A REPORT

On

atmos: CLOUD-BASED WEATHER FORECASTING MOBILE APPLICATION

By:

Divydeep Agarwal

Mi Zhou

Ziwen Li

Kuljeet Billing

(TEAM 31)

UNIVERSITY OF SOUTHERN CALIFORNIA

LOS ANGELES

NOVEMBER 15, 2015

1. ABSTRACT

Weather is an important part of all of our lives. So it's important to have the most up to date weather. Atmos is a new way of showing weather data, based entirely on Forecast, the open-source weather API. The app provides the most useful, and hyper-localized forecasts, conditions, and alerts in real-time directly to the user. The app provides current conditions for your exact location or you can search for weather in any neighborhood across the US. The user gets information for the current conditions, hour-by-hour conditions as well as a 7-day forecast.

2. DESCRIPTION

The Atmos app consists of two screens. The first screen gives user to search for any address and get the weather details for that location. User can also find weather information for his current location too from this screen. The second screen presents all the weather details which is classified into three different tabs. The first tab provides current weather details. The second tab provides 48-hour weather information and the third tab provides 7-day weather forecast. In case of emergency alert or extreme advisory conditions, the app provides push notifications about the alerts through National Weather Service warnings for the United States. Moreover, clicking on the weather parameter icon provides description about that parameter. To alert friends and family about weather conditions, users can share forecasts to Facebook as well as post weather images.

2.1 Description of Search Activity

The search activity is the first activity of the application where the user can enter an address (street, city, and state). The City field includes a list of all US cities from Google. The State field includes a list of all US States. The default temperature unit is Fahrenheit.

The search activity has three buttons-

- **Search button**: This button validates whether the user provided values for street address, city and state. If the user did not enter one of the data items, then an alert is shown with an appropriate message prompting the user to provide complete information. Once the user has provided valid data, the app sends a request to the server API with the data in JSON format. The app uses the Volley library to send a POST request to transfer data to the server. The server will grab the data and send it to Google Geocode web service to get the latitude and longitude.
- **Clear button**: This button clears the result area, all text fields, unselect the City and State values.
- **Weather for Current Location button**: This button uses the GPS and the Google Maps services to get the current location and translate it into latitude and longitude. The app then sends this data to the server which makes a POST request to Forecast.io web service to get the weather information.

All the fields are mandatory to fill before searching for any address. The Street field is a text field (EditText in Android) and the City and State fields have a dropdown selector (Spinner in Android). The labels are created using TextView in Android. All of these objects, including the three buttons are wrapped inside a LinearLayout.

2.2 Description of Detail Activity

The detail activity is used to provide results for the searched location or the current location. It displays three tabs viz. NOW, HOURLY and DAILY. The user can click on each icon of the parameter to get a detailed description about it.

- **NOW**: This tab displays the current weather information. It has the following parameters-

Parameter	Unit
Precipitation	Percentage
Chance of Rain	Percentage
Wind Speed	mph
Pressure	millibar
Humidity	Percentage
Visibility	miles
Temperature	Fahrenheit
Cloud Cover	Percentage

- **HOURLY**: This tab displays hourly weather over a 24-hour period starting from the current hour. It provides the following information-

Parameter	Unit
Summary	N/A
Time	12-hour time
Temperature	Fahrenheit
Humidity	Percentage
Wind Speed	mph
Cloud Cover	Percentage
Pressure	millibar

- **DAILY**: This tab displays the weather for a 7-day week. It has the following parameter for each day-

Parameter	Unit/Format
Summary	N/A
Time	12-hour time
Sunrise Time	12-hour time
Sunset Time	12-hour time

Max Temperature	Fahrenheit
Min Temperature	Fahrenheit
Chance of Rain	Percentage

2.3 Method Applied

In order to approach the problem and solve it, we divided the problem into the following parts:

- Frontend Development
- Backend Development
- Testing
- Cloud Deployment

The frontend is developed in Android using Java. It uses other SDKs such as Volley SDK for asynchronous requests to the backend server and Facebook SDK for sharing weather information on social media. The backend is developed using PHP v5.5 and deployed on Apache Tomcat server. The backend would handle all the requests from the app and return the requested data back to the server. The backend is deployed on AWS. We use Elastic Beanstalk to create two REST endpoints for our servers which provisions EC2 instances for them and handles the requests. To test the application, we use Device Farm to check the application display on various screen sizes and also measure response time. Moreover, we use the SNS service for sending emergency weather alerts to the user.

2.4 Workflow Plan

The high-level ER-diagram for the application is given in figure 1.

