 :** Swift API : Update container metadata

**Fig. 7.1.7 :** Swift API : View container metadata

**Fig. 7.1.8 :** Swift API : Delete empty container

**Fig. 7.1.9 :** Swift API : Retrieve objects in a container

**Fig. 7.1.10 :** Swift API : Create new object in container

**Fig. 7.1.11 :** Swift API : Download Object

**Fig. 7.1.12 :** Swift API : View object metadata

**Fig. 7.1.13 :** Swift API : Update object metadata

**Fig. 7.1.14 :** Swift API : Delete Object

**Fig. 7.1.15 :** Use Case Diagram for Course Creation

**Fig. 7.1.16 :** Data Flow Diagram for Course Creation

**Fig. 7.1.17 :** EkStep Content Archive (ECAR) file format Manifest

**Fig. 7.1.18 :** Request description for creating content

**Fig. 7.1.19 :** Hierarchy of content API details

**Fig. 7.1.20 :** Request description for publishing content

**Fig. 7.1.21 :** Terminal showing the newly created docker container for Django server

**Fig. 7.1.22 :** Example of an entry in main.yml (ansible) file for onboarding API to kong

**Fig. 7.1.23 :** Showing newly onboarded API in Kong API list

**Fig. 7.1.24 :** IITBX courses panel view in Sunbird portal

**Fig. 7.1.25 :** IITBX course detail view

**Fig. 7.1.26 :** Menu option for creating IITBX course

**Fig. 7.1.27 :** Create course form

**Fig. 7.1.28 :** Request Description for getting IITBX courses list

**Fig. 7.1.29 :** Request Description for getting course details for an IITBX course

**Fig. 7.1.30 :** Request Description for getting an object of an IITBX course

**Fig. 7.1.31 :** Request Description for deleting an object of an IITBX course

**Fig. 7.1.32 :** Request Description for creating an IITBX course

**Fig. 7.1.33 :** Request Description for deleting an IITBX course

**Fig. 7.1.34 :** Function definition for storing content metadata in Swift

**Fig. 7.1.35** : Request response showing file uploaded to Swift

**Fig. 7.1.36** : Swift container showing uploaded metadata file

**Fig. 13.A.1** : Angular.js data binding

**Fig. 13.A.2** : Angular.js template, controller and service relation

**Fig. 13.A.3** : Angular.js Model View Controller architecture

**Fig. 13.B.1** : Ansible playbook connected with hosts

**Fig. 13.B.2** : Ansible installation commands in UBUNTU-16.04 LTS

**Fig. 13.B.3** : Ansible playbook example

**Fig. 13.B.4** : Ansible running playbook

**Fig. 13.C.1** : Docker architecture

**Fig. 13.D.1** : Kong architecture

**Fig. 13.D.2** : Kong transforming APIs

**Fig. 13.E.1** : Node.js Event loop

**Fig. 13.F.1** : REST API Request/Response Flow Diagram

## 1. ABSTRACT

Sunbird is one of the online learning initiative at IIT Bombay. This platform use Open Source technologies extensively. This also use cloud as a data store for all types of content. This has also extensively adopted **REST(Representational State Transfer)** based microservices as building blocks for building the entire platform.

OpenStack Swift is an Open source cloud storage software providing ways to store and retrieve data on cloud using REST APIs.

This project aims to explore the possibility of enhancing the Sunbird platform to adopt OpenStack Swift as an alternate cloud based data store and to provide a direction for accomplishing the same. This project is for assessing the feasibility to use OpenStack Swift as a content store for course data and for identifying the code changes needed.



## 2. INTRODUCTION

### 2.1. Platform

#### **Sunbird Platform Architecture**

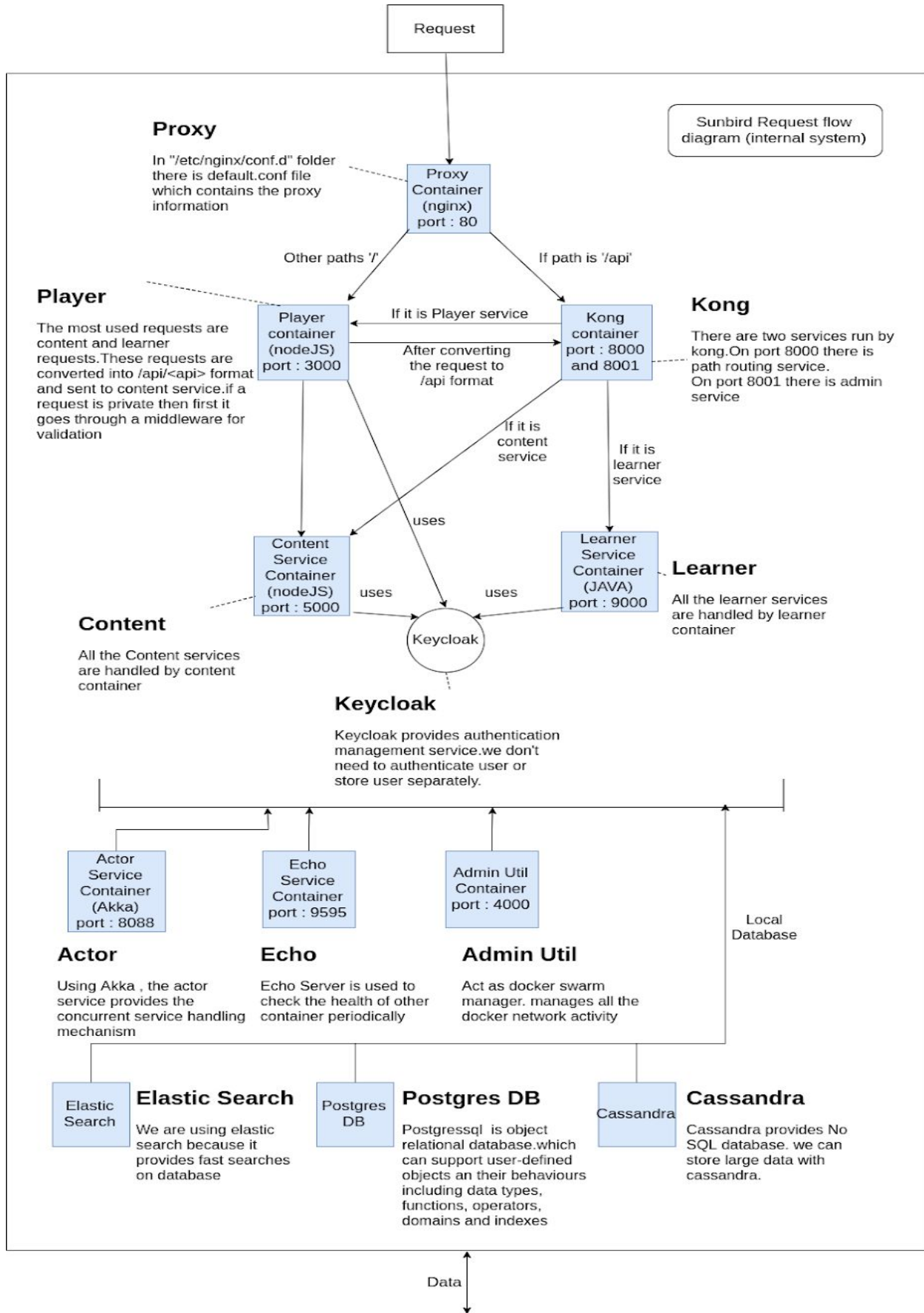
- **What is Sunbird?**

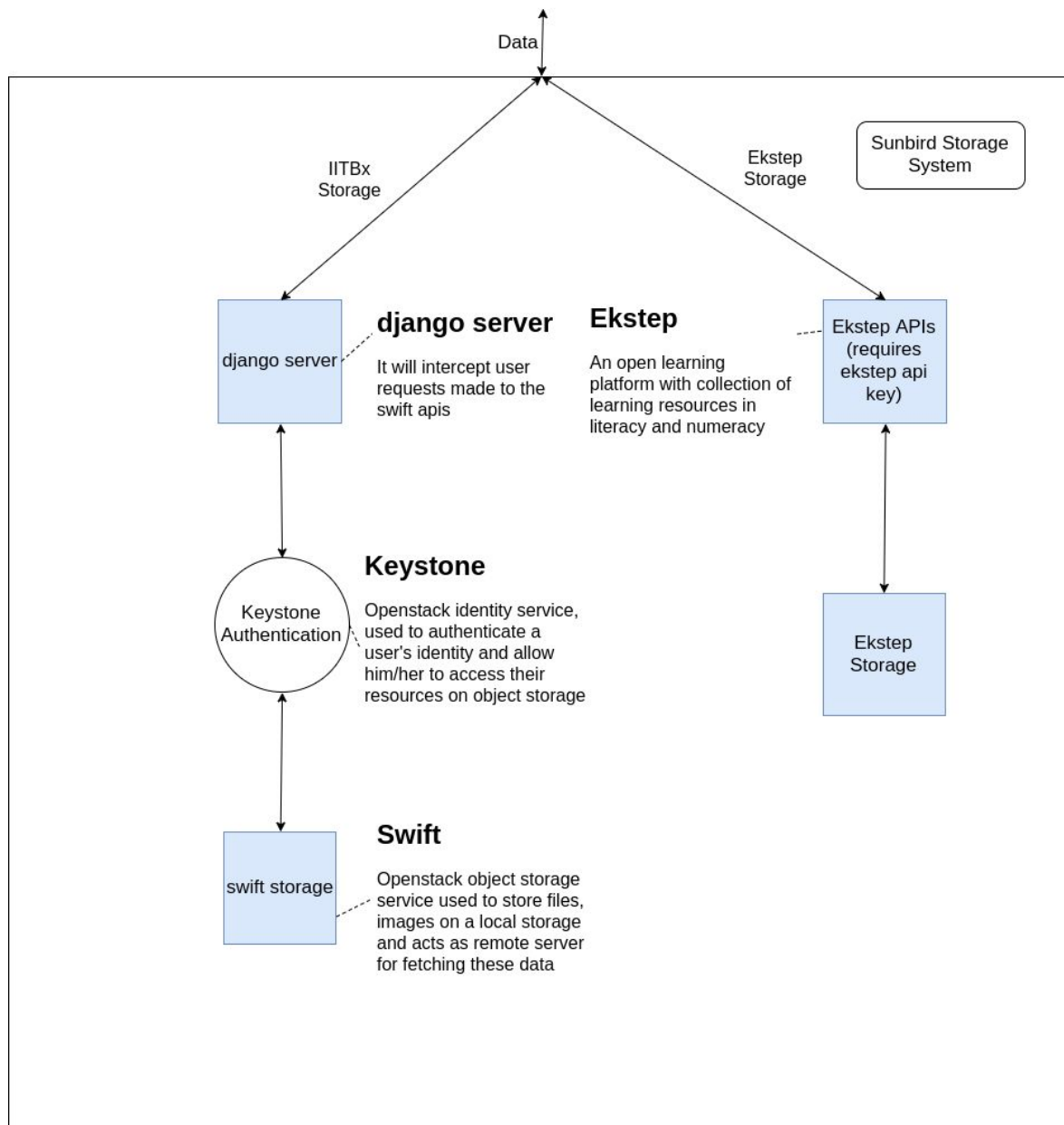
Sunbird is an open source, configurable, extendable, modular learning management platform architected for scale and designed to support multiple teaching and learning solutions supporting multiple languages and available for online and offline use.

In simple words, Sunbird is an platform for conducting online courses.

- **Sunbird request flow diagram**

Sunbird has a very complex architecture consisting of multiple components which can be deployed in a scalable and distributed manner. The different integrable components are distributed and deployed using containerization technologies. The containers running on Sunbird are: content service, player service, learner service, actor service, kong, proxy service, admin util, and echo service. Additionally keycloak is running as authentication management service with Sunbird. Postgres and Cassandra are the databases used and the search engine used is ElasticSearch. All these components work together and make possible for Sunbird to work properly. The basic request flow diagram is shown in below figure:





**Fig. 2.1.1 : Sunbird Request Flow Diagram**

## **2.2. Motivation**

Sunbird use cloud based storage to store the data related to course, students, submissions, telemetry. Sunbird sends content related request to Ekstep foundation and then Ekstep stores and retrieves data from Amazon S3. There are many benefits of having a storage solution based on an Open Source cloud solution over using other solutions. These benefits include cost reduction, control over data and infrastructure, customization, security and privacy, compliance. Considering these benefits, we decided to create private cloud for storage at IIT Bombay.

## **2.3. Problem Statement**

The scope of the project was to explore the architecture of Sunbird platform and to identify a feasible approach for using a private cloud for the data storage needs of the Sunbird platform. Following is the problem statement on which we worked:

- Implementation of content repository for Project Sunbird

# **3. PROJECT REQUIREMENTS**

## **3.1. Software Requirements**

The following are some of the core technologies on which Sunbird platform is based.

- OpenStack Swift
- OpenStack Keystone
- Docker
- Ansible
- NodeJS
- AngularJS
- Django, Django REST Framework
- Kong API Gateway

You can find details about these technologies in the [Appendix](#).

## **3.2. Hardware Requirements**

- Sunbird: Virtual machine running on Ubuntu 16.04 LTS with storage capacity of at least 60GB and 7GB RAM. We were able to install the Sunbird platform on a single VM.
- OpenStack Swift : Virtual machine running on Ubuntu 16.04 LTS with storage of at least 60GB and 8GB RAM

## **4. PROJECT EXECUTION APPROACH**

A software development process based on Agile software methodology was used for the implementation. This approach was chosen as it was the fastest way to deliver a feasibility report for implementing a complex solution on a complex platform within the available time frame. Adopting this approach meant that the amount of documentation would be reduced to a minimum and the major deliverables would be code and detailed instructions to accomplish the assigned tasks.

The activities performed were as below:

1. Study the technologies needed and create an ‘expert’ on the technology
2. Install the required technologies and create working prototypes for using the technologies
3. Study the architecture of Sunbird and identify relevant code and scripts
4. Study the build process of Sunbird
5. Coding to demonstrate the feasibility of the solution
6. As the project involved a lot of new technologies, a detailed step-by-step documentation was created so that it is easily available to future teams

Steps taken to ensure timely completion of tasks:

1. Regular meetings to track progress
2. Immediate resolution of technical difficulties
3. Working in pair where a senior programmer would allocate work and solve technical difficulties promptly

## **5. STORAGE ARCHITECTURE**

### **5.1. Cloud Storage**

- **What is Cloud?**

A Cloud is a computing model of powerful computers and programs that act as a remote server for storing and accessing data. In the simplest terms, cloud computing means storing and accessing data and programs over the Internet instead of your computer's hard drive, and accessing this data from anywhere via API calls.

- **How does a cloud work?**

Cloud storage works on the concept of data center virtualization which provides end users and applications, a virtual storage architecture that is scalable according to application requirements along with other networking services. Thus, instead of a program or document being stored on your computer, all data and functionality is stored in the cloud, which means that you can save dedicated hard drive space on your devices and access that data and service as and when required from anywhere using web-based API as long as you've got an internet connection and a compatible device.

- **Types of cloud service**

The many cloud services provided by cloud service providers (CSPs) often fall into one of three categories: A software available online, known as Software-as-a-Service (SaaS), a computing platform for developing or hosting applications, known as Platform-as-a-Service (PaaS) or an entire networking or computing infrastructure, known as Infrastructure-as-a-Service (IaaS).

- **Types of cloud storage service**

It is often divided into three categories: private, public, and hybrid, referring to who has access to the services or infrastructure. Public-cloud services are made available to anybody that wants to purchase or lease the services. Private-cloud services are built by enterprises for use by their employees and partners only. Hybrid-cloud services combine the two.

**Advantages and Disadvantages of Public, Private and Hybrid Services:**

Advantages of private cloud storage include high reliability and security. But this approach to cloud storage provides limited scalability and requires on-site resources and maintenance.

Public cloud storage offers high scalability and a pay-as-you-go model with no need for an on-premises storage infrastructure. However, performance and security measures can vary by service provider. In addition, reliability depends on service provider availability and internet connectivity.

Hybrid cloud storage offers the best of the private and public cloud with high scalability and on-premises integration that adds more layers of security. The result is better performance and reliability because active content is cached locally. While a hybrid cloud tends to be more costly than public storage, it is cheaper than private cloud storage. Reliability can be an issue, as users must depend on service provider availability and internet connectivity.

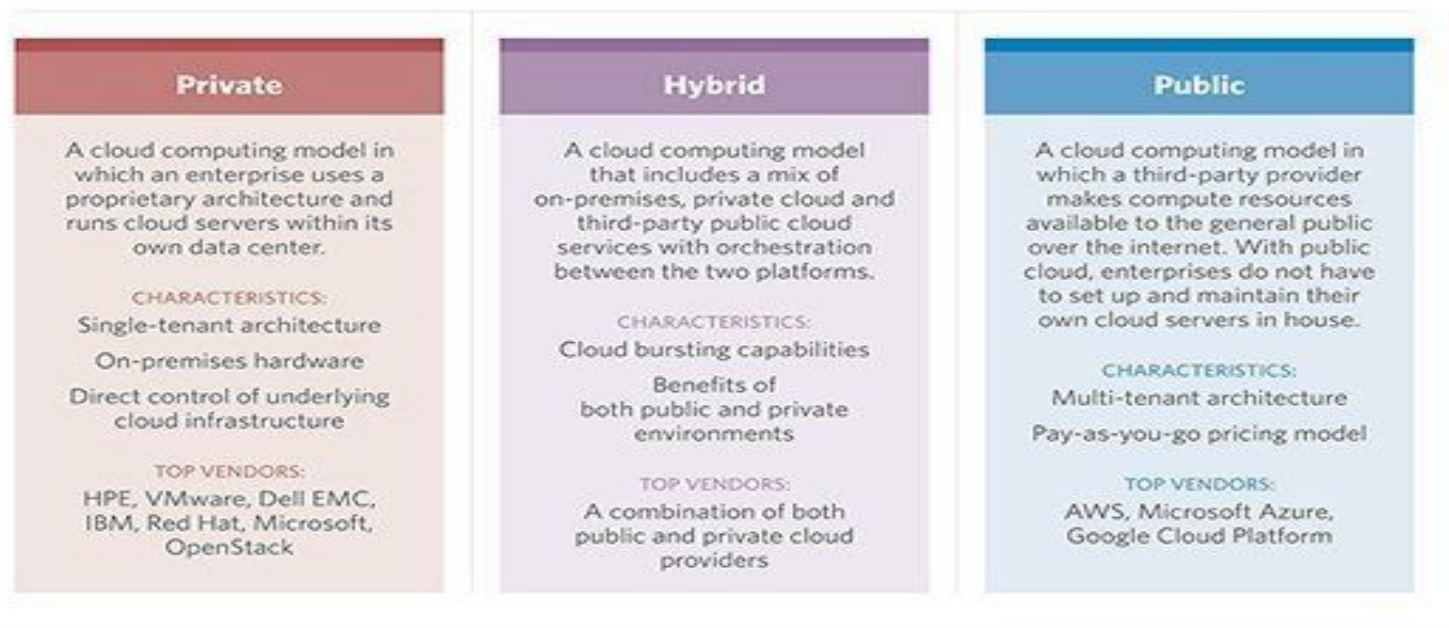


Fig. 5.1.1 : Cloud Computing deployment models

## 5.2. OpenStack Object Storage and Identity Service

- **Object Storage:**

Object storage is a computer data storage architecture that manages data as objects. It is different from forms of storage like file systems storage which manage data as a file hierarchy, and block storage which manages data as blocks within sectors and tracks. Here, each object typically includes the data itself, a variable amount of metadata, and a globally unique identifier. You can store any kind of data, photos, videos, and log files. The object store guarantees that the data will not be lost. Object storage data can be replicated across different data centers and offer simple web services interfaces for access.

