Welcome!

Foundations in Digital Development (for the Cloud)

Introduction

Foundations in Digital Development (for the Cloud)

Pre-Work

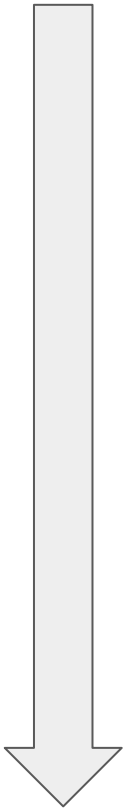
- Install `Git`: https://git-scm.com/downloads
- Install `Python 3`: https://www.python.org/downloads/
- Install `Atom Text Editor`: https://atom.io
- Install `Putty`: https://www.putty.org/
- Install `Docker Desktop`: https://docs.docker.com/v17.09/engine/installation/#supported-platforms
- Install AWS CLI: https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html
- Sign-up to `Github`: https://github.com/
- Sign-up to `Docker Hub`: https://id.docker.com/login/

What Might be Your Moonshot?

Learning Outcomes

1. Set-up a fit for purpose Intelligent Development Environment (IDE).
2. Write and execute basic programs in `Python`.
3. Use a Version Control System through making requests using `Git Bash`.
4. Clone and make changes to a pre-prepared website.
5. Log-in to an `AWS` environment.
6. Set-up security groups and access policies for cloud services.
7. Upload content to an `AWS S3` bucket and view via a URL.
8. Launch an `AWS EC2` instance, work with AMI's, establish secure connections and configure a web server.
9. Make requests to a `REST API`.
10. Read `JSON` and work with the response payload.
11. Create and deploy a simple web application.

Course Outline



Preparation 09:00-09:30	• Dependencies • Your moonshot idea!
The Development Ecosystem 09:30-12:15	• Code and language support • Intelligent development environments • Syntax highlighting and debugging • First-steps in programming • Libraries, packages and re-factoring • Version control systems
12:15-12:50	• Lunch
Cloud Services 12:50-14:30	• Overview of providers and typical services available • Compute and storage • Security and network protocols
Interfaces 14:30-17:00	• Application programmatic interfaces • Making simple `API` requests • Create and deploy a simple web application
Debrief 17:00-17:20	• Takeaway's • Summary

Preparation 09:00-09:30	• Dependencies • Your moonshot idea!
The Development Ecosystem 09:30-12:15	• Code and language support • Intelligent development environments • Syntax highlighting and debugging • First-steps in programming • Libraries, packages and re-factoring • Version control systems
12:15-12:50	• Lunch
Cloud Services 12:50-14:15	• Overview of providers and typical services available • Compute and storage • Security and network protocols
14:15-14:30	• Break
Interfaces 14:30-17:00	• Application programmatic interfaces • Making simple `API` requests • Create and deploy a simple web application
Debrief 17:00-17:20	• Takeaway’s • Summary

Code and Language Support

Foundations in Digital Development (for the Cloud)

Python Basics

- No need to declare variable types
- External variables need importing
- Uses indentation to declare sub-processes
- Processes run sequentially
- Processes run at latest opportunity
- Supports definition of classes and class variables
- Logic statements often applied
- Often used for analytics and server-side processes (though very versatile)
- Often run in a `virtual environment` using `pip` for package management

```
String = “Hello world”

if Python == Python 3:
     Print(String)
Out: “Hello world”

elif Python == Python 2:
     Print String
Out: “Hello World”

List = [5,6,7,8,9,11]
List[0] + List[4]
Out: 14

Tuple = (“hello”, “world”)
Tuple[1]
Out: “world”
```

Example code: https://gitlab.arup.com/technology-costmodelling/lifecycle-cost/building.py

Javascript Basics

- No need to declare variable types
- External variables need importing
- Uses brackets to declare variables
- Declaration of sub-processes end with “;”
- Processes often run asynchronously
- Supports definition of class variables
- Output typically logged or alerted, rather than printed in terminal
- Logic statements often applied
- Often used to handle `JSON` payloads with HTML interfaces (though very versatile)
- Often run in `Node.js` using `NPM` for package management

```
String = “Hello world”;
console.log(String);
Out: “Hello world”

List = [5,6,7,8,9,11];
console.log(List[0] + List[4]);
Out: 14

words: {
     “first”: “Hello”,
     “second”: “world”
};
console.log(words.second);
Out: “world”
```

Example code: https://gitlab.arup.com/technology-IAM/cofRN/App.js

HTML Basics

- Renders static content
- Uses a system of dividers, containers and elements
- Elements may be inherent or imported
- Containers and elements may have declared properties and styles
- Indentation is used to declare element hierarchy.
- `HTML` is able to render `JSON` as text

```
<section>
     <body
     display=”block”
     margin=”8px”>
          Hello world
     </body>
     <table
     style="width:100%">
          <tr>
               <th>0</th>
               <th>1</th>
               <th>2</th>
          </tr>
          <tr>
               <td>5</td>
               <td>6</td>
               <td>7</td>
          </tr>
     </table>
</section>
```

Example code: https://gitlab.arup.com/technology-bus-methodology/bus-website/index.html

Other Common Languages

`C`,

`C++`,

`C#`,

`Java`,

`Scala`,

`VisualBasic`,

`Fortran`,

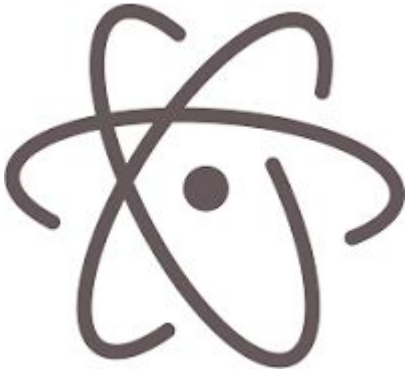
`Matlab`, ...