The work was divided among all the individuals of the team as follows-

- **Ziwen Li**: Android Frontend Development + Volley SDK + Facebook SDK
- **Kuljeet Billing**: Google Geocode API coding + SNS PUSH Notification Coding
- **Mi Zhou:** Forecast API Coding
- **Divydeep Agarwal**: Project Report + AWS Deployment + AWS Testing

3. TECHNICAL BACKGROUND

3.1 Android Studio

AndroidStudio is the official IDE for Android application development, based on IntelliJ IDEA. On top of the capabilities you expect from IntelliJ, Android Studio offers:

- Flexible Gradle-based build system
- Build variants and multiple apk file generation
- Code templates to help you build common app features
- Rich layout editor with support for drag and drop theme editing
- Lint tools to catch performance, usability, version compatibility, and other problems

- ProGuard and app-signing capabilities

We have used Android Studio as our default IDE for all the development needs for the application. We have used the Android SDK v22 and the Volley SDK for the project. It also uses the Google Play Service SDK v8.1 for the geolocation feature.

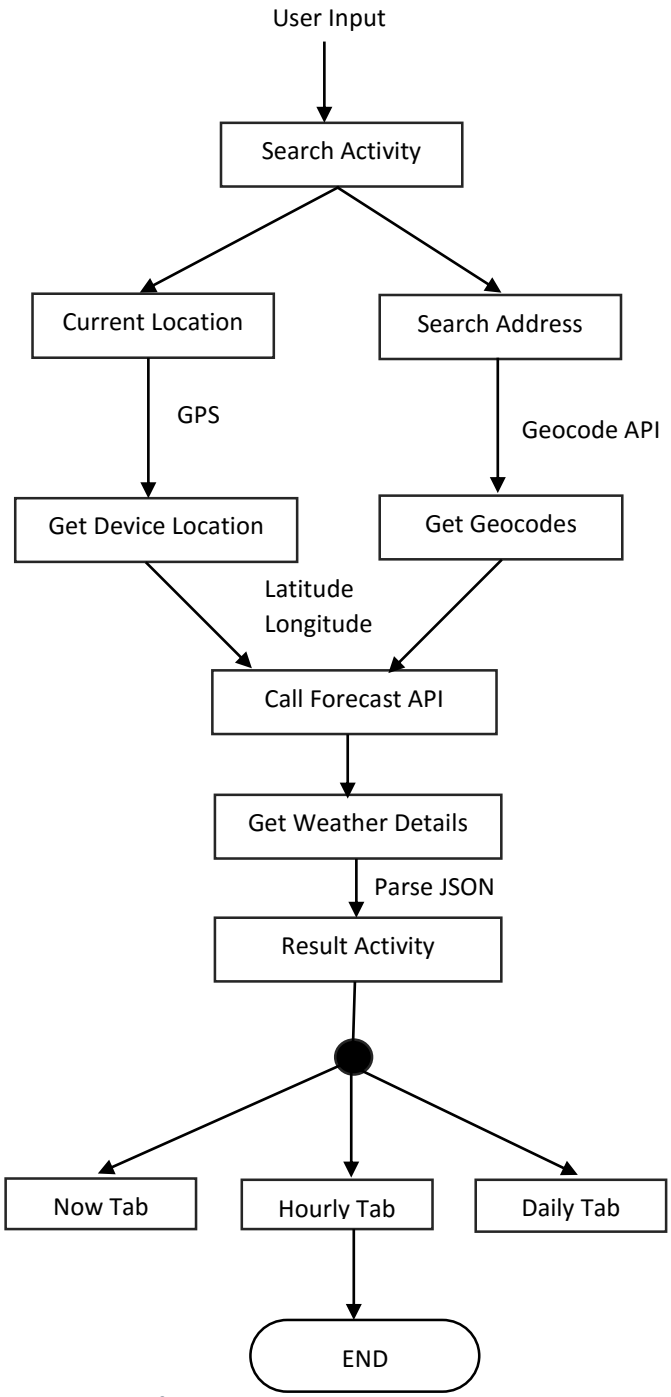


Figure 1 *ER Diagram for atmos*

3.2 Android

The Android OS is based on a Linux kernel, but provides a substantially different application abstraction than found in traditional Linux desktop and server distributions. Android applications are written in Java and compiled into a special DEX bytecode that executes in Android's Dalvik virtual machine. Applications may optionally contain native code components. Application functionality is divided into components. Android defines four types of components: activity, service, broadcast receiver, and content provider. The application's user interface is composed of a set of activity components. Service components act as daemons, providing background processing. Broadcast receiver components handle asynchronous messages. Content provider components are per-application data servers that are queried by other applications.

Application components communicate with one another using Binder inter-process communication (IPC). Binder provides message passing (called parcels) and thread management. Applications often interface with Binder indirectly using intent messages. The intent message abstraction is used for communication between activity and broadcast receiver components, as well as starting service components. Intent messages can be addressed to implicit action strings that are resolved by the Activity Manager Service (AMS). Intent messages and action strings allow end users and OEMs to customize the applications used to perform tasks.