The popular object storage options available are:

- 1) Amazon S3
- 2) Openstack Swift
- 3) Azure Blob Storage
- 4) Google Cloud Storage

- **Openstack Swift:**

OpenStack Swift, also known as OpenStack Object Storage, is open source software designed by Openstack community to manage the storage of large amounts of data cost-effectively on a

long-term basis across clusters of standard server hardware. Common use cases for OpenStack Swift include the storage, backup and archiving of unstructured data, such as documents, static web content, images, video files, email and virtual machine images.

- **Why use Swift?**

Ever since Swift was launched 2010, its popularity has grown over the years. The main reason behind this is that it is open source, i.e anyone can deploy it on their own hardware and have access to its technology. Thus the user has full ownership of their data and thus security related threats are eliminated. This gives Swift an upper-hand over Amazon S3, where security related threats often scare the users.

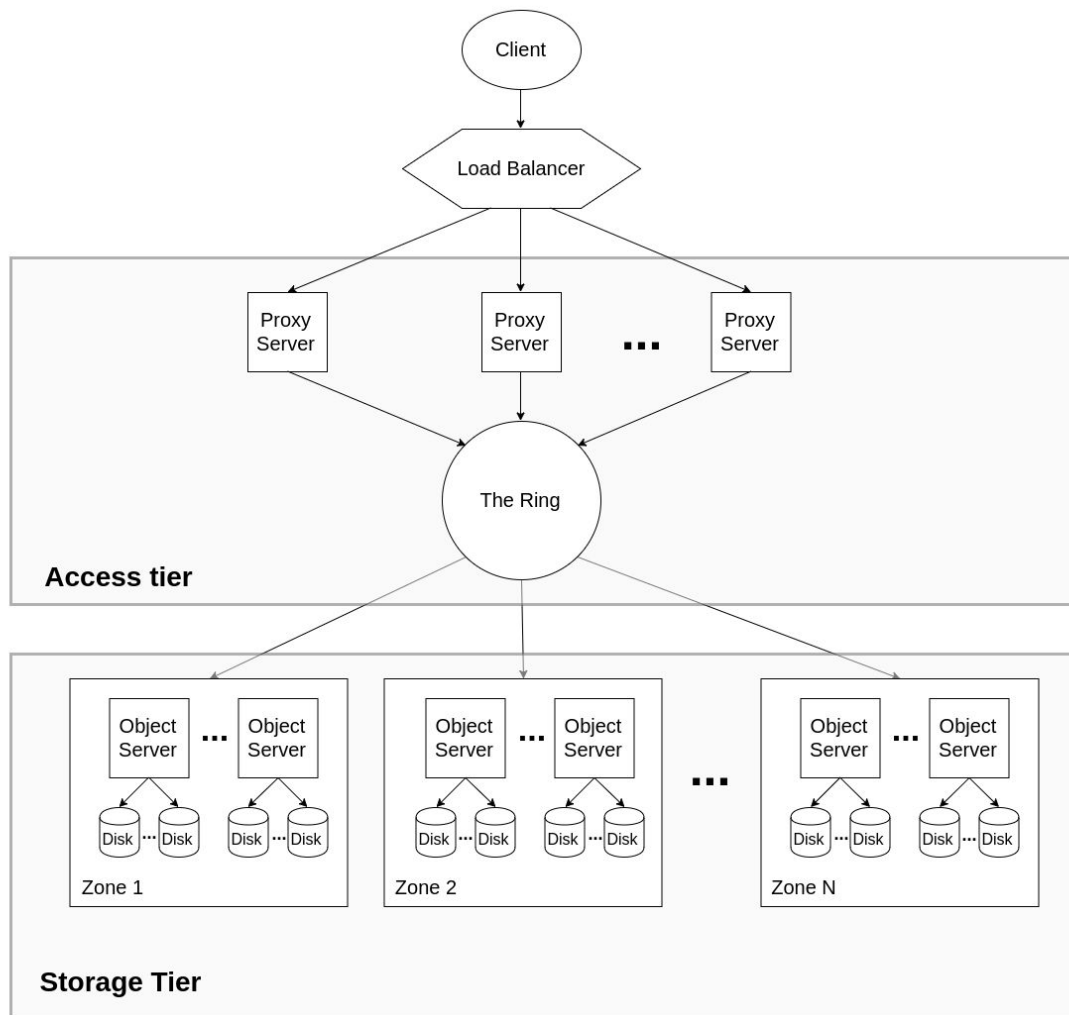
The other reason for using Swift is the accessibility it provides. Being an object storage platform, it allows the data to be accessed from anywhere, using REST APIs.

	Block Storage	Object Storage
Use	Add persistent storage to a VM	Store data, e.g., VM images, web graphics, video files
Accessed through	Operating System Kernel, often a file system is applied on top	REST API
Accessible from	Within a VM	Anywhere via HTTP(S)
Managed by	Cinder	Swift
Persists until	Deleted by user	Deleted by user
Sizing determined by	Specified by user in initial request	Amount of available physical storage
Example of typical usage	1 TB disk	10s of TBs of dataset storage

**Fig. 5.2.1 : Storage Concepts ( Block Storage vs Object Storage )**



- **Swift Architecture**



**Fig. 5.2.2 : Swift Architecture Overview**

**Access Tier:**

This tier consists of Load Balancer, Proxy Servers and the Ring structure.

- ➔ The Load Balancer takes care of the traffic (caused by the incoming and outgoing requests) by using a queue data structure.
- ➔ The Proxy server helps in directing the incoming requests to the respective user's account. For each request, it will look up the location of the account, container, or object in the ring and route the request accordingly.
- ➔ A Ring Structure represents a mapping between the names of entities stored on disk and their physical location. There are separate rings for accounts, containers, and one object ring per storage policy. When other components need to perform any operation on an object, container, or account, they need to interact with the appropriate ring to determine

its location in the cluster. The Ring maintains this mapping using zones, devices, partitions, and replicas. Each partition in the ring is replicated, by default, 3 times across the cluster, and the locations for a partition are stored in the mapping maintained by the ring. The ring is also responsible for determining which devices are used for handoff in failure scenarios.

### ***Capacity or Storage Tier:***

This tier consists of the Account Server, Container Server, Object Server and Consistency Server.

- The Account Server represents the top-level of the hierarchy and is responsible for the listings of containers.
- The Container Server defines a namespace for objects. It maintains a list of all the objects stored in it, in form of a sql database file.
- The Object server stores data content, such as documents, images, and so on. We can also store custom metadata with an object.
- Consistency Server helps in maintaining overall consistency in Swift through Replication of data, Updation and Auditing.

### ***Security Services:***

There are two authentication system used for securely using Swift storage. They are

- Tempauth : It is used mostly in development phase and it is recommended not to use it in production.
- Openstack Keystone: Keystone is the identity service used by Openstack for authentication and high-level authorization.

- **Openstack Keystone :**

It is another name for OpenStack Identity – a set of services developed for the purposes of user authentication and authorization by OpenStack. The service allows to securely check the client's identity and assign a unique access code (token) trusted by internal services. It's reliability and user-friendliness ensures the user is identified with no need to transmit its sensitive information to applications and thoroughly control client's rights in the system.

For accessing Swift, first a user auth token is generated by sending the details of the user's account to Keystone. The Keystone generates and returns the authentication token for the user. The User can access his/her account using this Auth token only.

## 6. PROJECT DETAILS

- **Objective:**

The Sunbird platform stores data on a remote centralized server. This project is about creating a storage service (A private cloud at IIT Bombay) and configuring Sunbird to use that private cloud for all data storage.

- **How Sunbird stores data currently?**

Sunbird currently uses Ekstep service for data storage. Whenever there is a request to store or retrieve data, Sunbird send a request to EkStep foundation for that. Ekstep stores data to Amazon S3 servers or retrieves from Amazon S3 servers.

- **Why do we want to use local content repository instead allowing EkStep to handle content related requests?**

By having the private cloud storage environment at the campus we can be assure the privacy of the data. We can put our own security measures and we don't need to rely on or pay for any public storage services like Amazon S3. We can also configure our storage solution as per our requirements.

- **Approach**

- 1) Installation of OpenStack Swift and OpenStack keystone:**

We first installed OpenStack Swift on a virtual machine. We tested the functionality of Swift using Keystone authentication service. But, on production scale, we cannot use tempauth for authentication. So we decided to use keystone for the authentication system. So, we installed keystone and connected it with the Swift.

- 2) Installation of Sunbird:**

We needed to install the Sunbird devops on our local server. After installation we would have Sunbird running on our local server in development mode. The installation guide for Sunbird installation is attached with appendix.

- 3) Understanding the codeflow of the Sunbird:**

In order to use Swift for the storage for the data from Sunbird, we needed to understand when does the Sunbird sends request to EkStep for the content related queries. We also needed to figure out what parameters and what other information about the content Sunbird is sending to EkStep. We also needed to figure out how does the EkStep processes the incoming request

and how it manipulates the data. We then needed to change the code where Sunbird is calling EkStep for content related queries.

#### **4) Creating the APIs for using Swift for the end user:**

We then created the APIs to access Swift (upload/download object from Swift, create/delete container from Swift atc..) using Django. This APIs would be used while we send content related request from Sunbird to Swift.

#### **5) Creating IITBx courses section on Sunbird:**

We created separate section for creating IITBx courses with custom course details form. Also we added separate IITBx course panel for showing existing IITBx courses fetched from Swift storage. Clicking on them will show details of the course. All the requests first handled by local content service running on Node.JS then redirected to django server.

## **7. IMPLEMENTATION DETAILS**

### **A. User and Organization Management APIs**

#### **1. API Description:**

Sunbird APIs are REST based (JSON over HTTPS). The REST protocol totally separates the user interface from the server and the data storage. It improves the portability of the interface to other types of platforms.

The User Management API resources are used to manage individual users and their memberships, also these resources allow you to add skills, block and unblock users, encrypt user data. You can perform various other user management related tasks.

The Organisation Management API resources perform operations related to management of an Organisation on the Sunbird Platform, to add and remove a user.

## 2. List of APIs and endpoints

### → User APIs

S.no	API	Method	URL
1.	Create User	POST	https://{Domain Name}/api/user/v1/create
2.	Search User	POST	https://{Domain Name}/api/user/v1/search
3.	Get User by Login ID	POST	https://{Domain Name}/api/user/v1/profile/read
4.	Get User by User ID	GET	https://{Domain Name}/api/user/v1/read/{User_ID}
5.	Add Users Current Login Time	PATCH	https://{Domain Name}/api/user/v1/update/logintime
6.	Get User Role Information	GET	https://{Domain Name}/api/user/v1/role/read
7.	User profile Visibility	POST	https://{Domain Name}/api/user/v1/profile/visibility
8.	Block user	POST	https://{Domain Name}/api/user/v1/block
9.	Unblock user	POST	https://{Domain Name}/api/user/v1/unblock

### → Organisation APIs

S.no	API	Method	URL
1.	Create a new Organisation	POST	https://{Domain Name}/api/org/v1/create
2.	Read Organisation details	POST	https://{Domain Name}/api/org/v1/read
3.	Search for an Organisation	POST	https://{Domain Name}/api/org/v1/search
4.	Update particulars of an existing Organisation	PATCH	https://{Domain Name}/api/org/v1/update

5.	Organisation update status	PATCH	https://{Domain Name}/api/org/v1/status/update
6.	Create new Organisation Type	POST	https://{Domain Name}/api/org/v1/type/create
7.	List all Organisation Types	GET	https://{Domain Name}/api/org/v1/type/list
8.	Update particulars of an existing Organisation Type	PATCH	https://{Domain Name}/api/org/v1/type/update
9.	Add a User to Organisation	POST	https://{Domain Name}/api/org/v1/member/add
10.	Remove a User from Organisation	POST	https://{Domain Name}/api/org/v1/member/remove

### 3. List of available roles

- **COURSE\_MENTOR** : A Course Mentor guides and instructs learners on how to undertake a course. They create batches of users to enrol for a course and ensure completion of a course within a stipulated time.
- **CONTENT\_REVIEWER** : A Content Reviewer assesses created content within prescribed guidelines.
- **TEACHER\_BADGE\_ISSUER** : A Teacher Badge Issuer is responsible for assigning badges to teachers based on the discretion of the concerned organization.
- **BOOK\_CREATOR** : A Book Creator curates and compiles books. Registered users become book creators when book creation rights are assigned to them.
- **BOOK\_REVIEWER** : A Book Reviewer assesses books within defined and prescribed guidelines.
- **PUBLIC** : A Public user is any user with registered credentials.
- **ANNOUNCEMENT\_SENDER** : An Announcement Sender creates and sends announcements. The Announcement feature allows to send system wide messages to a target audience.

- **CONTENT\_CREATOR** : A Content Creator is a registered user with permission to create content.
- **FLAG\_REVIEWER** : A Flag Reviewer assesses flagged content within prescribed guidelines.
- **ADMIN** : An admin user can upload organisation,upload users and remove users,assign rights to users,assign badges to users.
- **ORG\_ADMIN** : A org admin is a user who can upload users to an organisation,delete users,assign rights to users,assign badges to users.

#### 4. Pre-requisites

- **Authentication**

- Bearer  
Security scheme type : API Key  
header parameter name : Authorization
- userToken  
Security scheme type : API Key  
header parameter name : x-authenticated-user-token

- **Header Parameters for API(s)**

- Content-Type : the media type of the resource in body.  
  
(Application/json,Multipart/form-data,Application/x-www-form-urlencoded)
- Authorization : To make use of any User API, authorization is required.
- ts : Timestamp is a sequence of characters or encoded information identifying  
when the API call occurred, usually it gives date and time of day
- x-authenticated-user-token : It a unique token/key to authenticate the user each  
time an API is called.

- **Types of Response Codes:**

- **200 OK.**

Successful operation. The api operation was successfully executed.

- **400 'CLIENT\_ERROR.'**

The requested operation failed due to bad request from client. Possible reasons for failure: mandatory values might be missing or due to incorrect values.'

- 500 'INTERNAL SERVER ERROR.  
The requested operation failed due to a server error.'

## 5. Usage:

- **Create a new Organisation as a rootOrg :** This API is for creation of a new Organisation on the Sunbird Platform.

```

URL      : 10.129.103.85/api/org/v1/create
Method   : POST
Request Body :
{
  "id": " ",
  "ver": " ",
  "ets": 0,
  "params": {
    "msgid": " ",
    "resmsgid": " ",
    "status": "success"
  },
  "request": {
    "orgName": " ",
    "isRootOrg": true
  }
}

```

Fig. 7.1.1 : Request Description for creating a new Organisation

- **Add a User to Organisation :** This API is for adding a user to an existing organisation on the Sunbird Platform.

```

URL      : 10.129.103.85/api/org/v1/member/add
Method   : POST
Request Body :
{
  "request": {
    "organisationId": "0125329928463237123",
    "roles": [
      "COURSE_CREATOR"
    ],
    "userId": "de4895b1-5db8-43f3-b480-9df4bab4e6be"
  }
}

```

Fig. 7.1.2 : Request Description for adding a User to a Organisation



When the user api's and the organisation api's are used, the data sent by POST method is saved in the Cassandra database and the response for the GET method is retrieved from the database.