Syntax Highlighting and Debugging

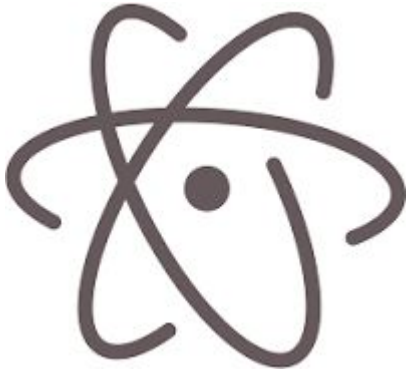
Foundations in Digital Development (for the Cloud)

Demonstration

vs

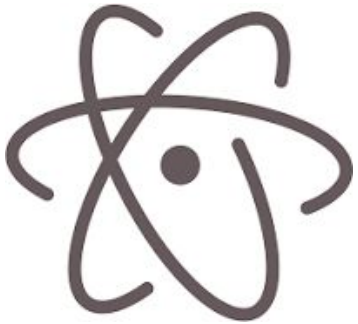


Some Text Editors



Activity

1. Set-up `Python` language support for `Atom`: https://atom.io/packages/ide-python
2. Set-up `HTML` language support for `Atom`: https://atom.io/packages/ide-html
3. Set-up `Jupyter Notebook` file support for `Atom`: https://atom.io/packages/jupyter-notebook
4. Explore Atom project folder, file management, file and terminal views



Python

```
import numpy as np

import plotly.plotly as py
import plotly.graph_objs as go

import math, random

size_of_factor_sample = 1000

number_of_element_cost_samples = 1000

class Building():

    def __init__(self, wb, name):
        self.name = name
        self.ws = wb.get_sheet_by_name(self.name)

    def build_data_for_building_lifetime(self, building_life):
        ws = self.ws
        self.dRate = ws["G3"].value*ws["G5"].value+ws["G4"].value+ws["G2"].value

        element_count = ws.max_row - 16

        self.elements_cost_for_building_lifes = []

        for i_sample in range(size_of_factor_sample):

            self.elements_cost_for_building_life = []

            for i_element in range(element_count):
                row = i_element+17
```

Javascript

```
submitUser = () => {
  (async () => {
    console.log("Awaiting mutation");
    console.log(this.state.currentUser);
    try {
      const result = await client.mutate({
        mutation: gql(createUser),
        variables: {
          input: {
            cognitoId: this.state.currentUser,
            userId: this.state.hex_md5v,
            username: this.state.currentUser,
            userType: "occupant",
          }
        }
      });
    } catch(error) {console.log(error)}
    console.log(result.data.createUser);
  })();
}

currentUsersResponse = () => {
  (async () => {
    console.log("Calling API");
    console.log(this.state.currentUser);
    const currentUsers =  await client.query({
        query: gql(allUser),
        variables: { cognitoId: this.state.currentUser }
    });
    console.log("currentUsers Response");
    console.log(currentUsers.data.allUser);
    if (currentUsers.data.allUser.length == 0) {
```

HTML

```
<React.Fragment>
  <Button
    iconLeft
    style={{ text: { color:"white", font: "Helvetica Neue", fontSize:15, fontWeight: "bold" },  container: { backgr
    text="Add Feedback"
    upperCase={false}
    icon="add"
    onPress={()=> this.submitConversation() }>
  </Button>
  <View style={{ margin: 20 }}>
    <Text style={{ color:"black", font: "Helvetica Neue", fontSize:15, fontWeight: "bold", margin: 5, flexWrap: "wra
      AN IMPRESSION
    </Text>
    <Form
      ref="form">
      <View
        style={styles.container}>
        <TextInput
          type="TextInput"
          name="feedbackTextInput"
          id="feedbackTextInput"
          multiline={true}
          style={{ color:"black", font: "Helvetica Neue", fontSize:15, margin: 20 }}
          onChangeText={(impression) => {this.setState({impression})} }>
          *
        </TextInput>
      </View>
    </Form>
  </View>
  <View style={styles.wrapper}>
    <Button
      style={{text: { color:"white", font: "Helvetica Neue", fontSize:15, fontWeight: "bold" },  container: { height
```

Activity

1. Write a programme to `print` “Hello, world” in `Python` using command line.
2. Update programme to `print` current date instead using `datetime Python` module.