We have leveraged Android as our base for the application and it supports Android v5.0 and above. Our app uses the following permissions from the OS:

- Internet Access
- Location Access
- Contacts Access

3.3 Google Maps Geocoding API

Geocoding is the process of converting addresses (like "1600 Amphitheatre Parkway, Mountain View, CA") into geographic coordinates (like latitude 37.423021 and longitude -122.083739), which can be used to place markers on a map, or position the map. The Google Maps Geocoding API provides a direct way to access this service via an HTTP request.

We use this API to convert the address searched by the user into geographic coordinates and use it find the weather information for the location.

3.4 Forecast.io API

The Forecast API lets us query for most locations on the globe, and returns:

- Current conditions
- Minute-by-minute forecasts out to 1 hour (where available)
- Hour-by-hour forecasts out to 48 hours
- Day-by-day forecasts out to 7 days

We use the current location geocodes or the one for the user-searched location and find its weather forecast.

3.5 Facebook Share API

The Facebook Share API provides a quick and simple way for users to post content from the app to Facebook. People can share content from the app to Facebook as URLs. We can enable link sharing with social plugins, with Share dialog or Feed dialog, or with your custom dialog that calls Graph API.

People can share different types of content from the web:

- Links - Share articles, photos, videos and other content as a URL.
- Status Updates - Share plain text status updates. These don't link to any specific content.
- Open Graph Stories - We can use Open Graph to let people publish rich, structured stories.

We use the Facebook Share Dialog to let users publish weather information for their current location. Users can share weather parameters such as temperature, precipitation, wind speed, cloud cover etc. to Facebook.

3.6 Amazon Web Services

AWS is Amazon's implementation of cloud computing. Included in AWS is Amazon Elastic Compute Cloud (EC2), which delivers scalable, pay-as-you-go compute capacity in the cloud, and AWS Elastic Beanstalk, an even easier way to quickly deploy and manage applications in the AWS cloud. You simply upload your application, and Elastic Beanstalk automatically handles the deployment details of capacity provisioning, load balancing, auto-scaling, and application health monitoring. Elastic Beanstalk is built using familiar software stacks such as the Apache HTTP Server, PHP, and Python, and Apache Tomcat. We have used the following services from Amazon-

- Amazon Elastic Compute Cloud (EC2) provides scalable virtual private servers using Xen.
- Amazon CloudFront, a content delivery network (CDN) for distributing objects to so-called "edge locations" near the requester
- AWS Elastic Beanstalk provides quick deployment and management of applications in the cloud.
- Amazon Simple Notification Service (SNS) provides a hosted multi-protocol "push" messaging for applications.
- Amazon Cognito a simple user identity and data synchronization service that helps you securely manage and synchronize app data for your users across their mobile devices.
- Device Farm is an app testing service that enables testing of Android apps on real, physical phones and tablets that are hosted by Amazon Web Services.

4. CLOUD SERVICES USED

4.1 Elastic Compute Cloud

Amazon EC2 provides scalable computing capacity in the Amazon Web Services cloud. Using Amazon EC2 eliminates the need to invest in hardware up front, so we can develop and deploy applications faster. We can use Amazon EC2 to launch as many or as few virtual servers as needed, configure security and networking, and manage storage. Amazon EC2 also enables to scale up or down to handle changes in requirements or spikes in popularity, reducing the need to forecast traffic. It also provides preconfigured templates for the instances, known as Amazon Machine

Images (AMIs), that package the bits needed for server (including the operating system and additional software).

We configure two EC2 instances for our app. One instance would provide the functionality to search for geocodes and the other would provide the functionality to get the weather data from Forecast API. Both the instances are exposed publicly and provide a REST API for POST requests.

4.2 Simple Notification Service

Amazon SNS is a web service that makes it easy to set up, operate, and send notifications from the cloud. It provides developers with a highly scalable, flexible, and cost-effective capability to publish messages from an application and immediately deliver them to subscribers or other applications. It is designed to make web-scale computing easier for developers.

Push notification services, such as APNS and GCM, maintain a connection with each app and associated mobile device registered to use their service. When an app and mobile device register, the push notification service returns a device token. Amazon SNS uses the device token to create a mobile endpoint, to which it can send direct push notification messages. In order for Amazon SNS to communicate with the different push notification services, the user submits the push notification service credentials to Amazon SNS to be used on his behalf.

Since we have made an Android application, we created a SNS topic called EmerAlerts which publishes notifications to the user through Google Cloud Messaging. The notifications consist of emergency/extreme weather alerts received from National Weather Service of United States.

4.3 CloudFront

Amazon CloudFront is a content delivery web service. It integrates with other Amazon Web Services products to give developers and businesses an easy way to distribute content to end users with low latency, high data transfer speeds, and no minimum usage commitments.