## 6. Schemas of user, organisation and user\_org tables

### A. User:

S.no	Attributes	Data Type
1.	id [Primary Key]	text
2.	avatar	text
3.	countrycode	text
4.	createdby	text
5.	createddate	text
6.	currentlogintime	text
7.	dob	text
8.	email	text
9.	emailverified	boolean
10.	firstname	text
11.	gender	text
12.	grade	list<text>
13.	isdeleted	boolean
14.	language	list<text>
15.	lastlogintime	text
16.	lastname	text
17.	location	text
18.	loginid	text

19.	password	text
20.	phone	text
21.	profilessummary	text
22.	profilevisibility	map<text,text>
23.	provider	text
24.	regorgid	text
25.	roles	list<text>
26.	rootorgid	text
27.	status	int
28.	subject	list<text>
29.	tcstatus	text
30.	tcupdateddate	text
31.	temppassword	text
32.	thumbnail	text
33.	updatedby	text
34.	updateddate	text
35.	userid	text
36.	username	text

**B. Organisation:**

S.no	Attributes	Data Type
1.	id [Primary Key]	text
2.	addressid	text
3.	approvedby	text

4.	approveddate	text
5.	channel	text
6.	communityid	text
7.	contactdetail	text
8.	createdby	text
9.	createddate	text
10.	datetime	timestamp
11.	description	text
12.	externalid	text
13.	hashtagid	text
14.	homeurl	text
15.	imgurl	text
16.	isapproved	boolean
17.	isdefault	boolean
18.	isrootorg	boolean
19.	locationid	text
20.	noofmembers	int
21.	orgcode	text
22.	orgname	text
23.	orgtype	text
24.	orgtypeid	text
25.	parentorgid	text
26.	prefferedlanguage	text

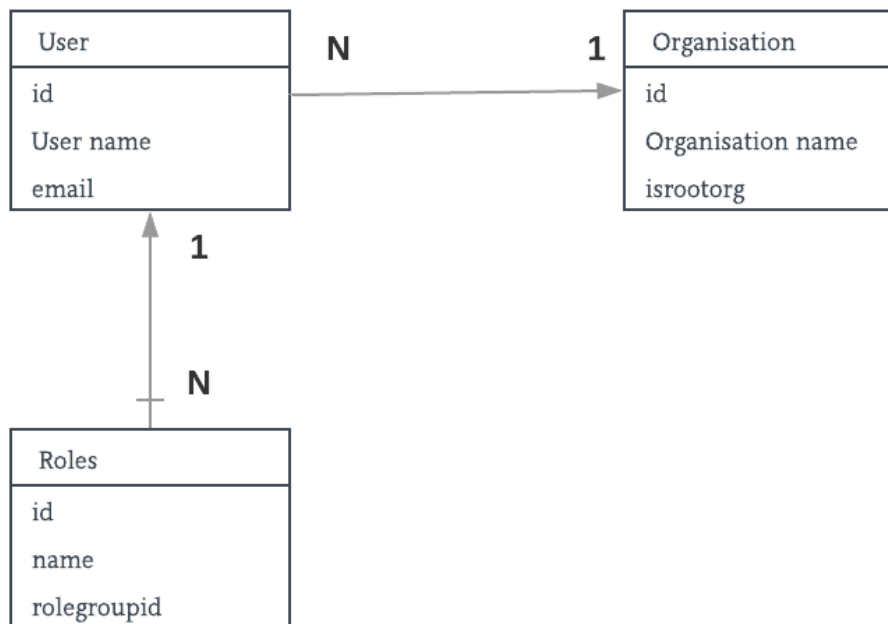
27.	provider	text
28.	rootorgid	text
29.	slug	text
30.	status	int
31.	theme	text
32.	thumbnail	text
33.	updatedby	text
34.	updateddate	text

**C. user\_org :** to store users assigned to a organisation with roles.

S.no	Attributes	Data Type
1.	id [Primary Key]	text
2.	addedby	text
3.	addedbyname	text
4.	approvaldate	text
5.	approvedby	text
6.	isapproved	boolean
7.	isdeleted	boolean
8.	isrejected	boolean
9.	organisationid	text
10.	orgjoindate	text
11.	orgleftdate	text
12.	position	text
13.	roles	list<text>

14.	updatedby	text
15.	updateddate	text
16.	userid	text

Cassandra Query Language(CQL) does not have a concept of foreign keys or any kind of constraints between tables. So, the constraints needed between the tables are handled in the code. While assigning a user to an organisation by giving roles to the user, this will be stored in a another schema “user\_org”. A Organisation can have many users and a user can have many roles.



**Fig. 7.1.3 : Entity-Relationship Diagram in Cassandra Query Language**

## B. Encapsulating Swift APIs in Django APIs

### 1. Introduction

OpenStack Swift APIs allow us to perform the CRUD (create, retrieve, update, delete) operations on the object store. These APIs are implemented as a set of Representational State Transfer (REST) web services. The Swift APIs were encapsulated in Django REST APIs to hide the technical details and the physical location of the object store. Applications such as Sunbird can use these Django REST APIs to store/retrieve their data onto cloud.

To hide the technical requirements and implementations of the Swift APIs from the business users, REST APIs created through Django-REST Framework act as an interface between Swift Object Storage and the End Users. The APIs process the data and parameters provided by the user and in turn forward the user requests to corresponding endpoints of the cloud storage platform. The details of the cloud platform as transparent to the user and the implementation of the API can change without impacting the user applications.

As an instance, these APIs provide smooth access to object storage for registered users without requiring them to authenticate by specifically generating authorization tokens.

The API is typically structured as:

**/v1/{customer organization or project}/{end user}/{document type}**

Where **{customer organization or project}** means an area allocated to each user who desires to be located in their own private space, **{course}** is the kind of similar data that the customer wishes to store.

In a learning context, **{customer organization or project}** means an institute implementing Sunbird, **{end user}** is a course and **{document type}** is a content object(such as a reference material or notes).

An example of how this API can be used in a context other than learning is a banking solution. In that case, **customer organization** would be the bank, **end user** would be an individual account holder and course content would be the type of document being stored eg. PAN card.

### 2. API Endpoints

- Display List of containers **GET**

```
curl -i <server ip>:<port no.>/files/info/
```

Eg:- curl -i 10.196.0.56:8000/files/info/

- **Display List of objects inside a container GET**

```
curl -i <server ip>:<port no.>/files/info/<container>
```

Eg:- curl -i 10.196.0.56:8000/files/info/marktwain

- **Create new container PUT**

```
curl -i <server ip>:<port no.>/files/info/ -X PUT -d '{"name":"<container-name>"}
```

Eg:- curl -i 10.196.0.56:8000/files/info/ -X PUT -d '{"name":"new\_cont"}'

- **View container metadata GET**

```
curl -i <server ip>:<port no.>/files/info/<container>/metadata -X POST -d
```

```
'{"X-Remove-Container-Meta-<field name>":"x"}'
```

Eg:- curl -i 10.196.0.56:8000/files/info/swift-container1/metadata -X GET

- **Add/ Update container metadata POST**

```
curl -i <server ip>:<port no.>/files/info/<container> -X POST -d
```

```
'{"X-Container-Meta-<field name>":"<value of the field>"}
```

Eg:- curl -i 10.196.0.56:8000/files/info/swift-container1 -X POST -d

```
'{"X-Container-Meta-Example":"Demo"}'
```

- **Remove container metadata POST**

```
curl -i <server ip>:<port no.>/files/info/<container> -X POST -d
```

```
'{"X-Remove-Container-Meta-<field name>":"x"}'
```

Eg:- curl -i 10.196.0.56:8000/files/info/swift-container1 -X POST -d

```
'{"X-Remove-Container-Meta-Example":"x"}'
```

- **Delete container DELETE**

```
curl -i <server ip>:<port no.>/files/info/<container> -X DELETE
```

Eg:- curl -i 10.196.0.56:8000/files/info/marktwain -X DELETE

Note: A container which is not empty cannot be deleted.

- **Upload a new Object PUT**

```
curl --form file=@<path/to/file> <server ip>:<port no.>/files/info/<container>/upload
```

Eg:- curl --form file=@desktop/inscription.pdf 10.196.0.56:8000/files/info/cont/upload

Note: Enter the path relative to the current directory.

- **Download object GET**

curl -i <server ip>:<port no.>/files/info/<container>/<object>/download -o "<file.ext>"

Eg:- curl -i 10.196.0.56:8000/files/info/marktwain/article.pdf/download -o "article.pdf"

Note: The object will be downloaded in the current directory where curl is looking into.

- **View object metadata HEAD**

curl -i <server ip>:<port no.>/files/info/<container>/<object>

Eg:- curl -i 10.196.0.56:8000/files/info/marktwain/article.pdf

- **Add/ Update object metadata POST**

curl -i <server ip>:<port no.>/files/info/<container>/<object> -X POST -d

'{"X-Object-Meta-<field name>":"<value of the field>"}'

Eg:- curl -i 10.196.0.56:8000/files/info/swift-container1/swiftfile.txt -X POST -d

'{"X-Object-Meta-Name":"swiftfile"}'

- **Remove object metadata POST**

curl -i <server ip>:<port no.>/files/info/<container>/<object> -X POST -d

'{"X-Remove-Object-Meta-<field name>":"x"}'

Eg:- curl -i 10.196.0.56:8000/files/info/swift-container1/swiftfile.txt -X POST -d

'{"X-Remove-Object-Meta-Name":"x"}'

- **Delete object DELETE**

curl -i <server ip>:<port no.>/files/info/<container>/<object> -X DELETE

Eg:- curl -i 10.196.0.56:8000/files/info/marktwain/birthday.png -X DELETE



The following APIs have been created :

- **Retrieve Containers** : It sends a GET request to the swift APIs and returns the list of containers to the caller.

```
$ curl -i 192.168.0.102:8000/files/info/
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100  147  100  147    0    0   147      0  0:00:01  0:00:01 --:--:-- 123HT
TP/1.1 200 OK
Date: Tue, 03 Jul 2018 08:15:06 GMT
Server: WSGIServer/0.2 CPython/3.6.5
Content-Type: application/json
Vary: Accept, Cookie
Allow: OPTIONS, GET, PUT
X-Frame-Options: SAMEORIGIN
Content-Length: 147

"Dusra_Course\nDusra_course\nNaya_Course\nTanmoy\ncontainer5\nmarktwain\nnew_con
t\nnew_container\nnew_course\nswift-container1\nswift-container2\n"
```

Fig. 7.1.4 : Swift API : Retrieve container

- **Create a New Container** : It receives the new container name as parameter and sends a PUT request to the corresponding swift API.

```
$ curl -i 192.168.0.102:8000/files/info/ -X PUT -d '{"name":"new_container"}'
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100  102  100   78  100   24    78      24  0:00:01  0:00:01 --:--:-- 89H
TTP/1.1 200 OK
Date: Tue, 03 Jul 2018 08:31:25 GMT
Server: WSGIServer/0.2 CPython/3.6.5
Content-Type: application/json
Vary: Accept, Cookie
Allow: GET, OPTIONS, PUT
X-Frame-Options: SAMEORIGIN
Content-Length: 78
```

Fig. 7.1.5 : Swift API : Create new container

- **Update Container Metadata** : It receives the container name as parameter and sends a POST request to swift API to add, update or delete container metadata depending on the request.

```
$ curl -i 192.168.0.102:8000/files/info/swift-container1 -X POST -d '{"X-Container-Meta-Example":"Demo"}'
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100	542	100	507	100	35	169	11
				0:00:03	0:00:03	--:--:--	136H

```
TP/1.1 200 OK
Date: Tue, 03 Jul 2018 08:28:39 GMT
Server: WSGIServer/0.2 CPython/3.6.5
Content-Type: application/json
Vary: Accept, Cookie
Allow: GET, OPTIONS, DELETE, POST
X-Frame-Options: SAMEORIGIN
Content-Length: 507

{"X-Container-Object-Count":"5","Accept-Ranges":"bytes","X-Storage-Policy":"gold","Last-Modified":"Tue, 03 Jul 2018 08:27:57 GMT","X-Container-Meta-Author":"SS","X-Container-Bytes-Used":"31104678","X-Timestamp":"1528451672.10720","Content-Type":"text/plain; charset=utf-8","X-Container-Meta-Example":"Demo","Content-Length":"119","X-Trans-Id":"tx90aedc3ee5844b5c87a4b-005b3b338c","X-Openstack-Request-Id":"tx90aedc3ee5844b5c87a4b-005b3b338c","Date":"Tue, 03 Jul 2018 08:27:56 GMT","Connection":"keep-alive"}
```

Fig. 7.1.6 : Swift API : Update container metadata

- **View Container Metadata** : It receives the name of the container as parameter and sends a HEAD request to swift API and returns the container metadata as response.

```
$ curl -i 192.168.0.102:8000/files/info/swift-container1/metadata -X GET
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100	507	100	507	0	0	507	0
				0:00:01	--:--:--	0:00:01	541H

```
TTP/1.1 200 OK
Date: Tue, 03 Jul 2018 08:35:32 GMT
Server: WSGIServer/0.2 CPython/3.6.5
Content-Type: application/json
Vary: Accept, Cookie
Allow: GET, OPTIONS
X-Frame-Options: SAMEORIGIN
Content-Length: 507

{"X-Container-Object-Count":"5","Accept-Ranges":"bytes","X-Storage-Policy":"gold","Last-Modified":"Tue, 03 Jul 2018 08:27:57 GMT","X-Container-Meta-Author":"SS","X-Container-Bytes-Used":"31104678","X-Timestamp":"1528451672.10720","Content-Type":"text/plain; charset=utf-8","X-Container-Meta-Example":"Demo","Content-Length":"119","X-Trans-Id":"tx2c5517d653104105acad1-005b3b3529","X-Openstack-Request-Id":"tx2c5517d653104105acad1-005b3b3529","Date":"Tue, 03 Jul 2018 08:34:49 GMT","Connection":"keep-alive"}
```

Fig. 7.1.7 : Swift API : View container metadata

- **Delete an empty Container** : It receives the name of the container to be deleted as parameter and sends a DELETE request to swift API and removes the container from the account.

```
$ curl -i 192.168.0.102:8000/files/info/new_container -X DELETE
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100    32  100    32    0    0    32    0  0:00:01  0:00:01 --:--:--  22H
HTTP/1.1 200 OK
Date: Tue, 03 Jul 2018 08:41:12 GMT
Server: WSGIServer/0.2 CPython/3.6.5
Content-Type: application/json
Vary: Accept, Cookie
Allow: POST, DELETE, GET, OPTIONS
X-Frame-Options: SAMEORIGIN
Content-Length: 32

"Successfully Deleted Container"
```

Fig. 7.1.8 : Swift API : Delete empty container

- **Retrieve Objects in a Container** : It receives the name of the container as parameter and sends a GET request to swift API and returns the list of objects inside that container as response.

```
$ curl -i 192.168.0.102:8000/files/info/marktwain
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100   428  100   428    0    0   428    0  0:00:01  0:00:01 --:--:--  234H
HTTP/1.1 200 OK
Date: Tue, 03 Jul 2018 08:43:25 GMT
Server: WSGIServer/0.2 CPython/3.6.5
Content-Type: application/json
Vary: Accept, Cookie
Allow: POST, DELETE, GET, OPTIONS
X-Frame-Options: SAMEORIGIN
Content-Length: 428

"123.png\nAnsible.pdf\nFriends.mp4\nIITs.png\nLec-7.3.pdf\nScreenshot_123.png\nSwiftAPIDocumentation.pdf\nZehnaseeb.mp3\nabcd.json\narticle_cPIe2tR.pdf\nbadri test123.txt\nnd123.png\nfirst.pdf\nfriday.PNG\nng1.pdf\nhelloworld.png\nif.txt\nimage.png\nimage_H0oyI3D.png\nimage_RTXXgkg.png\nindex.js\njavascript_tutorial.pdf\nlab.pdf\nlast1.png\nmanual.pdf\nmaths.pdf\nnodejs.txt\nnotebook1.png\nsend.txt\ntesttttt1_euYT7Yp.txt\n"
```

Fig. 7.1.9 : Swift API : Retrieve objects in a container

- **Create a New Object in a Container** : It receives the container name and the new object name as parameters and sends a PUT request to the corresponding swift API. The new object is then added to the container list.

```
$ curl --form file=@Ansible.pdf 192.168.0.102:8000/files/info/container5/upload
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 503k  100  23 100 503k    25  555k --:--:-- --:--:-- --:--:--  555k"
Successfully Uploaded"
```

Fig. 7.1.10 : Swift API : Create new object in container



- **Download Object** : It receives the name of the container and the name of object to be downloaded as parameters and sends a GET request to swift API. It fetches the content of object and makes it available for viewing or downloading to the user.

```
$ curl -i 192.168.0.102:8000/files/info/container5/Ansible.pdf/download -o "Ansible.pdf"
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100    70    100    70    0    0    41      0  0:00:01  0:00:01 --:--:--   41
```

Fig. 7.1.11 : Swift API : Download Object

- **View Object Metadata** : It receives the container name and the object name as parameters and sends a HEAD request to swift API and returns the object metadata as response.

```
$ curl -i 192.168.0.102:8000/files/info/container5/Ansible.pdf
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100    242    100    242    0    0   281      0  --:--:-- --:--:-- --:--:--  281HTTP/1.1 200 OK
Date: Tue, 03 Jul 2018 08:47:49 GMT
Server: WSGIServer/0.2 CPython/3.6.5
Content-Type: application/json
Vary: Accept, Cookie
Allow: DELETE, POST, GET, OPTIONS
X-Frame-Options: SAMEORIGIN
Content-Length: 242

{"Content-Length":"70","Content-Type":"text/html; charset=UTF-8","X-Trans-Id":"tx22e7aaf101b449f9be979-005b3b380a","X-Openstack-Request-Id":"tx22e7aaf101b449f9be979-005b3b380a","Date":"Tue, 03 Jul 2018 08:47:06 GMT","Connection":"keep-alive"}
```

Fig. 7.1.12 : Swift API : View object metadata

- **Update(Add/Remove) Object Metadata** : It receives the container and object name as parameter and sends a POST request to swift API to add, update or delete object metadata depending on the request.

```
$ curl -i 192.168.0.102:8000/files/info/swift-container1/swiftfile.txt -X POST -d '{"X-Remove-Object-Meta-Name":"x"}'
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100    408    100    375    0    0   375      0  0:00:01  0:00:01 --:--:--   375HTTP/1.1 200 OK
Date: Tue, 03 Jul 2018 08:51:44 GMT
Server: WSGIServer/0.2 CPython/3.6.5
Content-Type: application/json
Vary: Accept, Cookie
Allow: DELETE, POST, GET, OPTIONS
X-Frame-Options: SAMEORIGIN
Content-Length: 375

{"Content-Length":"21","Accept-Ranges":"bytes","Last-Modified":"Tue, 03 Jul 2018 08:51:02 GMT","Etag":"2261f16a802e7a91e18e73d1edc3128a","X-Timestamp":"1530607861.65365","Content-Type":"text/plain","X-Trans-Id":"tx781847fd858a4eb9bb604-005b3b38f5","X-Openstack-Request-Id":"tx781847fd858a4eb9bb604-005b3b38f5","Date":"Tue, 03 Jul 2018 08:51:01 GMT","Connection":"keep-alive"}
```

Fig. 7.1.13 : Swift API : Update object metadata

- **Delete Object** : It receives the name of the container and the name of object to be deleted as parameters and sends a DELETE request to swift API and removes the object from the container.

```
$ curl -i 192.168.0.102:8000/files/info/container5/Ansible.pdf -X DELETE
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             Dload  Upload  Total  Spent    Left     Speed
100    29    100    29    0    0    39      0  --:--:--  --:--:--  --:--:--   39HTTP/1.1 200 OK
Date: Tue, 03 Jul 2018 08:41:34 GMT
Server: WSGIServer/0.2 CPython/3.6.5
Content-Type: application/json
Vary: Accept, Cookie
Allow: POST, DELETE, GET, OPTIONS
X-Frame-Options: SAMEORIGIN
Content-Length: 29

"Successfully Deleted Object"
```

**Fig. 7.1.14 : Swift API : Delete Object**

## C. Understanding the Content Flow

Course is the sequence of contents which is created by a user of an organization who is assigned the role of CONTENT\_CREATOR and reviewed, published by the user who is assigned the role of CONTENT\_REVIEWER. Contents may be **Courses, Books, Resource, Collection or Lesson Plan**.