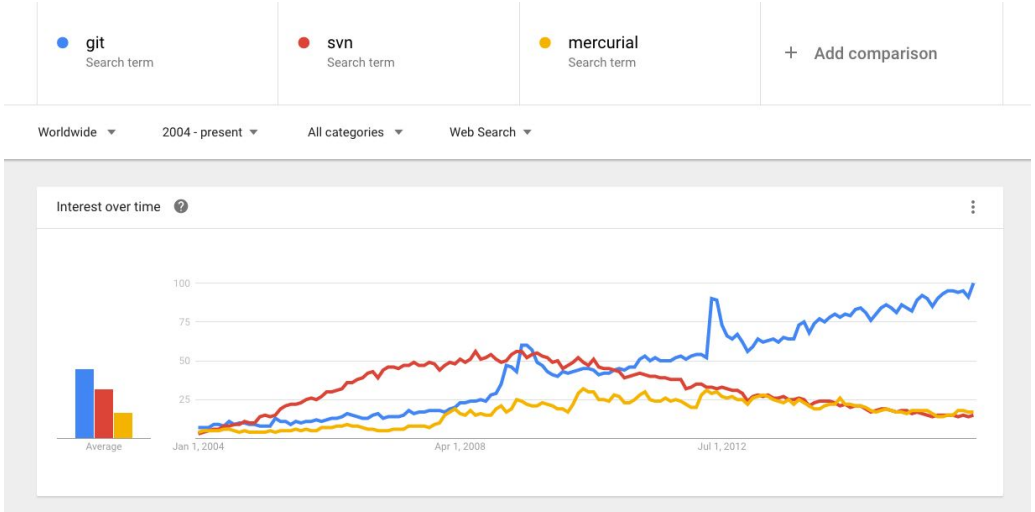
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import printer
>>> do = printer.Printer()
>>> do.hello_world()
Hello, world!
>>> do.time_now()
26/04/2019, 09:32:35
>>>
```

Version Control Systems

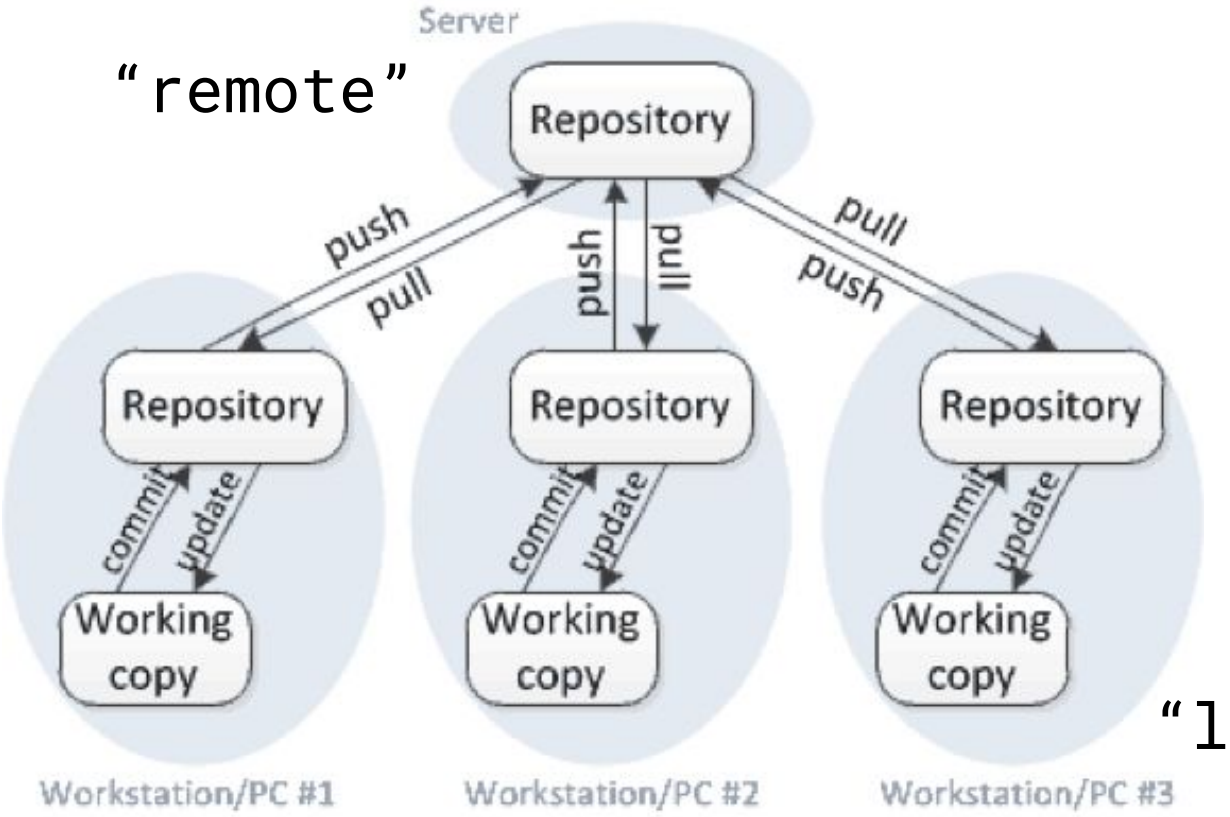
Foundations in Digital Development (for the Cloud)

System Software

- Tracks changes to computer files and coordinating work on those files among multiple agents.
- Primarily used for source code management in software development
- May also be used to keep track of changes to any set of files.



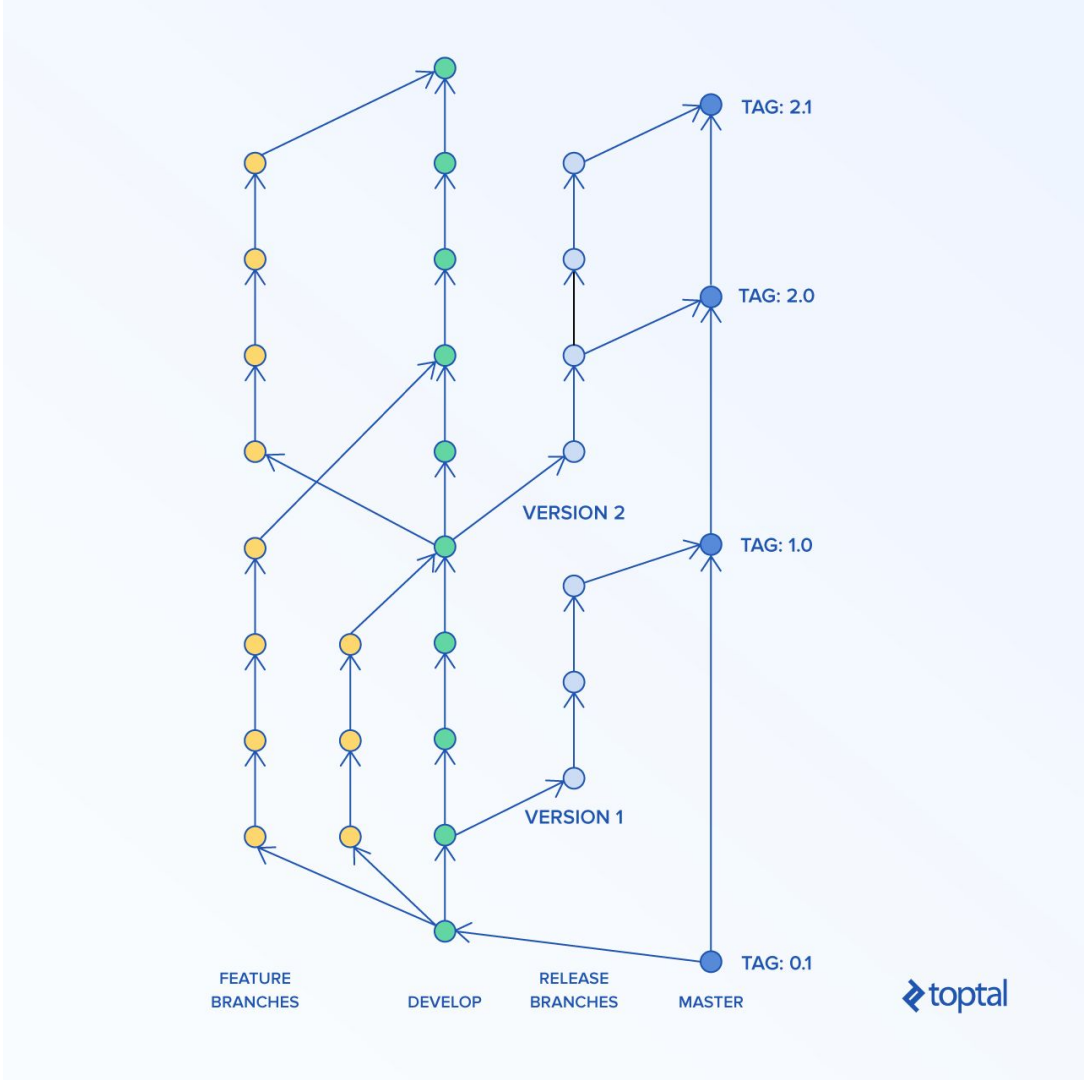
The Git Version Control System



“remote”

“local”

Branching


TAG: 2.1
TAG: 2.0
VERSION 2
TAG: 1.0
VERSION 1
TAG: 0.1
FEATURE BRANCHES
DEVELOP
RELEASE BRANCHES
MASTER
toptal

For Use Today

GitLab

https://gitlab.arup.com

https://github.com/

Activity

- `Add` trainee's to lesson `Github` project
- `Clone` course repository from https://github.com/ArupAus/digital-foundations-cloud.git
- `Add` trainee's "Hello World" python programmes developed previously.
- `Commit` local changes.
- `Push` changes to new feature branch.
- `Fetch` latest changes on remote.
- `Checkout` another trainee's feature branch.
- Make changes to "my-first-page.html".
- `Push` changes to new feature branch of `remote`.

Question

- Do you think that a `Git` Version Control System could be employed on any of your current projects to keep track of a shared set of files?

Preparation 09:00-09:30	• Dependencies • Your moonshot idea!
The Development Ecosystem 09:30-12:15	• Code and language support • Intelligent development environments • Syntax highlighting and debugging • First-steps in programming • Libraries, packages and re-factoring • Version control systems
12:15-12:50	• Lunch