Amazon CloudFront can be used to deliver the entire website, including dynamic, static, streaming, and interactive content using a global network of edge locations. Requests for content are automatically routed to the nearest edge location, so content is delivered with the best possible performance. Amazon CloudFront is optimized to work with other Amazon Web Services, like Amazon S3, Amazon EC2, Amazon Elastic Load Balancing, and Amazon Route 53.

We used CloudFront to make our server REST APIs be easily accessible and reduce latency. We uploaded the backend scripts in a S3 bucket and granted read permission to everyone. We then created a CloudFront Distribution to allow everyone to access those scripts from any edge location. This enables best performance for the users.

4.4 Device Farm

Device Farm is an app testing service that enables us to test our Android, iOS, and Fire OS apps on real, physical phones and tablets that are hosted by Amazon Web Services (AWS). A test report containing high-level results, low-level logs, pixel-to-pixel screenshots, and performance data is updated as tests are completed. We created a project in Device Farm and tested our app on a pool of devices includes mobile phones and tablets. We tested the proper display and

responsiveness of screens on various devices and if there is any problem in the formatting or layout.

4.5 Elastic Beanstalk

AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS. We can simply upload our code and it automatically handles the deployment, from capacity provisioning, load balancing, auto-scaling to application health monitoring.

We launch a new application consisting of 2 environments with the following configuration:

- **Platform**: PHP
- **Environment Tier**: Web Server
- **Environment Type**: Load balancing, auto scaling
- **Environment 1 URL**: http://atmos-env-weather.elasticbeanstalk.com/
- **Environment 2 URL**: http://atmos-env-search.elasticbeanstalk.com/

4.6 Geocoding API

The PHP script on the server will use the address information (street, city and state) to construct a web service URL to query the Google Geocode API appropriately. If the Geocode web service is able to find the location successfully, it returns its coordinates back the app. The API request must be of the following form:

https://maps.googleapis.com/maps/api/geocode/output?parameters

where output may be either of the following values:

- json: indicates output in JavaScript Object Notation (JSON)
- xml: indicates output as XML

The required parameters in a geocoding request are-

- address: The street address that you want to geocode, in the format used by the national postal service of the country concerned.
- key: The application's API key. This key identifies your application for purposes of quota management.

https://maps.googleapis.com/maps/api/geocode/json?address=646+USC+McCarthy+Way,+Los+Angeles,+CA&key=AIzaSyDXSw-BpTGpQHJBFTi70FRdI1sw0jsqVKg

The sample output from the API is-


```
{
  "results" : [
    {
      "formatted_address" : "646 USC McCarthy Way, Los Angeles, CA 90007, USA",
      "geometry" : {
```

```
          "bounds" : {
            "northeast" : {
              "lat" : 34.0206153,
              "lng" : -118.2813871
            },
            "southwest" : {
              "lat" : 34.0206015,
              "lng" : -118.2813957
            }
          },
          "location" : {
            "lat" : 34.0206153,
            "lng" : -118.2813871
          },
          "location_type" : "RANGE_INTERPOLATED",
          "viewport" : {
            "northeast" : {
              "lat" : 34.0219573802915,
              "lng" : -118.2800424197085
            },
            "southwest" : {
              "lat" : 34.0192594197085,
              "lng" : -118.2827403802915
            }
          }
        },
        "place_id" :
"EjA2NDYgVVNDIE1jQ2FydGh5IFdheSwgTG9zIEFuZ2VsZXMsIENBIDkwMDA3LCBVU0E",
        "types" : [ "street_address" ]
      }
    ],
    "status" : "OK"
}
```


4.7 Forecast API:

The API provides accurate short-term and long-term weather prediction for any searched location. The API uses a simple JSON interface. Forecast.io is backed by a wide range of data sources, which are aggregated together statistically to provide the most accurate forecast possible for a given location.

The Forecast API is backed by the following data sources:

- Dark Sky's own hyperlocal precipitation forecasting system (id darksky)
- The USA NOAA's LAMP system (USA, id lamp).
- The UK Met Office's Datapoint API (UK, id datapoint).
- The Norwegian Meteorological Institute's meteorological forecast API (global, id metno).
- The USA NOAA's Global Forecast System (global, id gfs).
- The USA NOAA's Integrated Surface Database (global, id isd).
- The USA NOAA's Public Alert system (USA, id nwspa).
- The UK Met Office's Severe Weather Warning system (UK, id metwarn).
- Environment Canada's Canadian Meteorological Center ensemble model (global, id cmc).
- The US Navy's Fleet Numerical Meteorology and Oceanography Ensemble Forecast System (global, id fnmoc).
- The USA NOAA and Environment Canada's North American Ensemble Forecast System (global, id naefs).
- The USA NOAA's North American Mesoscale Model (North America, id nam).
- The USA NOAA's Rapid Refresh Model (North America, id rap).
- The Norwegian Meteorological Institute's GRIB file forecast for Central Europe (Europe, id metno_ce).
- The Norwegian Meteorological Institute's GRIB file forecast for Northern Europe (Europe, id metno_ne).
- Worldwide METAR weather reports (global, id metar).
- The USA NOAA/NCEP's Short-Range Ensemble Forecast (North America, id sref).
- The USA NOAA/NCEP's Real-Time Mesoscale Analysis model (North America, id rtma).
- The USA NOAA/ESRL's Meteorological Assimilation Data Ingest System (global, id madis).