### 1. Use Case Diagram for Course Creation

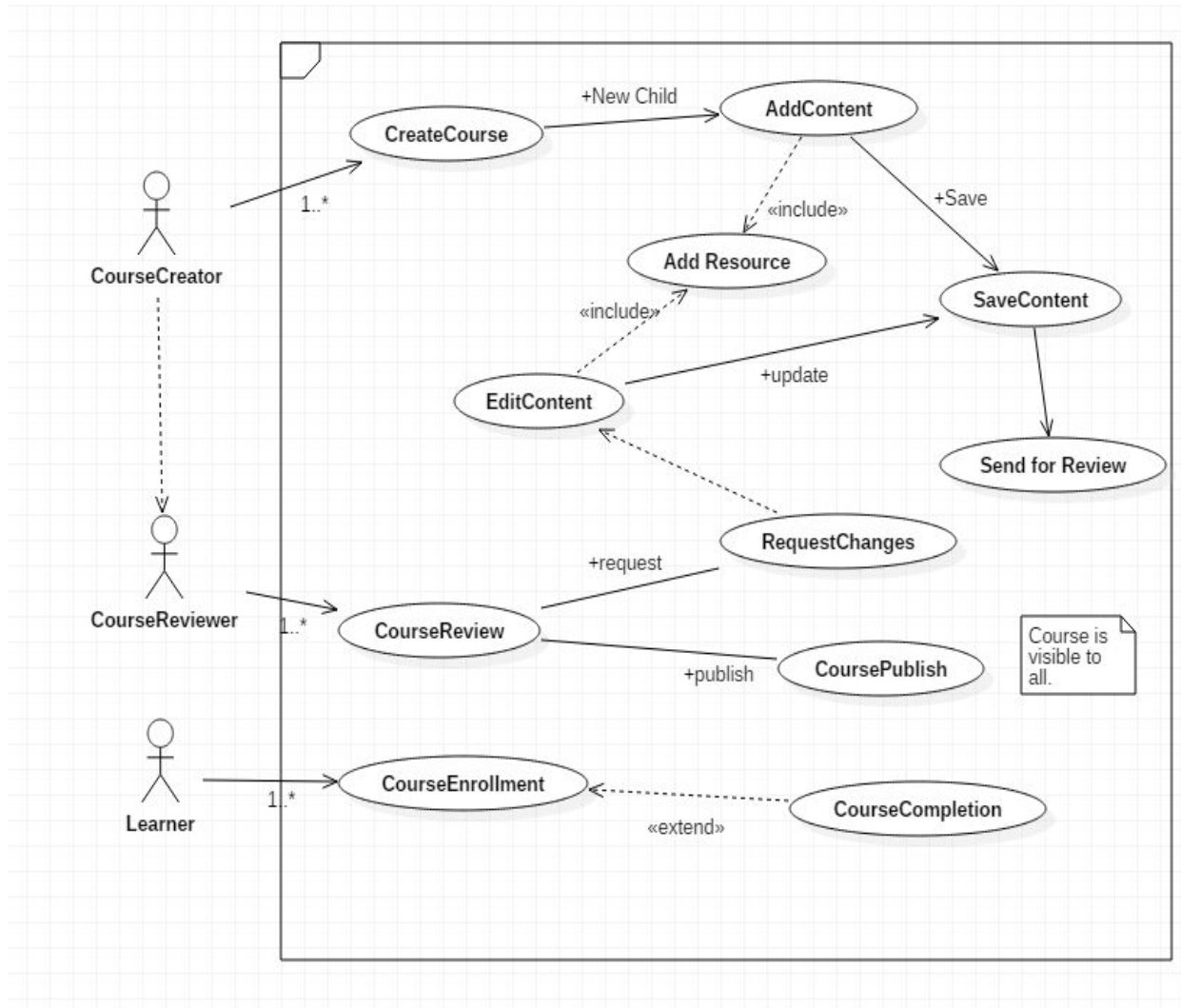


Fig. 7.1.15 : Use Case Diagram for Course Creation

## 2. Implementation Details

- **Course Creation:**

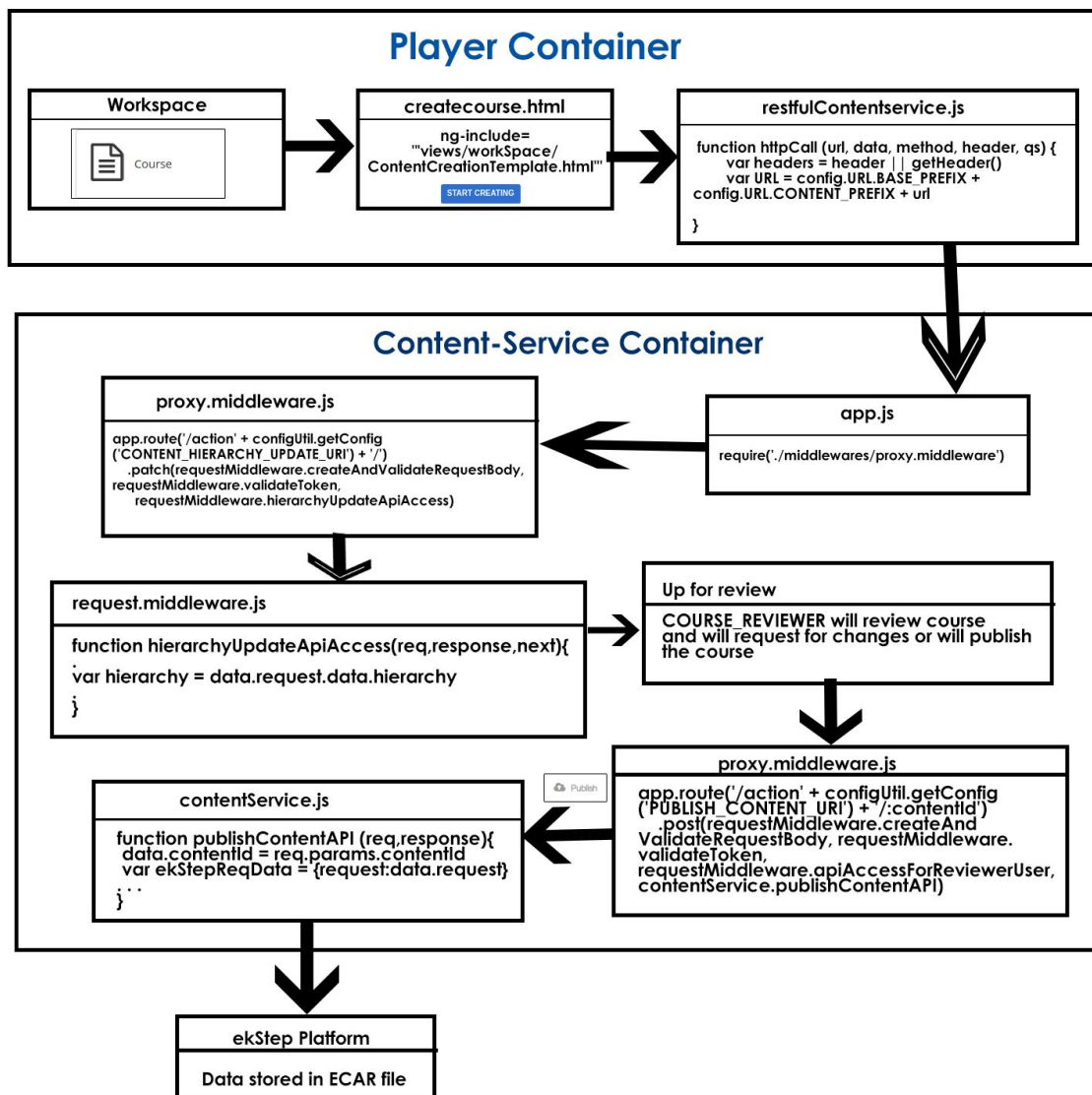


Fig. 7.1.16 : Data Flow Diagram for Course Creation

- **Types of Content**

- **Course:** It consists of contents in a hierarchy. It contains units and further, contains subunits.
- **Course Unit:** It is the content of the course. It may have siblings and childs.
- **Resource:** It is the simplest unit of content that can be created by any user. The various categories of resources that can be created are Experiment, Play, Learn etc.

- **Collection:** It is a compilation of content. Related or unrelated content can be compiled together.
- **Lesson Plan:** It is a structured outline of a given topic, unit, chapter to be taught on any given day. It aids to streamline teaching and create an engaging learning experience.
- **Book:** It is a collection of content units. It can be curated from syllabus books or any other book.

- **Course Data structures:**

The course details are stored in EkStep platform in ECAR file format. ECAR refers to EkStep Content Archive. The EkStep Content Archive is a file format for side-loading multiple content items. The format is a compressed zip structure, with a manifest that describes the details of content included in the archive. The metadata in the manifest, and resources such as icons and poster images are interpreted.

→ **Extension:** .ecar

→ **MIME Type:** application/vnd.ekstep.content-collection

A zip archive, containing the manifest.json in the root folder. All other resources such as the images and content archives are included in the zip. The manifest has the relative path to the resources. The manifest is a JSON file, containing the metadata about content items included in the archive.

```
{
  "id": "ekstep.content.archive",
  "ver": "1.1", // Packaging & metadata format version
  "ts": "2015-12-29T02:13:50ZZ", // When package was created
  "params": {
    "resmsgid": "eb1e9d98-b1c9-4db6-b044-e58d9c12fcba", // Package generation id
  },
  "archive": {
    "count": 5, // No of items in the package
    "items": [
      {
        "identifier": "", // System identifier for the content
        "name": "", // Title of the content (can be unicode for non-english titles)
        "code": "org.ekstep.delta", // System unique code for the content
        "description": "", // Text description
        "pkgVersion": "", // Version number of the content
        "mimeType": "application/vnd.android.package-archive", // Mime type (APK, ECML, Zipped HTML etc)
        "minSupportedVersion": "4.1" // Min supported version - anything older should be discarded
        // Rendering metadata
        "downloadUrl": "", // Relative path to the zip/apk within ecar
        "grayScaleAppIcon": "", // Relative image path within ecar
        "posterImage": "", // Relative image path within ecar (optional)
        "communication_scheme": "FILE", // Deprecated
        "launchUrl": "", // URL used by Genie to launch the content
        "activity_class": "", // Name of the main activity class to launch
        "osId": "", // Package name for the operating system
      }
    ]
  }
}
```

Fig.

### 7.1.17 :EkStep Content Archive (ECAR) file format Manifest



### 3. Use of Sunbird Content APIs

- **Create Content API Details:**

Following Api is called when user with role Content\_Creator starts creating content of a course:

```
URL : http://10.129.103.85/private/service/v1/content/content/v1/create
Method : POST
Request Body :{
  request:
    { content:
      {
        name: 'Untitled Course',
        mimeType: 'application/vnd.ekstep.content-collection',
        createdBy: 'bd73c8db-0450-4279-9e69-944ae20de80a',
        contentType: 'Course',
        resourceType: 'Course',
        organization: [],
        createdFor: [Object],
        creator: 'content_creator1 '
      }
    },
  ts: 2018-06-25T12:13:21.319Z,
  url: '/v1/content/create',
  path: '/v1/content/create',
  params: { msgid: '26d8f370-7871-11e8-b7c7-95537a0d3b69' }
}
```

Fig. 7.1.18 : Request description for creating content

- **Hierarchy of Content API Details:**

Hierarchy is the object that contains all the details of the course and their content.

```
{
  do_2125331769701744641893:
    {
      name: 'Poems',
      contentType: 'Course',
      children: [ 'c6b3903b-548e-4bc5-a87b-458cde565d06' ],
      root: true
    },
  'c6b3903b-548e-4bc5-a87b-458cde565d06':
    {
      name: 'Twinkle Twinkle Little Star',
      contentType: 'CourseUnit',
      children: [ 'do_2124786923763875841756' ],
      root: false
    },
  do_2124786923763875841756:
    {
      name: '1',
      contentType: 'Collection',
      children: [],
      root: false
    }
}
```

Fig. 7.1.19 : Hierarchy of content API details

- **Publish Content API Details:**

When user with Reviewer role click on publish button, the following API is called:

```
URL : http://10.129.103.85/action/content/v3/publish/<content_id>
Method : POST
request:
{
  content:
  {
    lastPublishedBy: '70f34f3b-6a6f-4e48-8c43-23d5e10c01f4',
    publishChecklist: [Object],
    publishComment: ''
  },
  ts: 2018-06-25T12:23:23.817Z,
  url: '/action/content/v3/publish/do_2125331769701744641893',
  path: '/action/content/v3/publish/:contentId',
  params: { msgid: '8df6d990-7872-11e8-b7c7-95537a0d3b69' },
  contentId: 'do_2125331769701744641893'
```

**Fig. 7.1.20 : Request description for publishing content**

## D. Distributing Django Cloud Storage APIs

In order to distribute the Django Cloud Storage APIs, there were a variety of options available. However, containerization technology helps to make it easier to distribute applications. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

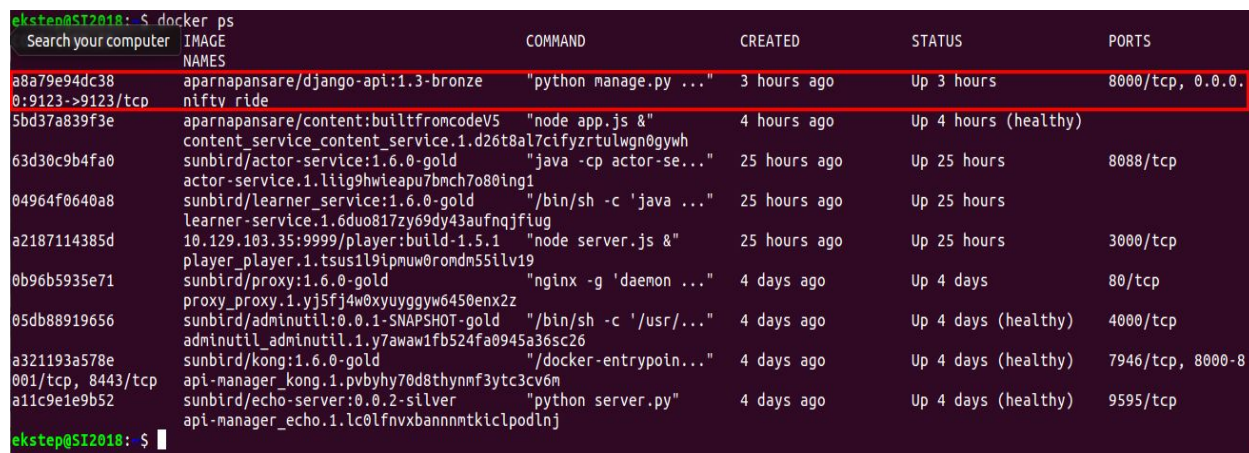
The Sunbird platform also uses containerization very extensively. In particular, it uses the Docker technology for distributing most of its software components. Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.

Docker gives you the capability to create your own Docker images which can be distributed as required.

Docker images can be created with the help of DockerFile. A DockerFile is a simple text file with instructions on how to build your images.

A DockerFile was created for the Django Cloud Storage APIs. The base Docker image was Django:python3-onbuild. Various modules had to be installed as part of the image creation. All these were added in requirements.txt. The source code of the Django application was copied. Using all these files, a docker image was created.

After creating the docker image, the image is running as a container and can be seen as shown in screenshot.



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
a8a79e94dc38	aparnapansare/django-api:1.3-bronze	"python manage.py ..."	3 hours ago	Up 3 hours	8000/tcp, 0.0.0.
0:9123->9123/tcp	niftv ride				
5bd37a839f3e	aparnapansare/content:builfromcodeV5	"node app.js &"	4 hours ago	Up 4 hours (healthy)	
63d30c9b4fa0	content_service_content_service.1.d26t8a17cifyzrtulwgn0gywh	"java -cp actor-se..."	25 hours ago	Up 25 hours	8088/tcp
04964f0640a8	sunbird/actor-service:1.6.0-gold	"bin/sh -c 'java ..."	25 hours ago	Up 25 hours	
a2187114385d	sunbird/learner_service:1.6.0-gold	"node server.js &"	25 hours ago	Up 25 hours	3000/tcp
0b96b5935e71	learner-service.1.6duo817zy69dy43aufnqjfiug	"nginx -g 'daemon ..."	4 days ago	Up 4 days	80/tcp
05db88919656	10.129.103.35:9999/player:build-1.5.1	"/bin/sh -c '/usr/..."	4 days ago	Up 4 days (healthy)	4000/tcp
a321193a578e	player_player.1.tsus19ipmw0rondm55ilv19	"/docker-entrypoin..."	4 days ago	Up 4 days (healthy)	7946/tcp, 8000-8
001/tcp, 8443/tcp	sunbird/kong:1.6.0-gold	"python server.py"	4 days ago	Up 4 days (healthy)	9595/tcp
a11c9e1e9b52	api-manager_kong.1.pvbyhy70d8thynmf3ytc3cv6m				
	sunbird/echo-server:0.0.2-silver				
	api-manager_echo.1.lc0lfnvxbannmtkiclpodlnj				

Fig.