Cloud Services 12:50-14:15	• Overview of providers and typical services available • Compute and storage • Security and network protocols
14:15-14:30	• Break
Interfaces 14:30-17:00	• Application programmatic interfaces • Making simple `API` requests • Create and deploy a simple web application
Debrief 17:00-17:20	• Takeaway's • Summary

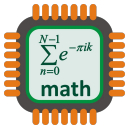
Cloud Services

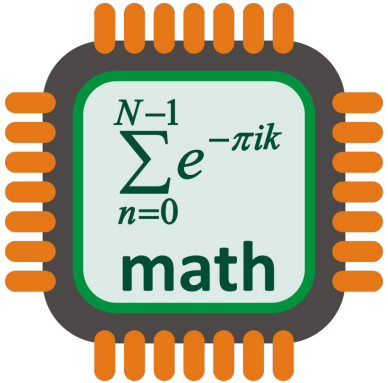
Foundations in Digital Development (for the Cloud)

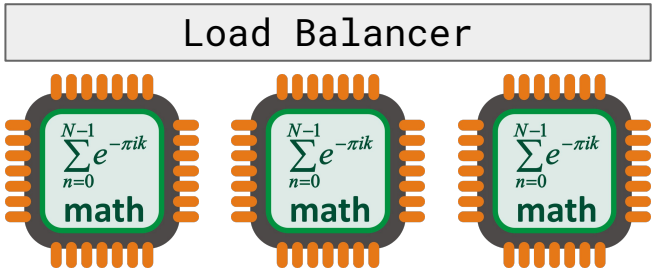
Major Cloud Service Providers

Common Cloud Services

- Compute

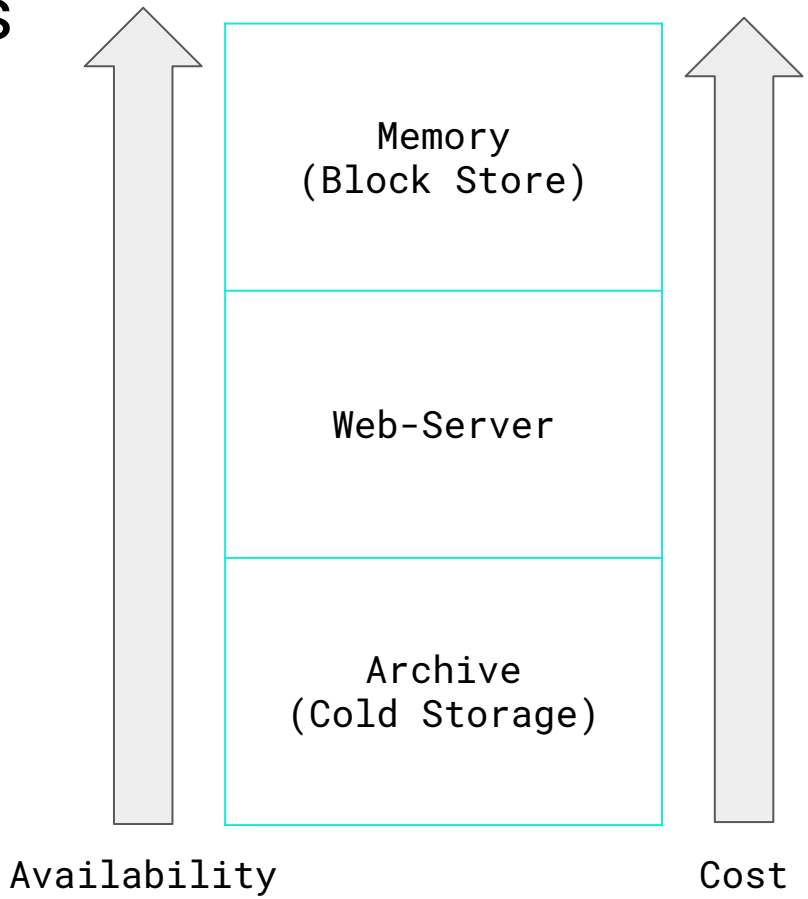






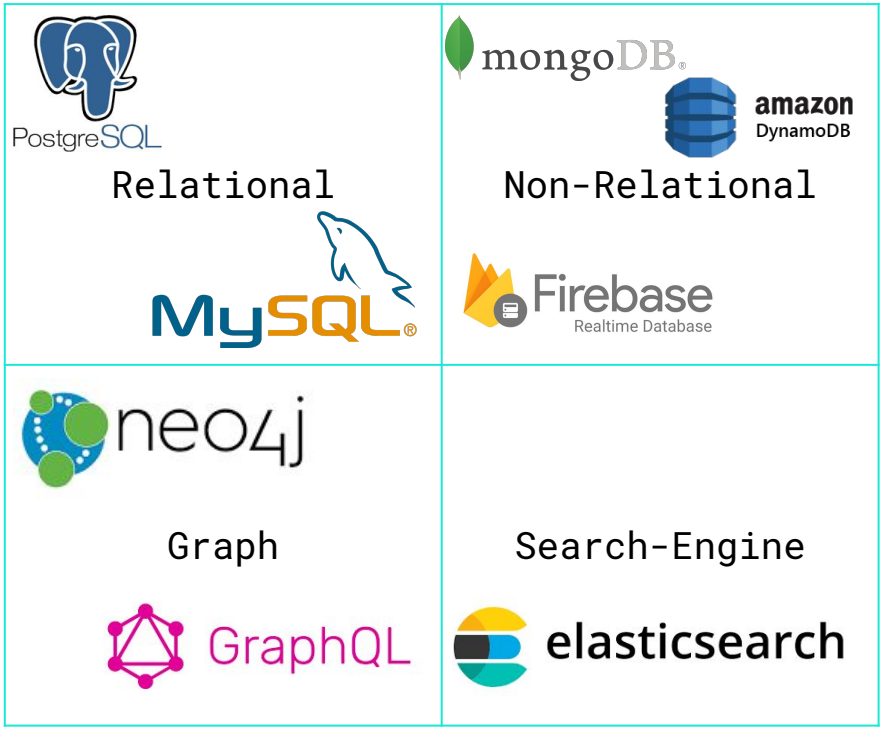
Common Cloud Services

- Compute
- Storage



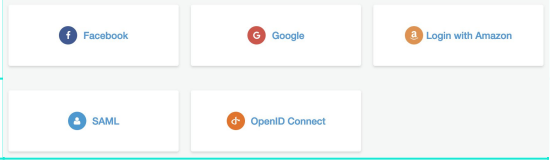
Common Cloud Services

- Compute
- Storage
- Database



Common Cloud Services

- Compute
- Storage
- Database
- Access Management

Identity and Access Management	**User Authentication**
Users (Human)	Application users
UI and/or Programmatic	May or may not be federated
Roles (Human or Resource)	
Policies	

Common Cloud Services

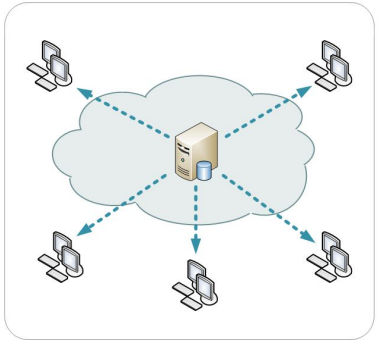
- Compute
- Storage
- Database
- Access Management
- Security

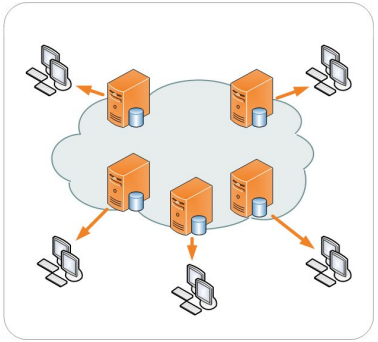
Virtual Private Cloud (VPC) designation and peering
Restrict outbound/inbound traffic by IP address/protocol/geography eg. TCP from 193.17.187.246
Data encryption
Message encryption
Key/secrets management
Request rate limiting

Common Cloud Services

- Compute
- Storage
- Database
- Access Management
- Security
- Distribution

DNS Records	**Content Delivery**
Name Servers	Distributed Networks
Mail Exchange	Security Automation
C Names	Monitoring
A Names	





Common Cloud Services

- Compute
- Storage
- Database
- Access Management
- Security
- Distribution
- Management and Governance

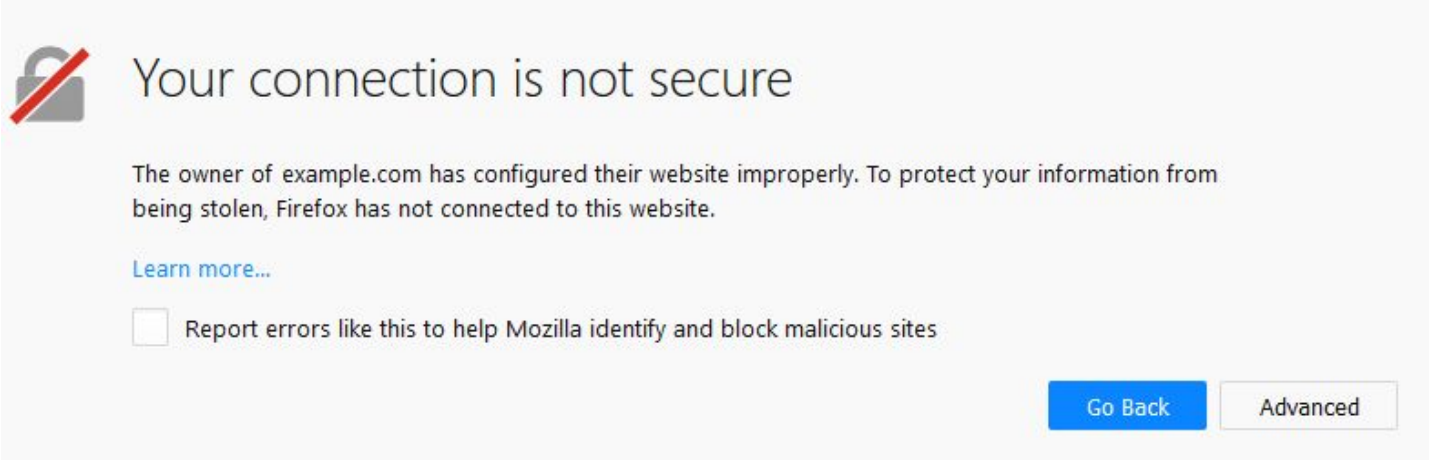
Logging	Alarms
Accounting	Optimisation