The Forecast API lets us query for most locations on the globe, and returns:

- Current conditions
- Minute-by-minute forecasts out to 1 hour (where available)
- Hour-by-hour forecasts out to 48 hours
- Day-by-day forecasts out to 7 days

The API call has the following format-

https://api.forecast.io/forecast/APIKEY/LATITUDE,LONGITUDE?units=unit_value&exclude=flags

where,

- **APIKEY**: Dark Sky API key
- **LATITUDE and LONGITUDE**: geographic coordinates of a location in decimal degrees
- **units**: This parameter is used to get the temperature in Celsius (units=si) or Fahrenheit (units=us)

The various output parameters being used are-

- **precipIntensity**: A numerical value representing the average expected intensity (in inches of liquid water per hour) of precipitation occurring at the given time conditional on probability (that is, assuming any precipitation occurs at all).
- **precipProbability**: A numerical value between 0 and 1 (inclusive) representing the probability of precipitation occurring at the given time.
- **temperature**: A numerical value representing the temperature at the given time in degrees Fahrenheit.
- **windSpeed**: A numerical value representing the wind speed in miles per hour.
- **cloudCover**: A numerical value between 0 and 1 (inclusive) representing the percentage of sky occluded by clouds.
- **humidity**: A numerical value between 0 and 1 (inclusive) representing the relative humidity.
- **pressure**: A numerical value representing the sea-level air pressure in millibar.
- **visibility**: A numerical value representing the average visibility in miles, capped at 10 miles.

5. CLOUD CONFIGURATION USED

5.1 EC2 Instance

- **Type** - t2.micro
- **CPU** - Intel Xeon (turbo boost up to 3.3GHz)
- **vCPU** - 1
- **Memory** - 1 GB
- **Storage** - 8GB (EBS-only)

5.2 Elastic Beanstalk

- **Template**: 64bit Amazon Linux 2015.09 v2.0.4 running PHP 5.6
- **Memory limit**: 256M
- **Rolling updates**: Enabled
- **Scaling**: Auto
- **Number of instances**: 1-4

6. BENCHMARK METRICS

6.1 Responsiveness

This metric measure the ability to provide an optimal viewing and interaction experience—easy reading and navigation with a minimum of resizing, panning, and scrolling—across a wide range of devices (from tablets to mobile phones). Generally, 100 to 200ms is the threshold beyond which users will perceive slowness in an application. Android uses 9-patch images which are essentially bitmap images that automatically resize to accommodate the contents of the view and the size of the screen. Selected parts of the image are scaled horizontally or vertically based indicators drawn within the image. We use the 9-patch images to show the weather indicators on the different tabs on the result screen. We use Amazon Device Farm to test the responsiveness of our application on various devices.

6.2 Server Response Time

This is the amount of time that it takes for the server to send a response to the application, subtracting out the network latency between Google, Forecast and our EC2 web server. There may be variance from one run to the next, but the differences should not be too large. In fact, highly variable server response time may indicate an underlying performance issue. An ideal server response time should be under 200ms. There are dozens of potential factors which may slow down the response of your server: slow application logic, slow database queries, slow routing, frameworks, libraries, resource CPU starvation, or memory starvation. To test the response time, we host the web server on three different places and measure the response time. We host the server at following places-

- Local Server on Host Machine (using WAMP Server stack)
- USC CS-Server
- Amazon AWS

7. RESULTS

7.1 Experimental Setup

For the experimental setup, we used two different types of Amazon EC2 instances in us-east-1a region as given in table below. These instances use the Intel Xeon E5 processors running at 2.4GHz.

Model	vCPU	Memory(GB)	Storage(GB	Network Performance	Performance Instance Type
t2.micro	1	1	1x8	Low-Moderate	Burstable
m4.xlarge	4	16	1x32	High	Fixed

The instances are configured with 64-bit Amazon Linux running PHP 5.6 on top of Apache Server stack. We have used the default template from Elastic Beanstalk with some modifications.