### 7.1.21 : Terminal showing the newly created docker container for Django server

The running Django application can be checked in your browser by accessing localhost:9123.

## E. Kong API: Onboarding APIs on Sunbird Kong API container

### • Implementation Steps

Onboarding APIs in API manager is automated via ansible role kong-api. List of APIs to be on-boarded is mentioned in kong\_apis variable in ansible/roles/kong-api/defaults/main.yml

The steps to onboard an API to kong are:

- 1) Add a new entry for the new API in kong\_apis variable in **sunbird-devops/ansible/roles/kong-api/defaults/main.yml**. You can copy one of the existing API entry and change values for your API. An existing entry is of the form:

```

name: swiftAPIs
request_path: '/swiftapis'
upstream_url: '10.129.103.85:9123/files/info'
strip_request_path: true
plugins:
- name: jwt
- name: cors
- '{{ statsd_plugin }}'
- name: acl
config.whitelist:
- name: rate-limiting
config.hour: '{{ medium_rate_limit_per_hour }}'
config.limit_by: credential
- name: request-size-limiting
config.allowed_payload_size: '{{ small_request_size_limit }}'

```

Fig. 7.1.22 : Example of an entry in main.yml (ansible) file for onboarding API to kong

Here the upstream\_url is the endpoint of our API service that we want to add to kong.

2) Now run the command `./sunbird_install.sh -s apis` from the sunbird-devops/deploy directory. Now the API will be onboarded and the same can be viewed at **10.129.103.85:8001/apis/** and the routed url will now be **10.129.103.85:8000/{request\_path}**

- **Snapshots**

By visiting `10.129.103.85:8001/apis/` (Kong Admin API Interface) , we can see our API onboarded using ansible.

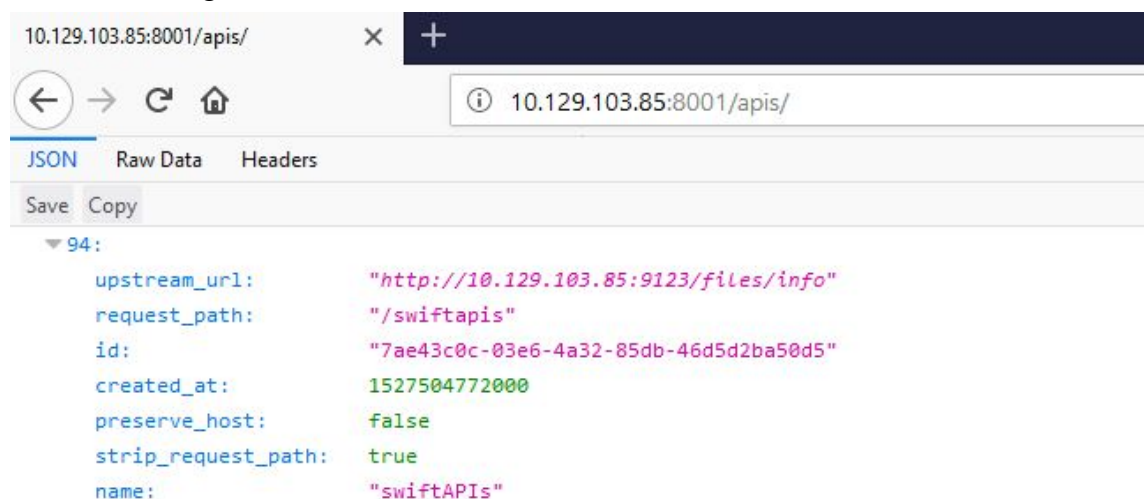


Fig. 7.1.23 : Showing newly onboarded API in Kong API list

## F. New Feature: IITBX Course on Sunbird

As we know that Sunbird fetches course data from EkStep platform. But now we need our own storage system built with Openstack cloud management system. So, As a modification, we have chapter we have added one more panel in course tab showing IITBx courses which are fetched from Swift storage.



Fig. 7.1.24 : IITBX courses panel view in Sunbird portal

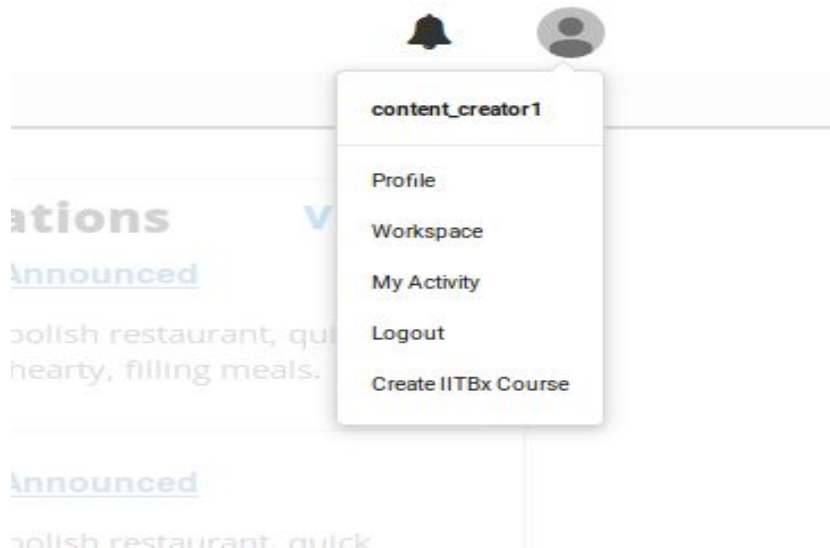
Now clicking on any course will show the details about that course which are again fetched from Swift storage with different API. For showing details of any IITBx course we have created separate section.



Fig. 7.1.25 : IITBX course detail view



We have also created the option for creating IITBx Course and access of creating course is only given to the person whose role is CONTENT\_CREATOR.



**Fig. 7.1.26 : Menu option for creating IITBX course**

A screenshot of a 'Create Course' form. The form has a light gray background. At the top, it says 'Create Course'. Below that, there are three main sections: 'Course Title' with a text input field, 'Description' with a larger text area, and 'Topics' with a list of topics. The first topic is 'computer.png' with a red 'remove' button next to it. Below the topics, there is a green '+Add' button and a blue 'Create' button.

**Fig. 7.1.27 : Create course form**

Any IITBx Course related request goes to the local content service first where the request is handled with nodeJS.

We have added basic routings for our api endpoints in the file `courseRoutes.js` in the directory `routes` inside the content service container, and have defined the functions as services inside file `iitbx.js` in directory `Services`.

**The API'S used to fulfil the tasks :-**

**1. getCoursesAPI :**

```
URL : http://10.129.103.85:5000/v1/course/iitbx/view
METHOD : GET
RESPONSE : {
  < List of Courses present in the Swift Storage >
} ( status:200 )
```

**Fig. 7.1.28 : Request Description for getting IITBX courses list**

**2. getObjectsAPI :**

```
URL : http://10.129.103.85:5000/v1/course/iitbx/<course_name>/view
METHOD : GET
RESPONSE : {
  description : < Course Details >
  objects : < List of Objects inside the Course >
} ( status:200 )
```

**Fig. 7.1.29 : Request Description for getting course details for an IITBX course**

**3. getParticularObjectAPI :**

```
URL : http://10.129.103.85:5000/v1/course/iitbx/<course_name>/<obj_name>/view
METHOD : GET
RESPONSE : Open the Object ( status:200 )
```

**Fig. 7.1.30 : Request Description for getting an object of an IITBX course**

**4. deleteParticularObjectAPI :**

```
URL : http://10.129.103.85:5000/v1/course/iitbx/<course_name>/<obj_name>/delete
METHOD : GET
RESPONSE : < obj_name > Deleted ( status:200 )
```

**Fig. 7.1.31 : Request Description for deleting an object of an IITBX course**

## 5. createCourseAPI :

```
URL : http://10.129.103.85:5000/v1/course/iitbx/createcourse
METHOD : POST
REQUEST TYPE : multipart/formdata
REQUEST BODY : {
  'title' : < Name of the Course >
  'description' : < Description of the Course >
  'files' : < List of files are uploaded for the course >
}
RESPONSE : Course Created ( status:200 )
```

Fig. 7.1.32 : Request Description for creating an IITBX course

## 6. deleteCourseAPI :

```
URL : http://10.129.103.85:5000/v1/course/iitbx/<course_name>/delete
METHOD : GET
RESPONSE : < obj_name > Deleted ( status:200 ) { Only if the Course is empty }
```

Fig. 7.1.33 : Request Description for deleting an IITBX course

## G. Storing Content to Swift

After understanding the content flow of Sunbird, we also integrated Swift with Sunbird to store the content metadata. When course is created, we called the function created by us to store that course's meta to a Swift container. The function is as mentioned below:

```
function sendFile()
{
  console.log('-----INSIDE SENDFILE -----');
  file = fs.readFileSync(__dirname+'/Metadata.txt');
  var postheaders = {
    'x-auth-token' : xAuth,
    'Content-Type' : 'text/plain',
    'Content-Length' : Buffer.byteLength(file, null)
  };

  var options = {
    host: '10.129.103.86',
    port: 8080,
    path: '/v1/AUTH_5b2bcbcb10f347aaa4c7b0e370c2c055/content/Metadata.txt',
    method: 'PUT',
    headers: postheaders,
    encoding: null
  };

  var reqPost = http.request(options, function(response) {
    console.log("-----statusCode: ", response.statusCode);
    response.on('data', function(d) {
    });
  });
  console.log('Writing Data');
  reqPost.write(file,null);
  reqPost.end();
  reqPost.on('error', function(e) {
    console.error(e);
  });
}
```

Fig. 7.1.34: Function definition for storing content metadata in Swift



- Following is what we got as output:

Status Code: 201 (Created)

Which means that the file has been created on Swift container.

```
-----INSIDE SENDFILE -----
Writing Data
Response to Function 2 in waterfall of getContentAPI
{ id: 'ekstep.content.find',
  ver: '3.0',
  ts: '2018-07-02T18:49:03ZZ',
  params:
    { resmsgid: 'b3e8cc04-906d-4e65-b7dc-49627d4d4d02',
      msgid: null,
      err: null,
      status: 'successful',
      errmsg: null },
  responseCode: 'OK',
  result:
    { content:
        { identifier: 'do_21253832575307776012057',
          createdBy: '8e6dd03b-d04f-4c72-b85a-6e2c83b2efdb',
          mimeType: 'application/vnd.ekstep.content-collection',
          languageCode: 'en',
          status: 'Draft' } },
      statusCode: 200 }
In Function 2, res.result:
{ content:
  { identifier: 'do_21253832575307776012057',
    createdBy: '8e6dd03b-d04f-4c72-b85a-6e2c83b2efdb',
    mimeType: 'application/vnd.ekstep.content-collection',
    languageCode: 'en',
    status: 'Draft' } }
-----statusCode: 201-----
```

Fig. 7.1.35 : Request response showing file uploaded to Swift

- Result: File uploaded to Swift with course metadata

```
utkarsh@utkarsh-Inspiron-5558:~$ curl -v -H "x-auth-token:7311e0fb690541f4b8f488b3ef7848bf" http://10.129.103.86:8080/v1/AUTH_5b2bcbcb10f347aaa4c7b0e370c2c055/content/Metadata.txt
* Trying 10.129.103.86...
* Connected to 10.129.103.86 (10.129.103.86) port 8080 (#0)
> GET /v1/AUTH_5b2bcbcb10f347aaa4c7b0e370c2c055/content/Metadata.txt HTTP/1.1
> Host: 10.129.103.86:8080
> User-Agent: curl/7.47.0
> Accept: */*
> x-auth-token:7311e0fb690541f4b8f488b3ef7848bf
>
< HTTP/1.1 200 OK
< Content-Length: 299
< Accept-Ranges: bytes
< Last-Modified: Mon, 02 Jul 2018 18:48:21 GMT
< Etag: 0a23000db38ceebc6cadfbf2eab367dd
< X-Timestamp: 1530557300.74749
< Content-Type: text/plain
< X-Trans-Id: tx35d1db71482145d9a7b65-005b3a7937
< X-Openstack-Request-Id: tx35d1db71482145d9a7b65-005b3a7937
< Date: Mon, 02 Jul 2018 19:12:55 GMT
<
* Connection #0 to host 10.129.103.86 left intact
{"request":{"content":{"name":"Untitled Course","mimeType":"application/vnd.ekstep.content-collection","createdBy":"8e6dd03b-d04f-4c72-b85a-6e2c83b2efdb","contentType":"Course","resourceType":"Course","organization":[],"createdFor":["ORG_001"],"creator":"course admin","code":"org.sunbird.G7PLSK"}}}utkarsh@utkarsh-Inspiron-5558:~$
```

Fig. 7.1.36 : Swift container showing uploaded metadata file

## **H. Build process for deploying source code changes using local registry**

Source code changes made as part of the projects have to be deployed onto the Sunbird server. For this, the deployment script was studied and modified to include the latest images being built.

It was found that loading images to dockerhub was a slow process depending on internet speed. In order to speed up the development time, a local registry was created for storing all the images being developed during development and these images were used for deploying the code changes to the server. For this, the script for loading docker images and creating container had to be modified to point to the local registry instead of dockerhub.com URL.

## **7. DELIVERABLES**

- Detailed step-by-step installation guide for Project Sunbird
- Architecture study of Sunbird platform
- OpenStack Swift installation guide
- OpenStack Swift API Guide
- REST API for data storage on OpenStack Swift
- Dockerization of the data storage REST APIs for easy redistribution
- Process for defining new microservice routes into the Sunbird platform using Kong API gateway
- An approach for enhancing Sunbird platform to store data (course) on OpenStack Swift
- Build process for deploying Sunbird source code changes using local docker registry

## **8. CONCLUSION**

We have studied Docker, Kong API, Sunbird API, Plugin Architecture and have successfully created new IIT Bombay Courses (IITBX Courses) and we also redirected metadata and hierarchy of courses to Swift Storage.

## 9. FUTURE SCOPE

- Improve APIs to store whole course details in Swift Storage.
- Creation of new Plugins in editors.

## 10. REFERENCES

### OpenStack Swift Storage

1. <https://docs.openstack.org/swift/latest/index.html>
2. <https://searchstorage.techtarget.com/definition/cloud-storage-service>
3. <https://docs.openstack.org/keystone/pike/index.html>
4. [https://en.wikipedia.org/wiki/Object\\_storage](https://en.wikipedia.org/wiki/Object_storage)
5. [Swift Object API Overview](#)
6. [Object Storage API](#)
7. [https://docs.openstack.org/swift/latest/development\\_saio.html#debugging-issues](https://docs.openstack.org/swift/latest/development_saio.html#debugging-issues)
8. <https://www.safaribooksonline.com/library/view/openstack-swift/9781491903841/ch09.html>
9. <https://searchcloudcomputing.techtarget.com/definition/cloud-computing>

### Implementation of content repository for Project Sunbird

1. <http://www.sunbird.org/about/>
2. [http://www.sunbird.org/features-documentation/course\\_creation/](http://www.sunbird.org/features-documentation/course_creation/)
3. <https://community.ekstep.in/specifications-guides>
4. <https://docs.docker.com/>
5. <https://angular.io/docs/>
6. <https://nodejs.org/en/docs/>
7. <https://opensource.com/resources/what-docker>
8. <https://www.tutorialspoint.com/docker/index.htm>
9. <https://docs.konghq.com/>
10. <https://github.com/project-sunbird/sunbird-devops/wiki/API-Manager:-Onboarding-APIs>
11. <https://www.getpostman.com/collections/796b00f031097f4b0463>
12. <https://drive.google.com/drive/folders/19OhYUHv0IYyxlEIMWueCB4HrxelDv-dl>

## 11.APPENDIX

### Appendix A: Angular.js

- **What is AngularJS?**

AngularJS is a leading framework for building javascript heavy single-page web applications. Single page apps or SPA's load the entire content of a site within a single page, with the single page usually being an index.html file. This means that once the main page is loaded, clicking on links will not reload the entire page but simply update sections within the page itself. The popularity of SPA-based web apps has taken off based on the fact that they allow us to deliver rich, dynamic and fast-loading content that mimics that of a desktop applications.

- **Why AngularJS?**

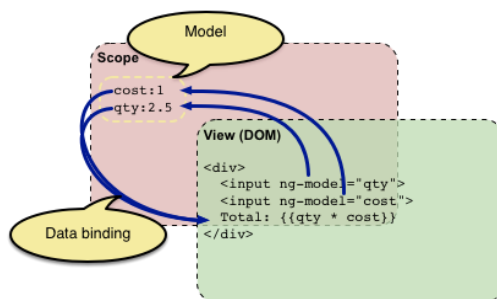
We use AngularJS because of the following advantages:

- No need to use observable functions, Angular analyses the page DOM and builds the bindings based on the Angular-specific element attributes. That requires less writing, the code is cleaner, easier to understand and less error prone.
- Angular modifies the page DOM directly instead of adding inner HTML code. That is faster.
- Data binding occurs not on each control or value change but at particular points of the JavaScript code execution. That dramatically improves performance as a single bulk Model/View update replaces hundreds of cascading data change events.
- Quite a number of different ways to do the same things, thus accommodating to particular development styles and tasks.
- Extended features such as dependency injection, routing, animations, view orchestration, and more.

- **Core features of AngularJS:**

Following are most important core features of AngularJS:

**Data-binding** : It allows us to sync data between model and view components.



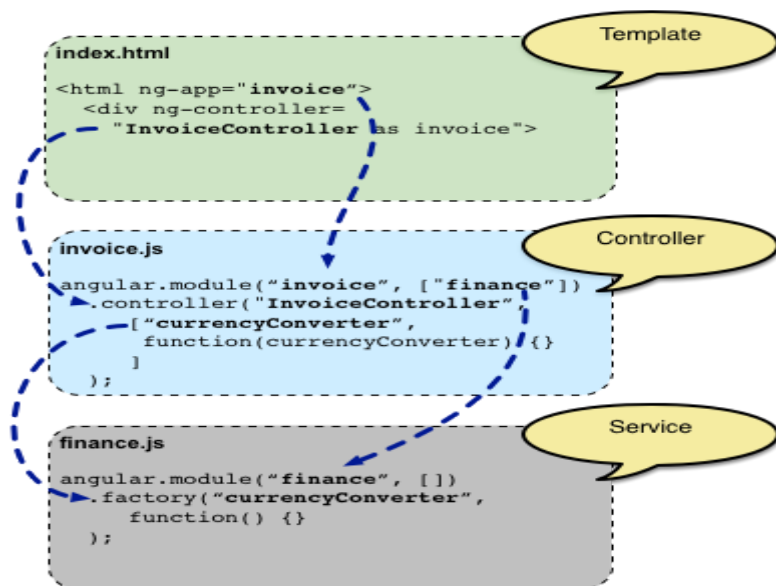
**Fig. 13.A.1 : Angular.js data binding**

**Scope** : They act as a glue between controller and view. Scope refers to the model of the object.

**Controller** : Controllers are javascript functions which are bound to an element and all the functionality of that element is described inside the controller.