Activity

- Log in to `AWS Console` at https://aws.arup.com
- Configure AWS CLI for programmatic access.
- Create an `S3` bucket and upload website created earlier
- Configure the bucket policy to allow access from current `IP address`.
- View website content on web browser via `S3 URL`.
- Create Amazon Linux `EC2` instance and log in using key file and `Putty`.
- Explore instance directory structure.
- Install `Git` and `Python 3` using `yum install` commands.
- `Clone` course repository to `EC2` instance and checkout trainee's feature branch.
- Run `Hello World` programme using `Python` on instance.
- Install and start the LAMP Web Server on instance by following this guide: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/install-LAMP.html
- Copy "my-first-page.html" from instance version of repository and add to directory "/var/www/html"

Question

- The web content you have made available is accessed using `HTTP` protocol. Why shouldn't we trust this web content?
- How could we ensure that such web content can be trusted?



Preparation 09:00-09:30	• Dependencies • Your moonshot idea!
The Development Ecosystem 09:30-12:15	• Code and language support • Intelligent development environments • Syntax highlighting and debugging • First-steps in programming • Libraries, packages and re-factoring • Version control systems
12:15-12:50	• Lunch
Cloud Services 12:50-14:15	• Overview of providers and typical services available • Compute and storage • Security and network protocols
14:15-14:30	• Break
Interfaces 14:30-17:00	• Application programmatic interfaces • Making simple `API` requests • Create and deploy a simple web application
Debrief 17:00-17:20	• Takeaway's • Summary

Application Programmatic Interfaces

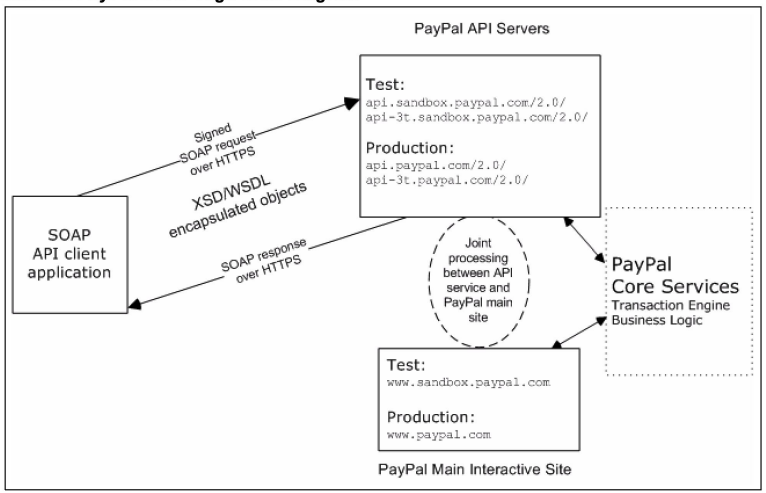
Foundations in Digital Development (for the Cloud)