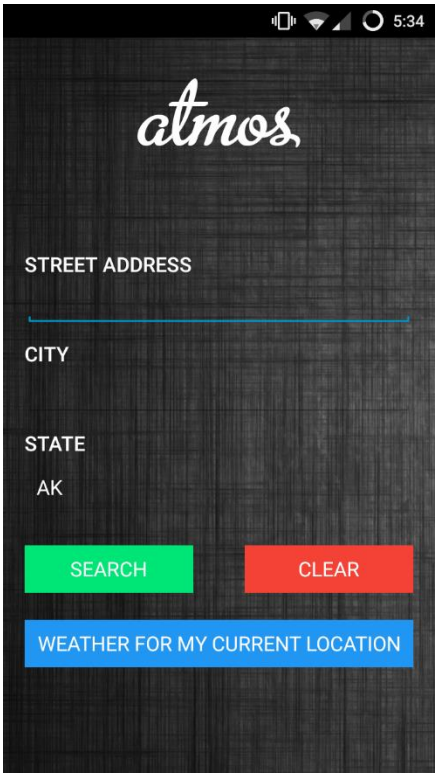
7.2 Amazon Device Farm

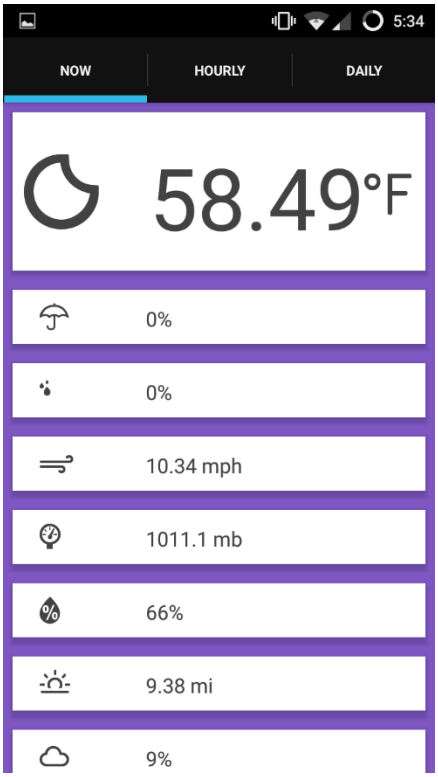
We tested our app using the Explorer test suite. The built-in explorer test crawls the app by analyzing each screen and interacting with it as if it were an end user. It takes screenshots as it explores. The device pool we selected consisted of the following devices-

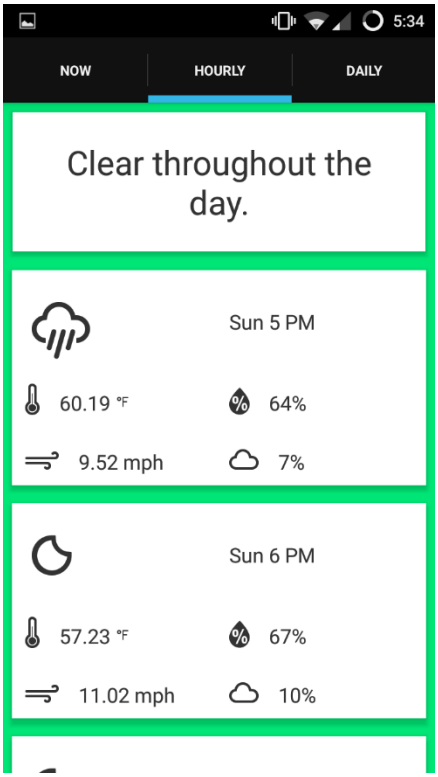
Model	Android OS Version
Samsung Galaxy S5	4.4.2
Samsung Galaxy Tab 4 10.1	4.4.2
Amazon Kindle Fire HDX 7	4.4.3
LG G Pad 7.0	4.4.2
Samsung Galaxy S6	5.0.2

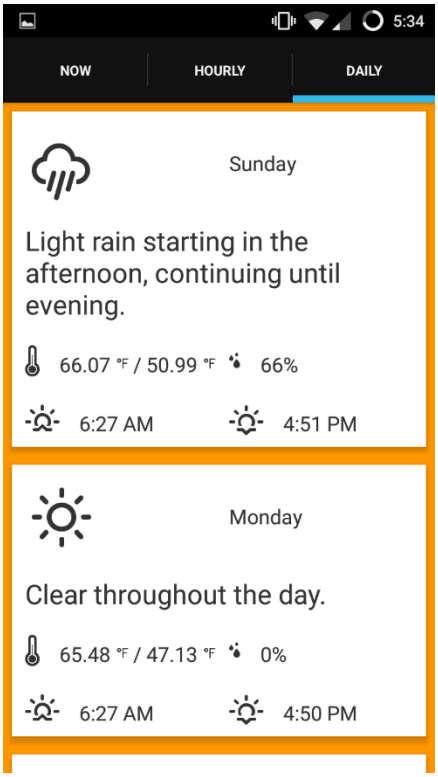
The application was responsive in both horizontal and vertical directions on almost all of the devices. The images were clear and were not clipped. The text was also appearing crisp. We have attached some screenshots from testing the app on a Samsung Galaxy S3 running Android 5.1.1.