**Services** : We can build services in angularJS which can be injected to controllers. Angular also provide some built in services , for example \$http, \$location, \$scope etc.

Here is an example of how template, controller and service together works:



**Fig. 13.A.2 : Angular.js template, controller and service relation**

**Filters** : These select a subset of items from an array and returns a new array.

**Directives** : Directives are used to build our own html element, these can be used to build custom html tag or attribute or class.

**Templates** : These are the rendered view with information from the controller, directives, services and model. These can be a single file or multiple views in one page.

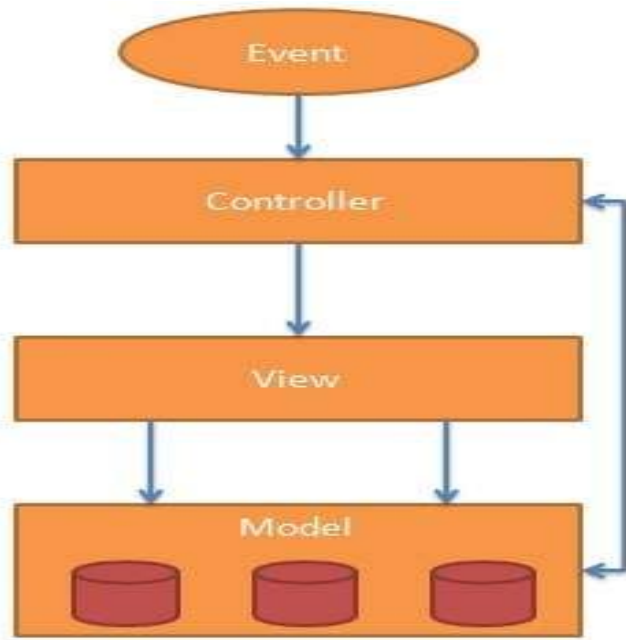
**Routing** : It is concept of switching views. We can specify different views with different routes.

**Dependency Injection** : AngularJS has a built-in dependency injection subsystem that helps the developer by making the application easier to develop, understand, and test.

**Model View Controller** : Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts,

- ➔ **Model** – It is the lowest level of the pattern responsible for maintaining data.
- ➔ **View** – It is responsible for displaying all or a portion of the data to the user.
- ➔ **Controller** – It is a software Code that controls the interactions between the Model and View.

The MVC abstraction can be graphically represented as follows.



**Fig. 13.A.3 : Angular.js Model View Controller architecture**

## Appendix B: Ansible

- **Introduction**

Ansible is a free software tool that allows you to configure and manage nodes. This is achieved by creating groups of machines and describing which actions should be taken on them. The required commands for this are issued from a central location. It has various built-in modules to allow easy configuration management. It uses SSH to connect to different nodes and hence nothing needs to be installed on the targeted machines. Ansible only runs on the main control machine which runs the commands.

- **Basic Definitions**

**Task** : A task is simply the use of one of Ansible modules. Module implements specific functionality. Ansible has a large collection of modules that you can run as tasks. For example, installing a package would be a task since it will require us to use the ‘yum’ module. There are many modules, so a task can be running a service, fetching files, adding user and many more modules waiting for you to explore.

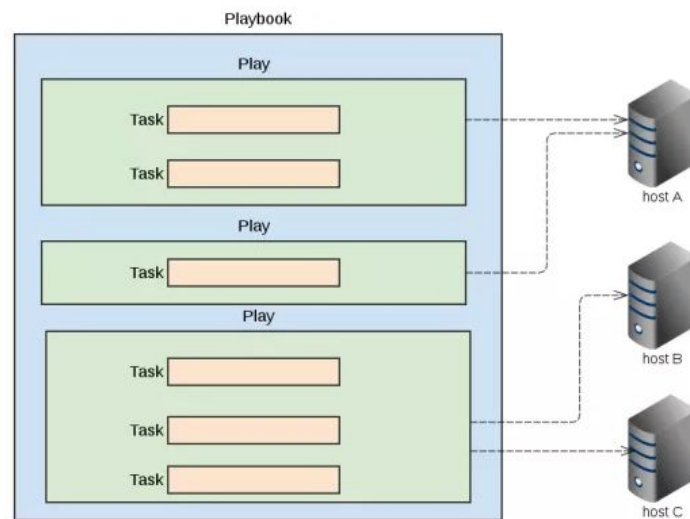
**A few common core modules include:**

- command - Executes a command on a remote node
- script - Runs a local script on a remote node after transferring it
- shell - Execute commands in nodes
- mysql\_db - Add or remove MySQL databases from a remote host
- mysql\_user - Adds or removes a user from a MySQL database
- postgresql\_db - Add or remove PostgreSQL databases from a remote host
- postgresql\_user - Adds or removes a users (roles) from a PostgreSQL database
- fetch - Fetches a file from remote nodes
- template - Templates a file out to a remote server
- yum - Manages packages with the yum package manager
- apt - Manages apt-packages
- git - Deploy software (or files) from git checkouts
- service - Manage services

**Play** : Play is a collection of tasks running on one or more hosts. It includes one or more task.

**Playbook** : Playbook composed of one or more plays. *Playbooks* in Ansible define a series of actions to run, and address particular sets of servers. The playbook should be written such that Ansible can take the template configuration file, compare it to the actual file, and create/update it only if necessary.

You can write playbooks to perform initial server configurations, add users and directories, ensure certain software packages are installed or uninstalled, move files, etc. A playbook can also run a few commands on one set of machines, switch to a different set to run different commands, and then switch back to the original or a different set of machines. It is procedural, and tasks are run in order, top to bottom.



**Fig. 13.B.1 : Ansible playbook connected with hosts**

- **Installation and Preparing Your Environment**

Ansible only needs to be installed on the *control machine*. This is the system from which you frequently access your server.

Make sure that you have Python 2 (versions 2.6 or 2.7) or Python 3 (versions 3.5 and higher) available on the control machine. Note that Windows is not supported as the control machine.

UBUNTU- 16.04 LTS

```
sudo apt-get install software-properties-common
sudo apt-add-repository ppa:ansible/ansible
sudo apt-get update
sudo apt-get install ansible
```

**Fig. 13.B.2 : Ansible installation commands in UBUNTU-16.04 LTS**

Now we need to add the hosts you would like to manage and configure to `/etc/ansible/hosts`.



Let's say you have two nodes, named hostA and hostB. You can simply add these two lines to `/etc/ansible/hosts`.

## • Writing your Playbooks

Before writing any lines, we need to have clear picture of what we want to run on our hosts and how we want them to look after running the playbook.

### Example 1

The following playbook downloads the appropriate packages, turns on the Apache and MySQL services, and creates a basic database and user.

So imagine our environment consists of localhost (127.0.0.1)

We want to have one play with 4 tasks:

1. Install Apache and MySQL
2. Turn ON Apache and MySQL and set them to turn on boot
3. Create a test database
4. Create a new user for connections

## Creating the Playbook

So create a new yml by running the following command:

```
vi first.yml
```

This how the code inside the yml file should look like:

```

- hosts: 127.0.0.1
  connection: local
  become: yes
  become_method: sudo
  tasks:
    - name: "Install Apache, MySQL"
      apt: name={{ item }} state=present
      with_items:
        - apache2
        - mysql-server
        - python-mysqldb

    - name: "Turn on Apache and MySQL and set them to run on boot"
      service: name={{ item }} state=started enabled=yes
      with_items:
        - apache2
        - mysql

    - name: Create a test database
      mysql_db: name=testdb
                state=present

    - name: Create a new user for connections
      mysql_user: name=webapp
                  password=mypassword
                  priv="*:ALL" state=present
  
```

**Fig. 13.B.3 : Ansible playbook example**

## Running the Playbook

```
ansible-playbook first.yml
```

```
sreekanth@sreekanth-HP-Pavilion-Notebook:~/ansible$ ansible-playbook first.yml --ask-become-pass
SUDO password:

PLAY [127.0.0.1] *****
TASK [Gathering Facts] *****
ok: [127.0.0.1]
TASK [Install Apache, MySQL] *****
changed: [127.0.0.1] => (item=[u'apache2', u'mysql-server', u'python-mysqldb'])
TASK [Turn on Apache and MySQL and set them to run on boot] *****
ok: [127.0.0.1] => (item=apache2)
ok: [127.0.0.1] => (item=mysql)
TASK [Create a test database] *****
changed: [127.0.0.1]
TASK [Create a new user for connections] *****
changed: [127.0.0.1]
PLAY RECAP *****
127.0.0.1 : ok=5    changed=3    unreachable=0    failed=0

sreekanth@sreekanth-HP-Pavilion-Notebook:~/ansible$
```

Fig. 13.B.4 : Ansible running playbook

## Appendix C: Docker

- **Introduction of Docker :**

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. Build once, run anywhere . A clean, safe, hygienic and portable runtime environment for your app. No worries about missing dependencies, packages and other pain points during subsequent deployments. Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying. And eliminate inconsistencies between development, test, production, and customer environments

- **Steps to install Docker :**

1. To update apt, issue the command:

```
$ sudo apt update
```

2. Mostly the docker package is available in all linux systems. So by below command you can install docker

```
$ sudo apt-get install docker.io
```

3. If you find any problem in the above command you will have to run an installation script provided by Docker.

```
$ sudo apt-get install wget  
$ wget -qO- https://get.docker.com/ | sh
```

4. If you want to be able to run Docker containers as your user, not only as root, you should add yourself to the group called docker using the following command.

```
$ sudo groupadd docker  
$ sudo usermod -aG docker $USER
```

Then you'll want to log out and then log back in for the changes to take effect.

- **Docker components :**

- 1. Docker images**

- A Docker image is a read-only template. For example, an image could contain an Ubuntu operating system with Apache and your web application installed.
- Images are used to create Docker containers. Docker provides a simple way to build new images or update existing images, or you can download Docker images that other people have already created.
- Docker images are the build component of Docker.

- 2. Docker File**

- Docker can build images automatically by reading the instructions from a Dockerfile.
- A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.
- The docker build command builds an image from a Dockerfile and a context.

- 3. Docker Containers**

- Executing an image is called “container” .
- Containers, in short, contain applications in a way that keep them isolated from the host system that they run on.
- Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

- 4. Docker Hub**

- It is docker registry which holds images.
- These are public or private stores from which you upload or download images.
- The public Docker registry is provided with the Docker Hub. (<https://hub.docker.com>)
- It serves a huge collection of existing images for your use. These can be images you create yourself or you can use images that others have previously created.
- Docker registries are the distribution component of Docker.

- **Docker Architecture :**

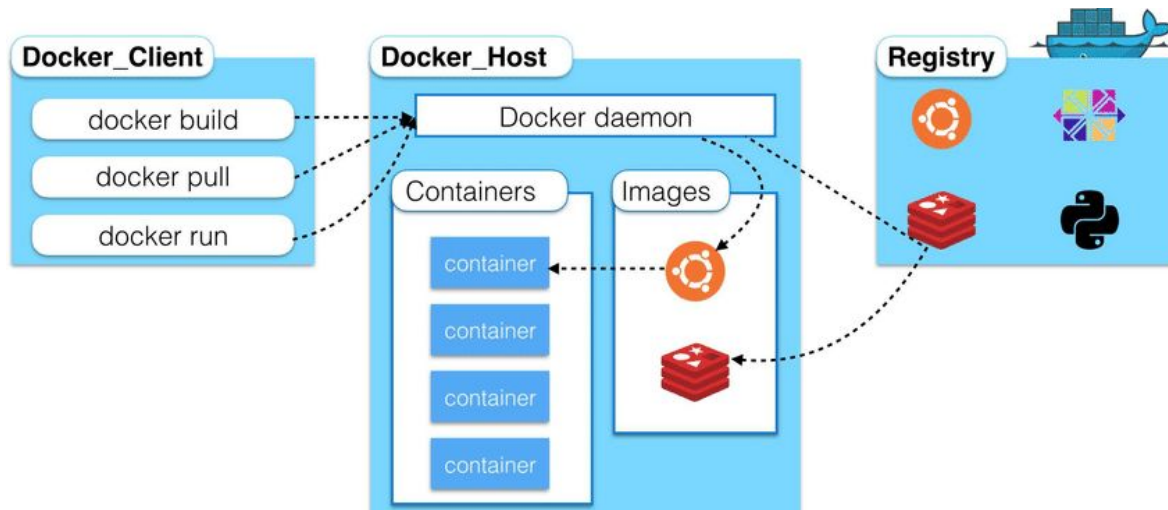


Fig. 13.C.1 : Docker architecture

- **Docker Commands :**

1. **docker pull ubuntu**

- Executes “docker pull Ubuntu”
- This pulls(downloads) an image from the docker library
- The image contains bare copy of ubuntu

2. **docker images**

- Executes “docker images”
- This generates a list of images known to Docker on your machine

3. **docker run ubuntu**

- Executes “docker run Ubuntu”
- This executes an image called ubuntu. An executing image is called a “container”.
- You are now inside the container.
- Execute “ls”.

A directory structure is set up (The files responsible for the container).

4. **docker ps -a**

- Executes “docker ps -a”
- This generates a list of all of the containers (both running and not-running).

5. **docker ps**

- Executes “docker ps”
- This generates a list of all running containers.

**6. docker rm <container\_name (or) id>**

- Executes “docker rm <container\_name (or) id>”
- This will remove the container specified. You can remove only stopped containers. (‘docker stop <container\_id>’ )

**7. docker exec -it <container\_id (or) name> /bin/sh**

- Executes “docker exec -it <container\_id (or) name> /bin/sh”
- You are now inside the container.
- Execute “ls”. A directory structure is set up (The files responsible for the container).

**8. docker rmi <image\_name (or) id>**

- Executes “docker rmi <image\_name (or) id>”
- This will remove the image specified.

**9. docker logs -f <container\_id (or) container\_name>**

- Executes “docker logs -f <container\_id (or) container\_name>”
- To see all the logs of a particular container until the command executed and also ongoing logs .

- **Sharing image**

Multiple team members may wish to share images. Images can be in production, under development or under test. Docker Hub is a repository where images can be stored and shared. You can also store images in your private registry . Each image is tagged to allow versioning Any image can be “pulled” to any host (with appropriate credentials). Tagging as “latest” allows updates to be propagated. Pull :latest gets the last image checked into repository with that name.

**Sharing involves mainly in three different types :**

**(a) Sharing via docker hub :**

- Create a account in dockerhub (hub.docker.com). And remember your username and password.
- Now go to terminal and run the command “docker login” .And then enter your username and password to login.
- Then tag your image to a new name such that it should be in the format like {username}/{imagename:tag}

**\$ docker tag your\_imagename:tag your\_username/imagename:tag**

- Now push your image to docker hub.

```
$ docker push your_username/imagename:tag
```

- ➔ Now the docker hub has your image.
- ➔ You can also push your same image with different tag. So that it will push into same image repository with different tag. And then the repository contains different images with same imagename and different tags. You can pull the image by differentiating the tag.
- ➔ Now the person who needs to pull that image can pull with the below Command.

```
$ docker pull <usernameofimageholder>/<imagename:tag>
```

### **(b) Sharing via Private repository :**

You might have the need to have your own private repositories. You may not want to host the repositories on Docker Hub. For this, there is a repository container itself from Docker.

- ➔ Use the Docker run command to download the private registry. This can be done using the following command :

```
$ docker pull registry:2
```

‘registry’ is the container managed by Docker which can be used to host private Repositories.

- ➔ And it is better to mount a volume which keeps a backup of all your pushed Images. So that if the registry container is stopped it will restore all your pushed images whenever it restarts.

```
$ mkdir registry_backup  
$ cd registry_backup
```

and run the registry image along with mounting :

```
$ docker run -d -p 9999:5000 -v $(pwd)/:/var/lib/registry registry:2
```

Let's do a ‘docker ps’ to see that the registry container is indeed running

Now your private registry setup is done.

- ➔ Now let's tag one of our existing images (let's take centos) so that we can push it to our local Repository.

```
$ docker tag centos localhost:9999/centos
```

➔ Now let's use the Docker push command to push the repository to our private Repository.

```
$ docker push localhost:9999/centos
```

➔ You can share your private registry image to others also. To do that go to the file /etc/docker/daemon.json and add this line.

```
{
  "insecure-registries" : [ "<your_IP_address>:9999" ]
}
```

And in the system from where you need to pull this private image also add the above line in /etc/docker/daemon.json in its system. (The ip address which is mentioning here should be the image holder's ip address.)

➔ After that you can pull that private image with the following command.

```
$ docker pull <ipaddressofimageholder>:9999/centos
```

### (c) Share image via .tar file

- ➔ Sometimes I want to save a docker image and then use it on another computer without going through the hassle of uploading it to docker hub or private registry
- ➔ Docker images aren't really stored as files that you can just grab and move to another computer. But you can save and then load the images like this:

If you want to save the image as a tar archive, using docker save -o:

```
$ docker save -o abcd.tar <image_name:tag>
```

Then copy the abcd.tar file into another computer and load there :

```
$ docker load abcd.tar
```

➔ Run “ docker images ” . It will show up in your docker images list .

### Docker clustering :

- cluster is a group of docker-enabled nodes . Each node has either a manager or worker role in the cluster. At least one master node is required for a cluster to operate.
- manager refers to the node maintaining the state of the cluster. There can be one or more managers in a cluster.



- worker refers to the node sharing the workload in the cluster.