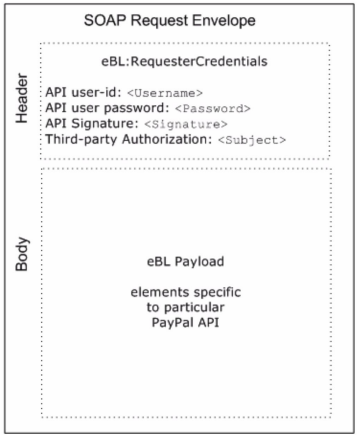
Question

- Do applications need store and manage all dependent data themselves?

SOAP

***FIGURE 1.1** PayPal SOAP High-level Diagram*





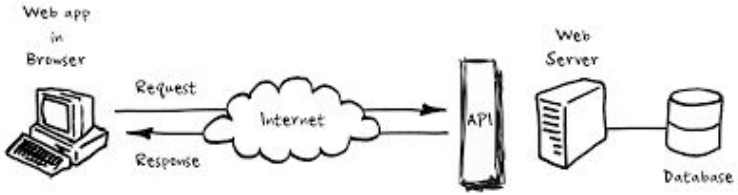
PayPal SOAP API Architecture

SOAP Implementation

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
     xmlns:cc="urn:ebay:apis:CoreComponentTypes"
     xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
     xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
     xmlns:ebl="urn:ebay:apis:eBLBaseComponents"
     xmlns:ns="urn:ebay:api:PayPalAPI">
     <SOAP-ENV:Header>
        <Security
           xmlns="http://schemas.xmlsoap.org/ws/2002/12/secext"
           xsi:type="wsse:SecurityType"
        />
        <RequesterCredentials xmlns="urn:ebay:api:PayPalAPI"
           xsi:type="ebl:CustomSecurityHeaderType">
           <Credentials
              xmlns="urn:ebay:apis:eBLBaseComponents"
              xsi:type="ebl:UserIdPasswordType"
           />
        </RequesterCredentials>
     </SOAP-ENV:Header>
     <SOAP-ENV:Body id="_0">
        <specific_api_name_Response xmlns="urn:ebay:api:PayPalAPI">
           <Timestamp xmlns="urn:ebay:api:PayPalAPI">
              dateTime_in_UTC/GMT
           </TIMESTAMP>
           <Ack xmlns="urn:ebay:apis:eBLBaseComponents">Success</Ack>
           <Version xmlns="urn:ebay:apis:eBLBaseComponents">
              serviceVersion
           </Version>
           <CorrelationId xmlns="urn:ebay:apis:eBLBaseComponents">
              applicationCorrelation
           </CorrelationID>
           <Build xmlns="urn:ebay:apis:eBLBaseComponents">
              api_build_number
           </Build>
           <elements_for_specific_api_response>
              data
           </elements_for_specific_api_response>
        </specific_api_name_Response>
     </SOAP-ENV:Body>
```