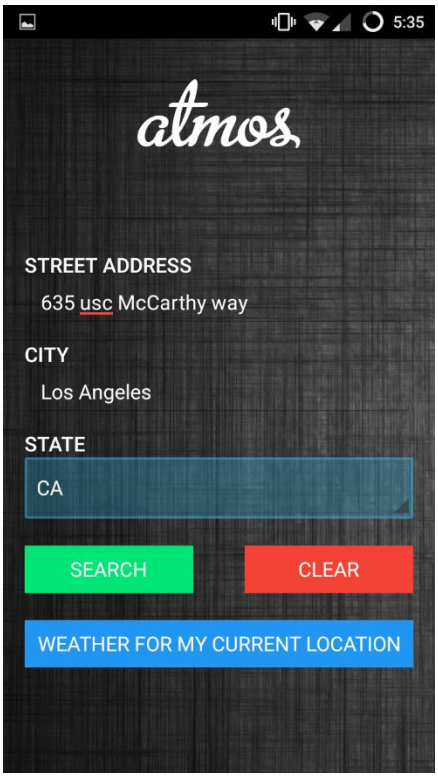
For the Samsung Galaxy Tab 4 10.1, it failed the tests because we did not have 9-patch images for such a big screen size. Hence the images were distorted or had their edges cut-off. Apart from that, it worked fine.