### Docker swarm:

With Docker Swarm, you can create and manage Docker clusters. Swarm can be used to distribute containers across multiple hosts.

### Cluster initialisation :

```
$ docker swarm init
```

- This node will become the manager node.

Join cluster :

```
$ docker swarm join-token manager    (To join the cluster as manager)
$ docker swarm join-token worker     (To join the cluster as worker)
```

The token used here is generated when that cluster is initialized.

### Leave cluster :

```
$ docker swarm leave    (For worker )
$ docker swarm leave -f  (For manager)
```

- The node leaves the cluster is NOT removed automatically from the node table. Instead, the node is marked as Down. If you want the node to be removed from the table, you should run the command **docker node rm**.

In swarm cluster, a service is created by deploying a container in the cluster. The container can be deployed as a single instance (i.e. task) or multiple instances to achieve load-balancing. Services are really just “containers in production.” A service only runs one image, but it codifies the way that image runs—what ports it should use, how many replicas of the container should run so the service has the capacity it needs, and so on.

It's very easy to define, run, and scale services with the Docker platform, just write a docker-compose.yml file. A docker-compose.yml file is a YAML file that defines how Docker containers should behave in production.

**Ex: (docker-compose.yml)**

```

version: "3"
services:
  web:
    image: username/repo:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "4000:80"
    networks:
      - application_default
networks:
  Application_default:

```

- (replicas:5) - Run 5 instances of “repo” image as a service called “web”. Resulting to run that image as 5 containers which helps to distribute the incoming requests among them for load-balancing.
- Restart\_policy - Immediately restart containers if one fails.
- Network - Instruct web’s containers to share port 80 via a network called “application\_default”.
- Define the webnet network with the default settings

**Create app with that service :**

```
$ docker stack deploy -c docker-compose.yml <app-name>
```

- single service stack runs replicas no.of container instances of our deployed image on one host.

-To list all the services running on the host.

```
$ docker service ls
```

-To list all the stacks with no.of services in that host.

```
$ docker stack ls
```

## Appendix D: Kong

- **Introduction**

Kong is an open source API gateway. That means it is a form of middleware between computing clients and your API-based applications. Kong easily and consistently extends the features of your APIs. It helps in managing API traffic with different plugins such as rate limiting, authentication etc.

- **Terminologies**

**client:** Refers to the downstream client making requests to Kong's proxy port.

**upstream service:** Refers to your own API/service sitting behind Kong, to which client requests are forwarded.

**Service:** Service entities, as the name implies, are abstractions of each of your own upstream services. Examples of Services would be a data transformation microservice, a billing API, etc.

**Route:** This refers to the Kong Routes entity. Routes are entry points into Kong, and defining rules for a request to be matched, and routed to a given Service.

**Plugin:** This refers to Kong "plugins", which are pieces of business logic that run in the proxying lifecycle. Plugins can be configured through the Admin API - either globally (all incoming traffic) or on specific Routes and Services

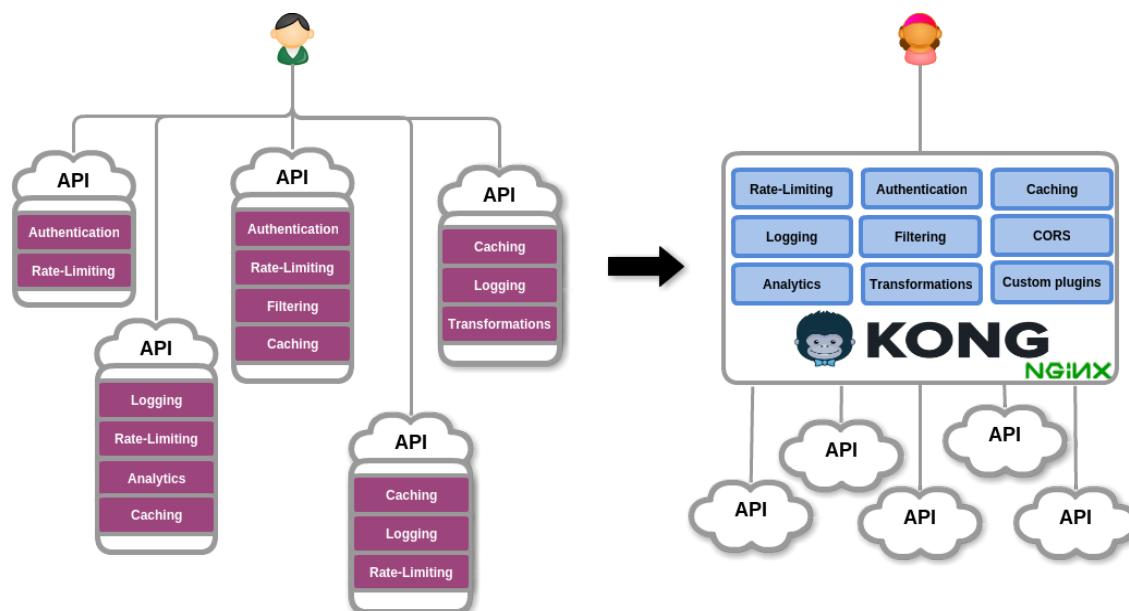
- **Architecture of Kong**



Fig. 13.D.1 : Kong architecture

- **Advantages of using Kong**

1. Radically Extensible: Ready-to-deploy plugins add powerful functionality to your APIs and applications
2. Open Source: Open and developer-friendly, tens of thousands have embraced the Kong microservice API gateway.
3. RESTful Interface: Kong operates through a simple and easy-to-use RESTful API. In other words the transformation we obtain after we use Kong is:



**Fig. 13.D.2 : Kong transforming APIs**

- **Functionality of Kong**

A typical kong setup is made of two components:

**1) Kong's server (based on NGINX HTTP server)**

The Kong Server is the server that will actually process the API requests and execute the configured plugins to provide additional functionalities to the underlying APIs before proxying the request upstream. The default ports through which kong listens are:

8000: for proxying

8443: for proxying HTTPS content

8001: for admin API

8444: for admin API over HTTPS.

Kongs Admin API can be used to create new users, configure Kongs, enable/disable plugins etc. Since this admin API uses REST interface, it becomes easy to integrate kong with existing systems.

**2) Kong's dataserver (PostgreSQL or Apache Cassandra)**

Kong uses an external datastore to store its configuration such as registered APIs, Consumers and Plugins. Plugins themselves can store every bit of information they need to be persisted, for example rate-limiting data or Consumer credentials.

Kong maintains a cache of this data so that there is no need for a database roundtrip while proxying requests, which would critically impact performance. This cache is invalidated when calls to the Admin API are made.

This architecture allows Kong to scale horizontally by simply adding new nodes that will connect to the same datastore and maintain their own cache.

- **Kong Plugins**

Plugins are one of the most important features of Kong. Many Kong API gateway features are provided by plugins. Authentication, rate-limiting, transformation, logging etc, are all implemented independently as plugins. Plugins can be installed and configured via the Admin API running alongside Kong.

- **Examples to Add API**

Kong listens at port 8001 for admin API requests, so by sending the following request it is possible to add an API to local instance of kong.

POST : <http://10.129.103.85/apis/>

Request Body (example):

```
name: Swift
upstream_url: http://10.196.0.56:8000/files/info/
request_path: /swiftapis
strip_request_path: true
```

## Appendix E: Node.js

- **What is Node.js ?**

Node.js is a JavaScript runtime environment built on Chrome's V8 JavaScript engine. It includes everything you need to execute a program written in JavaScript.

Engine : parse your code and convert it to runnable commands

JavaScript ==> V8(written in C++) ==> Machine Code

Runtime : provide some objects to javascript so that it can interact with the outside world. In Chrome you have the window, DOM objects etc, while node gives you require, Buffers and Processes.

- **How it came into existence ?**

Earlier Javascript could only be used for making a website interactive. Node.js came into existence when the original developers of JavaScript extended it from something you could only run in the browser to something you could run on your machine as a standalone application.

JavaScript now has the capability to do things that other scripting languages like Python can do.

- **Why Node.js ?**

Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

- **What is Event-driven ?**

In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected. When Node starts its server, it simply initiates its variables, declares functions and then simply waits for the event to occur.

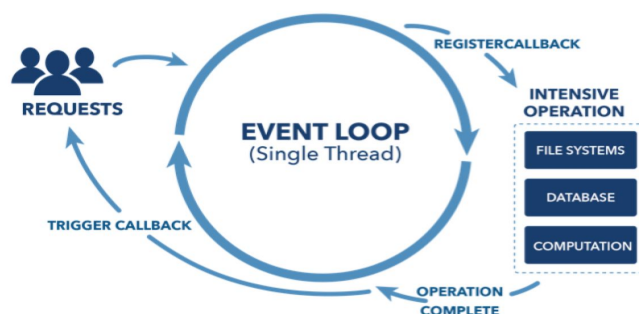


Fig. 13.E.1 : Node.js Event loop

- **What is Callback?**

Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.

Node.js is a single-threaded application, but it can support concurrency via the concept of event and callbacks. Every API of Node.js is asynchronous and being single-threaded, they use async function calls to maintain concurrency. Node uses observer pattern. Node thread keeps an event loop and whenever a task gets completed, it fires the corresponding event which signals the event-listener function to execute.

- **Blocking vs Non-Blocking**

Blocking	Non-Blocking
<b>Blocking</b> is when the execution of JavaScript in the Node.js process must wait until a non-JavaScript operation completes. This happens because the event loop is unable to continue running JavaScript while a blocking operation is occurring.	<b>Non-Blocking</b> is when the execution of JavaScript in the Node.js process doesn't wait for a non-JavaScript to complete.
<b>Blocking</b> methods execute <b>synchronously</b> .	<b>Non-Blocking</b> methods execute <b>asynchronously</b> .
Code	
<pre>const fs = require('fs'); const data = fs.readFileSync('/file.md'); // Further execution is blocked, waiting for the read File to complete.</pre>	<pre>const fs = require('fs'); fs.readFile('/file.md', (err, data) =&gt; {   if (err) throw err; });</pre>



## **Appendix F: Introduction of REST API and Django Rest Framework**

### **1. DJANGO**

Django is a high-level Python Web framework, written in Python, which follows the model-view-template (MVT) architectural pattern. Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability of components, less code, low coupling, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models. Django also provides an optional administrative create, read, update and delete, interface that is generated dynamically through introspection and configured via admin models. It's free and open source.

#### **Advantages of Django:**

- Object-Relational Mapping (ORM) Support
- Multilingual Support
- Framework Support
- Ready-to-use Administration GUI
- Lightweight Development Environment
- Secure

### **2. REST API**

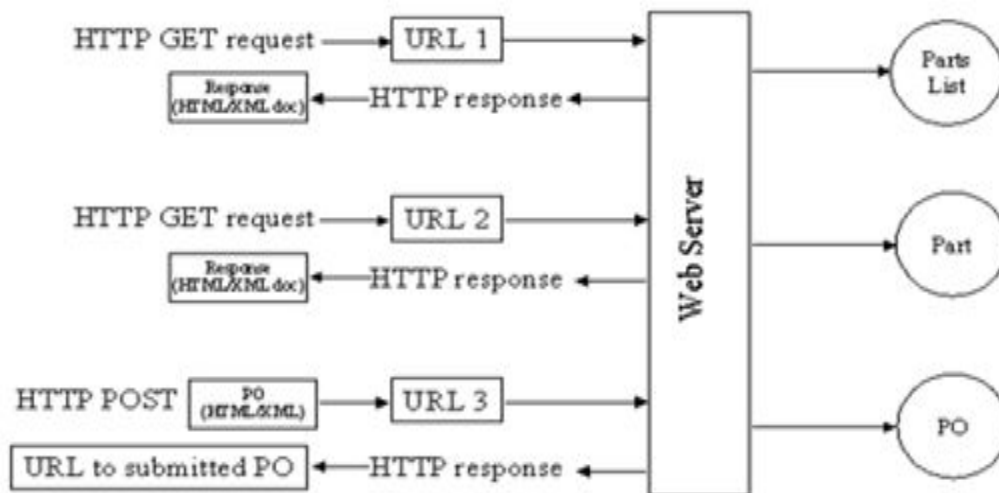
A REST API, also known as a RESTful web service, which means Representational State Transfer (REST), is an architectural style and an approach to communications between services that are online & often used in Web Services / Web API development. REST is a web standards based architecture and uses HTTP Protocol for data communication. It revolves around resources where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.

In REST architecture, a REST Server simply provides access to resources and the REST client accesses and presents the resources. Here each resource is identified by URIs/ Global IDs. REST uses various representations to represent a resource like Text, JSON and XML. JSON is now the most popular format being used in Web Services.

#### **Understanding Rest API Design:**

- Client-Server: The client-server constraint works on the concept that the client and the server should be separate from each other and allowed to evolve individually and independently. I should be able to make changes to my mobile application without impacting either the data structure or the database design on the server.

- Stateless : REST APIs are stateless, meaning that calls can be made independently of one another, and each call contains all of the data necessary to complete itself successfully.
- Cache : To improve network efficiency, responses must be capable of being labeled as cacheable or non-cacheable.
- Uniform Interface : Decoupling of client from server is done using uniform interface that allows independent evolution of the application without having the application's services, models, or actions tightly coupled to the API layer itself. (e.g., HTTP GET, POST, PUT, DELETE).
- Layered System : Each layer has a specific functionality and responsibility, with the models comprising how the data should be formed, the controller focusing on the incoming actions and the view focusing on the output.
- Named Resources : The system is comprised of resources which are named using a URL.



**Fig. 13.F.1 : REST API Request/Response Flow Diagram**

### 3. DJANGO REST FRAMEWORK

In our project we have used Django Rest Framework (DRF) which is a powerful and flexible toolkit for building Web APIs. It supports both ORM and Non-ORM data sources. A DRF API is composed of 3 layers: the serializer, the viewset, and the router:

- **Serializers:** converts the information stored in the database and defined by the [Django models](#) into native Python data types that can be easily rendered in JSON, XML or other content types.
- **ViewSet:** defines the functions (read, create, update, delete) which will be available via the API
- **Router:** defines the URLs which will provide access to each viewset

**Need of Framework:**

- Serialization/Deserialization
- Custom Queryset
- Pagination
- Proper Http Response Handling
- Data Validation
- Caching
- Authentication
- Throttling

## Appendix G: Sunbird Installation

- **Prerequisites:**
  - A system, running Ubuntu server 16.04 LTS.
  - Ekstep API keys to access the Ekstep content repository.
  - Create a common linux user having root privileges (e.g. ekstep) on all the servers.
- **Generate Ekstep API key:**
  - Go to [qa.ekstep.in](https://qa.ekstep.in) and request for API access to the Ekstep admin.
  - After we get response from the admin to our notified mail, We can Generate new credentials.
  - From there we generate key and secret.
  - Then go to [JSON Web Tokens - jwt.io](https://jwt.io) to generate API key

- **Git clone of Sunbird-devops repository:**

### Commands:

```
apt-get update -y && apt-get install git -y
git clone https://github.com/project-sunbird/sunbird-devops.git
cd sunbird-devops/deploy
```

- **Updating of config file:**
  - Important fields: sunbird\_auth\_token, ekstep\_api\_key, azure credentials
  - Azure credentials are not required in this projects , but still you should not keep that field empty.
  - Find sample config file on confluence

[NOTE: Sunbird installation has different stages sanity, deps, dbs, apis, proxy, keycloak, badger, core, logging, monitoring.

Among these stages **logging** and **monitoring** stages are **optional**, and **badger** stage requires **azure account details**, we have not tried badger stage.]

- **Sunbird stages installation with commands:**

### Stage 1- config:

```
$ ./sunbird_install.sh -s config
```

### Stage 2- deps:

```
$ ./sunbird_install.sh -s deps
```

### Stage 3- dbs:

```
$ ./sunbird_install.sh -s dbs
```

- ➔ If you get connection refused error: follow the steps in error guide(2)
- ➔ Then retry ./sunbird\_install.sh -s dbs

### Change pg\_hba.conf file

- This step is necessary for executing next installation stage.
- Follow the steps in error guide(3)

### Stage 4- apis:

```
$ ./sunbird_install.sh -s apis
```

- Check Running Docker Containers:

```
$ docker ps
```

- You must have three running docker containers:  
Admin utils, Kong API, Echo server

### Copy JWT token to config file

- After stage 4 the jwt token player will be generated , copy that to sunbird\_suth\_token in config file.

```
$ vi ~/jwt_token_player.txt
```

- Copy the token,
- Paste in sunbird\_auth\_token field of config file

### Stage 5- proxy:

- ./sunbird\_install.sh -s proxy
- One more container will be added called proxy service

### Stage 6- keycloak:

```
$ ./sunbird_install.sh -s keycloak
```

### Verify keycloak installation:

- Go to url: http://<IP address>/auth

### Stage 7- core:

```
$ ./sunbird_install.sh -s core
```

-You must have following containers running on docker:

Player, Actor, Learner\_service, Content\_service

- Error guide:

1 ) Linux User not connected with the Root User

**Error message :- 10.129.103.89 : UNREACHABLE**

Things to do-

1. Unlock the root

```
$ sudo passwd root { to change the root password }
```

2. Install ansible using command:

```
$ sudo apt install ansible
```

3. Go the ansible hosts file and create a host

[ekstep]

10.129.103.85

4. Generate the rsa key

```
$ ssh-keygen -t rsa
```

5. Run copy-id command:

```
$ ssh-copy-id 10.129.103.89
```

6. Then try to check if the host are connected:

```
$ ansible all -m ping
```

#### **Extra :**

While doing ssh-copy-id 10.192.103.89 , if you get permission denied then go to /etc/ssh

**\$ vi sshd\_config**

Change **Permit Root login prohibit**

To Permit Root login yes

2 ) If there is a connection refusal error ( Database refuse to connect )

**Error message :- Refused to connect**

Things to do-

1. Check the logs for debugging in **/var/log** .
2. Go inside **/etc**
3. Edit the hosts ( add a host )  
**127.0.1.1      <hostname for which error is coming as unknown host name>**
4. Then try again **./sunbird\_install -s dbs**

3 ) Connecting Kong API docker container with the Postgresql Database

**Error message :- RETRYING: Ensure kong\_admin\_api\_url is available before running the scripts**

Things to do-

1. Go to **/etc/postgresql/9.5/main**
2. Check **ifconfig**
3. Copy dockerbrigde's Network address i.e **172.18.0.0/16**

**\$ sudo vi pg\_hba.conf**

4. Add a line  
**host all all 172.18.0.0/16 trust**
5. Restart the Postgres service

**\$ sudo systemctl restart postgresql@9.5-main.service**

## Appendix H: Swift Installation

- **Installation of Openstack Swift(SAIO - Swift All In One) on Ubuntu 16.04:**

### Step 1: Install dependencies:

```
sudo apt-get update
sudo apt-get install curl gcc memcached rsync sqlite3 xfsprogs \
    git-core libffi-dev python-setuptools \
    librasurecode-dev libssl-dev
sudo apt-get install python-coverage python-dev python-nose \
    python-xattr python-eventlet \
    python-greenlet python-pastedeploy \
    python-netifaces python-pip python-dnspython \
    python-mock
```

### Step 2: Setup a partition for storage

Here, we will be using a separate partition for the storage. So please make sure that you add another device when creating VM.