REST

[get, put, post, patch, delete]

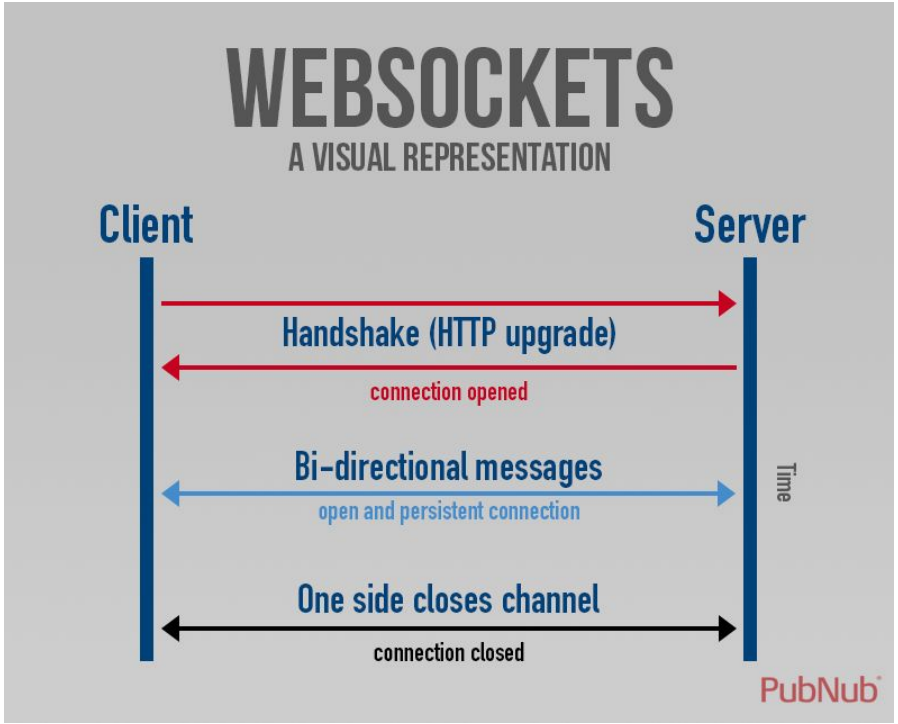


fetch(“https://api.mapbox.com/geocoding/v5/mapbox.places/-122.406417,37.785834.json?access_token=pk.eyJ1IjoiYWlkYW5wYXJraW5zb24iLCJhIjoiY2prOXZscDVrMnIwMzNrbzJyYmh0dHV3MiJ9.HtB81oxPLG6olZEn8J3Gdg”)