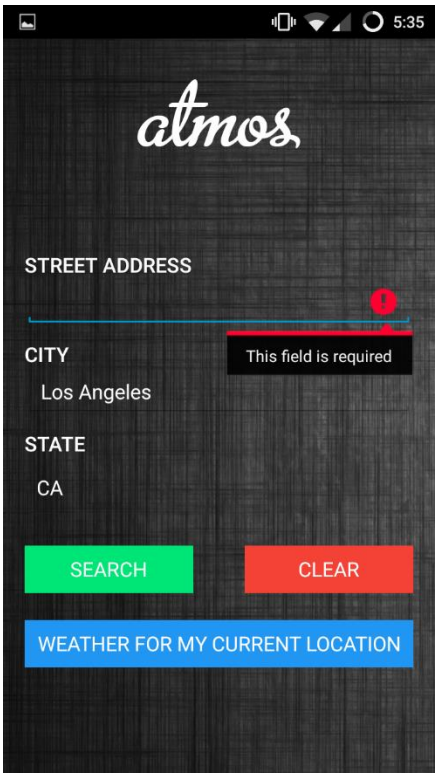




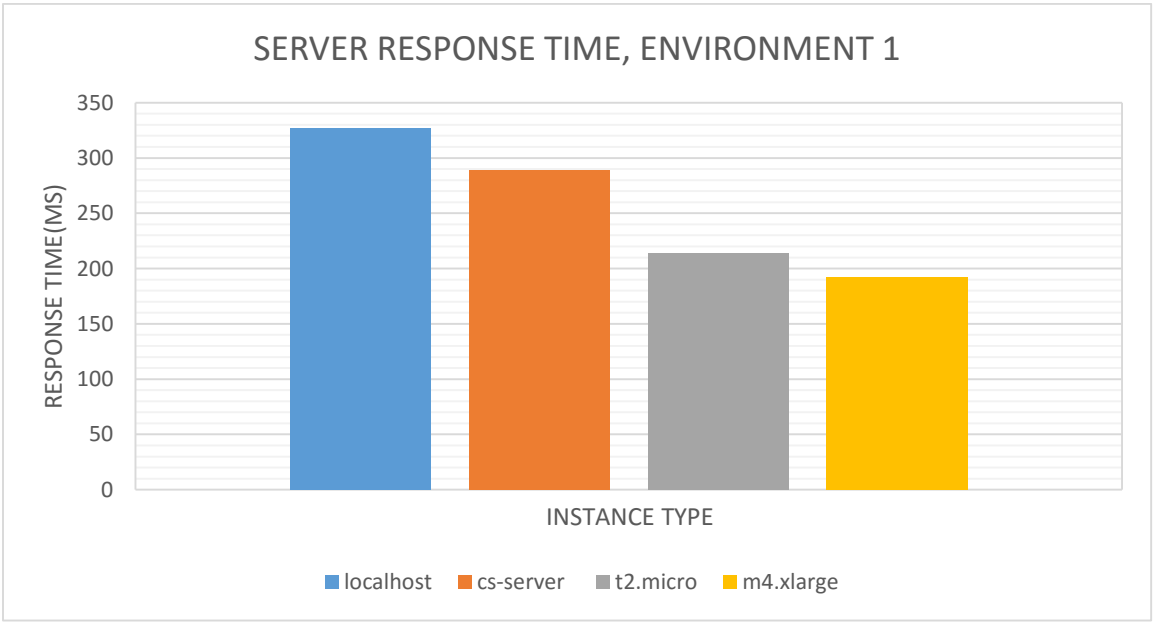


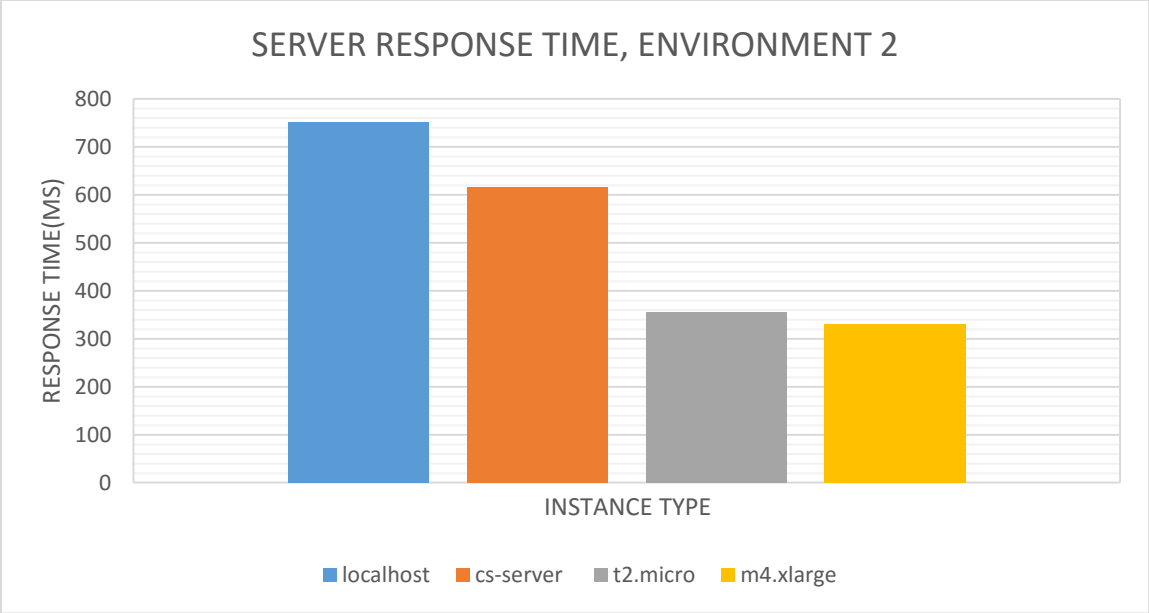






7.3 Response Time





For measuring response time, we setup the backend on 3 different servers. We can see that the local server running Apache Tomcat server has the maximum latency. This is because the server has very low resources. We then move the backend to the hosting available on cs-server.usc.edu using Apache server. We can see that there is better response time as compared to the local server setup. This is because the underlying host hardware provides better run configuration and resources for the Apache server. For the final set of experiments, we host the server on AWS and observe almost 50% reduction in response time of the application server. This has been made possible by using CloudFront and a dedicated AWS instance for Apache server which helps in minimizing the network latency between the server and Google/Forecast. On scaling up the instance to m4.xlarge, we see that the response time has further improved by approximately 10%. This is primarily because of the better network throughput of the m4.xlarge instance in comparison to the t2.micro instance.

7.4 EC2 Performance Matrix

The performance results for the EC2 instances are:

7.2% CPU Utilization
1.1 Average Latency
29KB Max Network In
27KB Max Network Out

8. CONCLUSION

We developed an Android application which has a backend deployed on Amazon Web Services. The app provides a neat and clean interface to show the weather details and provides push notifications for emergency weather alerts. We used various service offerings from the Amazon Web Services and it was easy to get our app up and running. We used the Device Farm service to measure the screen responsiveness. We also used the EC2 instance experiments to show that the server deployment on

AWS provides the best server response time and it can auto-scale automatically according to the number of requests.

9. REFERENCES

- **https://developers.google.com/maps/documentation/geocoding/intro**
- **https://developer.forecast.io/docs/v2**
- **https://developers.google.com/cloud-messaging/gcm**
- **https://aws.amazon.com/ec2/instance-types/**
- **https://s.yimg.com/ge/labs/v1/files/ycsb-v4.pdf**
- **http://developer.android.com/guide/index.html**
- **https://developers.facebook.com/docs/sharing/android**
- **http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/GettingStarted.html**
- Hwang, K. Yue Shi and X, Bai, "Scale-Out and Scale-Up Techniques for Cloud Performance and Productivity", IEEE CloudCom 2014, Singapore, Dec. 18, 2014.
- Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R. Fahringer, T. and. Epema, D., "A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing," Proc. Int'l Conf. on Cloud Computing, Springer 2010.