#### Step 2.1: Setup a single a partition

```
sudo fdisk /dev/sdb
sudo mkfs.xfs /dev/sdb1
```

Where ‘sdb’ is the name of the disk attached for storage. Here, we formatted the disk into ‘xfs’ file system.

#### Step 2.2: Mounting the sdb drive on every boot of the system.

There are 2 possible ways to do this.

You can either edit the /etc/fstab file and write your mount instructions in that or you can run mount script manually after every restart of system.

If you want to edit the /etc/fstab file, add the following line in the file:

```
/dev/sdb1 /mnt/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 0
```

(Note: It is recommended not to use the fstab file for this purpose. /etc/fstab is used to automatically mount filesystems at boot time. Make sure that the only items listed there are critical for the booting of the machine, because the machine might hang upon boot if it can’t mount a device. This is likely when running a big box full of disks.)



If you want to run mount script manually after every restart of the system use the following command:

```
mount -t xfs -o nobarrier,logbufs=8 /dev/sdb1 /mnt/sdb1
```

### Step 2.3: Create the mount point and the individualized links:

```
sudo mkdir /mnt/sdb1
sudo mount /mnt/sdb1
sudo mkdir /mnt/sdb1/1 /mnt/sdb1/2 /mnt/sdb1/3 /mnt/sdb1/4
sudo chown ${USER}:${USER} /mnt/sdb1/*
sudo mkdir /srv
for x in {1..4}; do sudo ln -s /mnt/sdb1/$x /srv/$x; done
sudo mkdir -p /srv/1/node/sdb1 /srv/1/node/sdb5 \
    /srv/2/node/sdb2 /srv/2/node/sdb6 \
    /srv/3/node/sdb3 /srv/3/node/sdb7 \
    /srv/4/node/sdb4 /srv/4/node/sdb8 \
    /var/run/swift
sudo chown -R ${USER}:${USER} /var/run/swift
# **Make sure to include the trailing slash after /srv/$x/**
for x in {1..4}; do sudo chown -R ${USER}:${USER} /srv/$x/; done
```

Here, “\${USER}” denotes the username of the system that you are working on.

In above listed commands, we are creating 4 directories in our sdb1 drive to represent 4 different devices of the storage cluster. Then we have linked those 4 partitions with the 4 different server directories. Then we have given our user the ownership of /var/run/swift and all the server directories.

### Step 3: Post-Device Setup

Add the following lines to **/etc/rc.local** (before the **exit 0**):

```
mkdir -p /var/cache/swift /var/cache/swift2 /var/cache/swift3 /var/cache/swift4
chown <your-user-name>:<your-group-name> /var/cache/swift*
mkdir -p /var/run/swift
chown <your-user-name>:<your-group-name> /var/run/swift
```

### Step 4: Creating an XFS tmp dir

The tests given at the end of the installation process require having an XFS directory available in /tmp or in the TMPDIR environment variable. To set up /tmp with an XFS filesystem, do the following:

```
cd ~
truncate -s 1GB xfs_file # create 1GB fil for XFS in your home directory
mkfs.xfs xfs_file
```

```
sudo mount -o loop,noatime,nodiratime xfs_file /tmp
sudo chmod -R 1777 /tmp
```

To persist this, edit and add the following to **/etc/fstab**:

```
/home/<your-user-name>/xfs_file /tmp xfs rw,noatime,nodiratime,attr2,inode64,noquota 0 0
```

## **Step 5: Get the code**

### **Step 5.1: Clone the python-swiftclient repo**

```
cd $HOME;
git clone https://github.com/openstack/python-swiftclient.git
```

### **Step 5.2: Build a development installation of python-swiftclient**

```
cd $HOME/python-swiftclient; sudo python setup.py develop; cd -
```

### **Step 5.3: Clone the Swift repo**

```
git clone https://github.com/openstack/swift.git
```

### **Step 5.4: Build a development installation of Swift**

```
cd $HOME/swift;
sudo pip install --no-binary cryptography -r requirements.txt;
sudo python setup.py develop; cd -
```

### **Step 5.5: Install Swift's test dependencies**

```
cd $HOME/swift; sudo pip install -r test-requirements.txt
```

## **Step 6: Setting up rsync**

### **Step 6.1:**

**Create /etc/rsyncd.conf and edit as per the sed command mentioned below.**

```
sudo cp $HOME/swift/doc/saio/rsyncd.conf /etc/
sudo sed -i "s/<your-user-name>/${USER}/" /etc/rsyncd.conf
```

### **Step 6.2: edit the following line in /etc/default/rsync:**

```
RSYNC_ENABLE=true
```

### Step 6.3: Start the rsync daemon

```
sudo systemctl enable rsync
sudo systemctl start rsync
```

### Step 6.4: Verify rsync is accepting connections for all servers

#### Command:

```
rsync rsync://pub@localhost/
```

#### Expected output:

```
account6012
account6022
account6032
account6042
container6011
container6021
container6031
container6041
object6010
object6020
object6030
object6040
```

### Step 7: Start memcached

```
sudo systemctl enable memcached.service
sudo systemctl start memcached.service
```

### Step 8: Setting up rsyslog for individual logging

#### Step 8.1: Install the Swift rsyslogd configuration

```
sudo cp $HOME/swift/doc/saio/rsyslog.d/10-swift.conf /etc/rsyslog.d/
```

To decide if you want all logs in one file or all the logs separated out or if you want hourly logs for stat processing, review the conf file. (Default contents of the conf file are given below)

```
# Uncomment the following to have a log containing all logs together
#local1,local2,local3,local4,local5.* /var/log/swift/all.log

# Uncomment the following to have hourly proxy logs for stats processing
```

```

#$template
HourlyProxyLog,"/var/log/swift/hourly/%$YEAR%%$MONTH%%$DAY%%$HOURL%"
#local1.*;local1.!notice ?HourlyProxyLog

local1.*;local1.!notice /var/log/swift/proxy.log
local1.notice      /var/log/swift/proxy.error
local1.*           ~

local2.*;local2.!notice /var/log/swift/storage1.log
local2.notice      /var/log/swift/storage1.error
local2.*           ~

local3.*;local3.!notice /var/log/swift/storage2.log
local3.notice      /var/log/swift/storage2.error
local3.*           ~

local4.*;local4.!notice /var/log/swift/storage3.log
local4.notice      /var/log/swift/storage3.error
local4.*           ~

local5.*;local5.!notice /var/log/swift/storage4.log
local5.notice      /var/log/swift/storage4.error
local5.*           ~

local6.*;local6.!notice /var/log/swift/expirer.log
local6.notice      /var/log/swift/expirer.error
local6.*           ~

```

**Step 8.2: Edit /etc/rsyslog.conf and make the following change (usually in the “GLOBAL DIRECTIVES” section)**

```
$PrivDropToGroup adm
```

**Step 8.3: If using hourly logs (see above) perform:**

```
sudo mkdir -p /var/log/swift/hourly
```

**Otherwise perform:**

```
sudo mkdir -p /var/log/swift
```

**Step 8.4: Setup the logging directory and start syslog**

```
sudo chown -R syslog.adm /var/log/swift  
sudo chmod -R g+w /var/log/swift  
sudo service rsyslog restart
```

## Step 9: Configuring each node

### Step 9.1: Optionally remove an existing directory

```
sudo rm -rf /etc/swift
```

### Step 9.2: Populate the /etc/swift directory itself

```
cd $HOME/swift/doc; sudo cp -r saio/swift /etc/swift; cd -  
sudo chown -R ${USER}:${USER} /etc/swift
```

### Step 9.3: Update <your-user-name> references in the Swift config files

```
find /etc/swift/ -name \*.conf | xargs sudo sed -i "s/<your-user-name>/${USER}/"
```

## Step 10: Setting up scripts for running Swift

### Step 10.1: Copy the Swift scripts for resetting the environment

```
mkdir -p $HOME/bin  
cd $HOME/swift/doc; cp saio/bin/* $HOME/bin; cd -  
chmod +x $HOME/bin/*
```

### Step 10.2: Install the sample configuration file for running tests

```
cp $HOME/swift/test/sample.conf /etc/swift/test.conf
```

### Step 10.3: Add an environment variable for running tests below

```
echo "export SWIFT_TEST_CONFIG_FILE=/etc/swift/test.conf" >> $HOME/.bashrc
```

### Step 10.4: Be sure that your PATH includes the bin directory

```
echo "export PATH=${PATH}:${HOME}/bin" >> $HOME/.bashrc
```

### Step 10.5: Source the above environment variables into your current environment

```
. $HOME/.bashrc
```

### Step 10.6: Construct the initial rings using the provided script

```
remakerings
```

The **remakerings** script looks like the following:

```
#!/bin/bash

set -e

cd /etc/swift

rm -f *.builder *.ring.gz backups/*.builder backups/*.ring.gz

swift-ring-builder object.builder create 10 3 1
swift-ring-builder object.builder add r1z1-127.0.0.1:6010/sdb1 1
swift-ring-builder object.builder add r1z2-127.0.0.2:6020/sdb2 1
swift-ring-builder object.builder add r1z3-127.0.0.3:6030/sdb3 1
swift-ring-builder object.builder add r1z4-127.0.0.4:6040/sdb4 1
swift-ring-builder object.builder rebalance
swift-ring-builder object-1.builder create 10 2 1
swift-ring-builder object-1.builder add r1z1-127.0.0.1:6010/sdb1 1
swift-ring-builder object-1.builder add r1z2-127.0.0.2:6020/sdb2 1
swift-ring-builder object-1.builder add r1z3-127.0.0.3:6030/sdb3 1
swift-ring-builder object-1.builder add r1z4-127.0.0.4:6040/sdb4 1
swift-ring-builder object-1.builder rebalance
swift-ring-builder object-2.builder create 10 6 1
swift-ring-builder object-2.builder add r1z1-127.0.0.1:6010/sdb1 1
swift-ring-builder object-2.builder add r1z1-127.0.0.1:6010/sdb5 1
swift-ring-builder object-2.builder add r1z2-127.0.0.2:6020/sdb2 1
swift-ring-builder object-2.builder add r1z2-127.0.0.2:6020/sdb6 1
swift-ring-builder object-2.builder add r1z3-127.0.0.3:6030/sdb3 1
swift-ring-builder object-2.builder add r1z3-127.0.0.3:6030/sdb7 1
swift-ring-builder object-2.builder add r1z4-127.0.0.4:6040/sdb4 1
swift-ring-builder object-2.builder add r1z4-127.0.0.4:6040/sdb8 1
swift-ring-builder object-2.builder rebalance
swift-ring-builder container.builder create 10 3 1
swift-ring-builder container.builder add r1z1-127.0.0.1:6011/sdb1 1
swift-ring-builder container.builder add r1z2-127.0.0.2:6021/sdb2 1
swift-ring-builder container.builder add r1z3-127.0.0.3:6031/sdb3 1
swift-ring-builder container.builder add r1z4-127.0.0.4:6041/sdb4 1
swift-ring-builder container.builder rebalance
swift-ring-builder account.builder create 10 3 1
```

```

swift-ring-builder account.builder add r1z1-127.0.0.1:6012/sdb1 1
swift-ring-builder account.builder add r1z2-127.0.0.2:6022/sdb2 1
swift-ring-builder account.builder add r1z3-127.0.0.3:6032/sdb3 1
swift-ring-builder account.builder add r1z4-127.0.0.4:6042/sdb4 1
swift-ring-builder account.builder rebalance

```

The expected output is:

```

Device d0r1z1-127.0.0.1:6010R127.0.0.1:6010/sdb1_"" with 1.0 weight got id 0
Device d1r1z2-127.0.0.2:6020R127.0.0.2:6020/sdb2_"" with 1.0 weight got id 1
Device d2r1z3-127.0.0.3:6030R127.0.0.3:6030/sdb3_"" with 1.0 weight got id 2
Device d3r1z4-127.0.0.4:6040R127.0.0.4:6040/sdb4_"" with 1.0 weight got id 3
Reassigned 3072 (300.00%) partitions. Balance is now 0.00. Dispersion is now 0.00
Device d0r1z1-127.0.0.1:6010R127.0.0.1:6010/sdb1_"" with 1.0 weight got id 0
Device d1r1z2-127.0.0.2:6020R127.0.0.2:6020/sdb2_"" with 1.0 weight got id 1
Device d2r1z3-127.0.0.3:6030R127.0.0.3:6030/sdb3_"" with 1.0 weight got id 2
Device d3r1z4-127.0.0.4:6040R127.0.0.4:6040/sdb4_"" with 1.0 weight got id 3
Reassigned 2048 (200.00%) partitions. Balance is now 0.00. Dispersion is now 0.00
Device d0r1z1-127.0.0.1:6010R127.0.0.1:6010/sdb1_"" with 1.0 weight got id 0
Device d1r1z1-127.0.0.1:6010R127.0.0.1:6010/sdb5_"" with 1.0 weight got id 1
Device d2r1z2-127.0.0.2:6020R127.0.0.2:6020/sdb2_"" with 1.0 weight got id 2
Device d3r1z2-127.0.0.2:6020R127.0.0.2:6020/sdb6_"" with 1.0 weight got id 3
Device d4r1z3-127.0.0.3:6030R127.0.0.3:6030/sdb3_"" with 1.0 weight got id 4
Device d5r1z3-127.0.0.3:6030R127.0.0.3:6030/sdb7_"" with 1.0 weight got id 5
Device d6r1z4-127.0.0.4:6040R127.0.0.4:6040/sdb4_"" with 1.0 weight got id 6
Device d7r1z4-127.0.0.4:6040R127.0.0.4:6040/sdb8_"" with 1.0 weight got id 7
Reassigned 6144 (600.00%) partitions. Balance is now 0.00. Dispersion is now 0.00
Device d0r1z1-127.0.0.1:6011R127.0.0.1:6011/sdb1_"" with 1.0 weight got id 0
Device d1r1z2-127.0.0.2:6021R127.0.0.2:6021/sdb2_"" with 1.0 weight got id 1
Device d2r1z3-127.0.0.3:6031R127.0.0.3:6031/sdb3_"" with 1.0 weight got id 2
Device d3r1z4-127.0.0.4:6041R127.0.0.4:6041/sdb4_"" with 1.0 weight got id 3
Reassigned 3072 (300.00%) partitions. Balance is now 0.00. Dispersion is now 0.00
Device d0r1z1-127.0.0.1:6012R127.0.0.1:6012/sdb1_"" with 1.0 weight got id 0
Device d1r1z2-127.0.0.2:6022R127.0.0.2:6022/sdb2_"" with 1.0 weight got id 1
Device d2r1z3-127.0.0.3:6032R127.0.0.3:6032/sdb3_"" with 1.0 weight got id 2
Device d3r1z4-127.0.0.4:6042R127.0.0.4:6042/sdb4_"" with 1.0 weight got id 3
Reassigned 3072 (300.00%) partitions. Balance is now 0.00. Dispersion is now 0.00

```

Note that 3 object rings are created in order to test storage policies and EC in the SAIO environment. The EC ring is the only one with all 8 devices. There are also two replication rings, one for 3x replication and another for 2x replication, but those rings only use 4 devices.

#### Step 10.7: Verify the unit tests run

```
$HOME/swift/.unittests
```

Note that the unit tests do not require any Swift daemons running.

**Step 10.8: Start the “main” Swift daemon processes (proxy, account, container, and object)**

```
startmain
```

(The “**Unable to increase file descriptor limit. Running as non-root?**” warnings are expected and ok.)

**Step 10.9: Get an X-Storage-Url and X-Auth-Token**

```
curl -v -H 'X-Storage-User: test:tester' -H 'X-Storage-Pass: testing' http://127.0.0.1:8080/auth/v1.0
```

**Step 10.10: Check that you can GET account**

```
curl -v -H 'X-Auth-Token: <token-from-x-auth-token-above>' <url-from-x-storage-url-above>
```

**Step 10.11: Check that swift command provided by the python-swiftclient package works**

```
swift -A http://127.0.0.1:8080/auth/v1.0 -U test:tester -K testing stat
```

**Step 10.12: Verify the functional tests run**

```
$HOME/swift/.functests
```

(Note: functional tests will first delete everything in the configured accounts.)

**Step 10.13: Verify the probe tests run**

```
$HOME/swift/.probetests
```

(Note: probe tests will reset your environment as they call resetswift for each test.)



- **Troubleshooting for swift installation:**

Following are some useful tips for troubleshooting while installing OpenStack Swift:

- 1) You might encounter some errors related to some installed packages. The error might be because of some compatibility issues with the installed version of that package. In such cases, clone that package's latest code from github and then again install that package.  
**(Installing from apt repository won't help.)**
- 2) Whenever you encounter any permission related errors, check that in **/var/run/swift** directory, all the directories' and their **sub-directories**' owner is same as the user on which you are working.
- 3) To remove some errors, you might have to make some files under **/var/run/swift** executable.
- 4) More tips about troubleshooting can be found on the following link:

[https://docs.openstack.org/swift/latest/development\\_saio.html#debugging-issues](https://docs.openstack.org/swift/latest/development_saio.html#debugging-issues)