{"type":"FeatureCollection","query":[-122.406417,37.785834],"features":
[{"id":"poi.25769853785","type":"Feature","place_type":
["poi"],"relevance":1,"properties":{"landmark":true,"category":"jewelry,
watches, accessories, shop","address":"833 Market St Ste
508"},"text":"Zaki's Jewelry","place_name":"Zaki's Jewelry, 833 Market St
Ste 508, San Francisco, California 94102, United States","center":
[-122.40647,37.78586],"geometry":{"coordinates":
[-122.40647,37.78586],"type":"Point"},"context":
[{"id":"neighborhood.293547","text":"Downtown"},
{"id":"postcode.644855979161210","text":"94102"},
{"id":"place.15734669613361910","wikidata":"Q62","text":"San Francisco"},
{"id":"region.11319063928738010","short_code":"US-
CA","wikidata":"Q99","text":"California"},
{"id":"country.9053006287256050","short_code":"us","wikidata":"Q30","text"
:"United States"}]},
{"id":"neighborhood.293547","type":"Feature","place_type":
["neighborhood"],"relevance":1,"properties":
{},"text":"Downtown","place_name":"Downtown, San Francisco, California
94102, United States","bbox":
[-122.420691,37.782213,-122.403401,37.792311],"center":
[-122.4112,37.7881],"geometry":{"type":"Point","coordinates":
[-122.4112,37.7881]},"context":
[{"id":"postcode.644855979161210","text":"94102"},
{"id":"place.15734669613361910","wikidata":"Q62","text":"San Francisco"},
{"id":"region.11319063928738010","short_code":"US-
CA","wikidata":"Q99","text":"California"},
{"id":"country.9053006287256050","short_code":"us","wikidata":"Q30","text"
:"United States"}]},
{"id":"postcode.644855979161210","type":"Feature","place_type":
["postcode"],"relevance":1,"properties":
{},"text":"94102","place_name":"San Francisco, California 94102, United
States","bbox":
[-122.429931001842,37.7694394579571,-122.40474104605,37.7892279476408],"ce
nter":[-122.42,37.78],"geometry":{"type":"Point","coordinates":
[-122.42,37.78]},"context":
[{"id":"place.15734669613361910","wikidata":"Q62","text":"San Francisco"},
{"id":"region.11319063928738010","short_code":"US-
CA","wikidata":"Q99","text":"California"},
{"id":"country.9053006287256050","short_code":"us","wikidata":"Q30","text"
:"United States"}]},
{"id":"place.15734669613361910","type":"Feature","place_type":
["place"],"relevance":1,"properties":{"wikidata":"Q62"},"text":"San
Francisco","place_name":"San Francisco, California, United States","bbox":
[-122.517910874663,37.6044780500533,-122.354995082683,37.8324430069081],"c
enter":[-122.463,37.7648],"geometry":{"type":"Point","coordinates":
[-122.463,37.7648]},"context":
[{"id":"region.11319063928738010","short_code":"US-
CA","wikidata":"Q99","text":"California"},
{"id":"country.9053006287256050","short_code":"us","wikidata":"Q30","text"
:"United States"}]},
{"id":"region.11319063928738010","type":"Feature","place_type":
["region"],"relevance":1,"properties":{"short_code":"US-
CA","wikidata":"Q99"},"text":"California","place_name":"California, United
States","bbox":[-124.581979,32.454411,-114.131211,42.009517],"center":
[-120,37],"geometry":{"type":"Point","coordinates":[-120,37]},"context":
[{"id":"country.9053006287256050","short_code":"us","wikidata":"Q30","text
":"United States"}]},
{"id":"country.9053006287256050","type":"Feature","place_type":
["country"],"relevance":1,"properties":
{"short_code":"us","wikidata":"Q30"},"text":"United
States","place_name":"United States","bbox":

Web-Sockets

[publish, subscribe]



Activity

- `Checkout` master branch of training course repository.
- Create a virtual environment.
- Activate the virtual environment.
- Use `pip` to install project dependencies listed in “requirements.txt”.
- Launch `jupyter notebook`.
- Run notebook “my-first-request.ipynb” to make a RESTful GET request to an Arup Elasticsearch domain containing sensor data.
- Change notebook environment variables to retrieve most recent “temperature” record.
- Change notebook environment variables and code to retrieve a time-stamped 10-minute period of “temperature” records and `print` as a `pandas DataFrame`.

Question

- First to complete the task is to volunteer an explanation of the `JSON` response payload received from the `Elasticsearch` domain and how to query parts of it.

Activity

- Login to `Docker` account and create a `Docker` image
- Tag `Docker` image and push to `AWS Elastic Container Repository (ECR)` https://docs.aws.amazon.com/AmazonECR/latest/userguide/docker-basics.html#use-ecr
- Start a `Fargate` cluster using `AWS Elastic Container Service (ECS)` https://docs.aws.amazon.com/AmazonECS/latest/userguide/create_cluster.html
- Launch an `Application Load Balancer` and `Target Group` https://docs.aws.amazon.com/AmazonECS/latest/userguide/create-application-load-balancer.html
- Launch an `AWS Elastic Container Service` https://docs.aws.amazon.com/AmazonECS/latest/userguide/create-service.html
- Distribute service over `AWS Cloudfront` https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/distribution-working-with.html
- Interact with service at `[resource].cloudfront.net`

Preparation 09:00-09:30	• Dependencies • Your moonshot idea!
The Development Ecosystem 09:30-12:15	• Code and language support • Intelligent development environments • Syntax highlighting and debugging • First-steps in programming • Libraries, packages and re-factoring • Version control systems
12:15-12:50	• Lunch
Cloud Services 12:50-14:15	• Overview of providers and typical services available • Compute and storage • Security and network protocols
14:15-14:30	• Break
Interfaces 14:30-17:00	• Application programmatic interfaces • Making simple `API` requests • Create and deploy a simple web application
Debrief 17:00-17:20	• Takeaway's • Summary

Activity

- Explain two key learning outcomes that you will take away from the course to each other in pairs.

Learning Outcomes

1. Set-up a fit for purpose Intelligent Development Environment (IDE).
2. Write and execute basic programs in `Python`.
3. Use a Version Control System through making requests using `Git Bash`.
4. Clone and make changes to a pre-prepared website.
5. Log-in to an `AWS` environment.
6. Set-up security groups and access policies for cloud services.
7. Upload content to an `AWS S3` bucket and view via a URL.
8. Launch an `AWS EC2` instance, work with AMI's, establish secure connections and configure a web server.
9. Make requests to a `REST API`.
10. Read `JSON` and work with the response payload.
11. Create and deploy a simple web application.