NSDI Cooperative Agreements Program

Category 7: Geospatial Platform Cloud Service Test Bed

Final Report

Date: September 2013

Agreement Number: G12AC20138

Project title: **Category** 7: IndianaMap Cadastral Cloud Implementation

Organization: Indiana University
620 Union Drive, Room 618
Indianapolis, IN 46202-5167
http;//www.iupui.edu

Principle Investigator: David J Bodenhamer
V. 317-274-2455
F. 317-278-1830
Intu100@iupui.edu

Collaborating Organizations: Indiana Office of Technology (IOT)
Indiana Geographic Information Council (IGIC)
Indiana University Science Gateways Group
Polis Center - IUPUI.

EXECUTIVE SUMMARY

The primary objective of this project is to evaluate the technical issues, opportunities, and costs associated with deployment and geoprocessing of IndianaMap Cadastral data in Amazon Cloud Platforms. The IndianaMap portals were developed to provide a central single-source repository of data to support transportation planning, economic development, environmental assessment, and emergency response. The Indiana Office of Technology (IOT) working with the Indiana Department of Homeland Security (IDHS), and the Indiana Geographic Information Council (IGIC) have developed a strategy for creating a seamless parcel map for Indiana. Indiana IOT harvests parcel data from county web feature services monthly. The process includes integrated Spatial ETL (Extract, Transform, and Load) tools to build a seamless state-wide parcel feature class. Currently 85 of 92 counties containing 3,225,000 parcels are included in the statewide layer.

We tested Amazon-based deployments of GeoServer and ArcGIS Server. In order to obtain user feedback of the cloud-based geospatial server implementations, we developed a parcel search query application by defining a circular buffer around a given point. Upon execution of the query, the search results were displayed and analyzed for parcel property characteristics and attributes. For the GeoServer implementation we performed a comprehensive benchmark testing. We randomized the point location and the radius of the buffer in order to simulate the parcels search query application and performed this test an arbitrary number of times. For the ArcGIS Server implementation, with the participation from the Indiana Geographic Information Council (IGIC) membership, we performed queries and submitted the survey results via Survey Monkey and email.

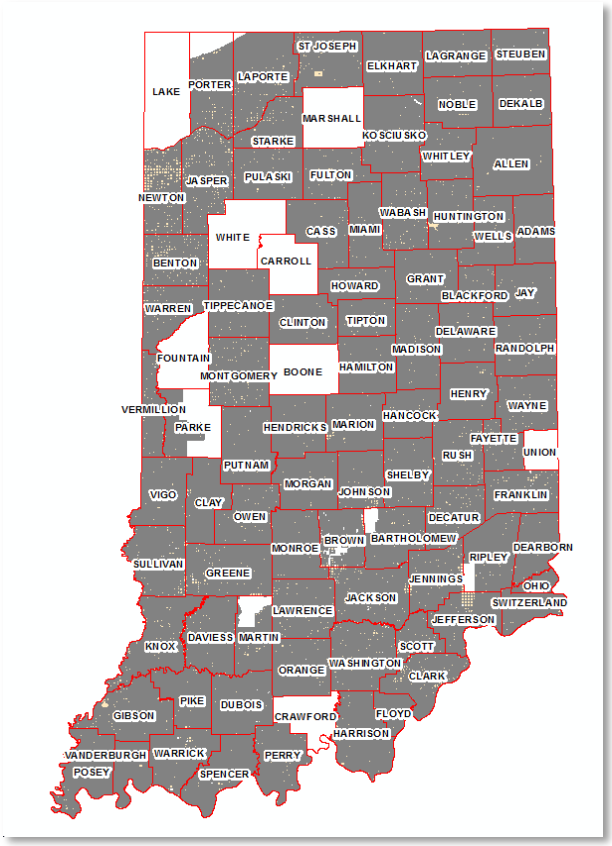
Project Narrative

The list of project activities completed as part of this project is listed below.

Preparation of Parcel Data: We worked with IOT and IGIC to develop the latest harvest and enhance the statewide dataset by joining additional attributes from the state property appraisal system. No issues encountered. Figure 1 is screen capture of the August 2012 Indiana Harvest Parcel coverage.

Delivery file name:	**GISStatewide_Parcels_101212_NO_BOONE.gdb**
Projected Coordinate System:	NAD_1983_UTM_Zone_16N
Projection:	Transverse_Mercator
Total Record Count:	3,221,920 parcel polygons

Figure1: August 2012 Indiana Harvest Parcels



OGC Web Services Development

Cloud Deployment Summary: The team explored and experimented with the GeoCloud Sandbox environment, and established the project prototype in Amazon Elastic Computing Cloud (EC2), using Amazon Identity and Access Management (IAM) for access control of developers and collaborators. We setup two Amazon EC2 on-demand instances for the ArcGIS and Geoserver deployment. Each deployment is associated with an Elastic IP (EIP) address.

Amazon Cloud Features: AWS (Amazon Web Services) offers a comprehensive set of cloud infrastructure and application services. Our goal was to explore various technologies available, and investigate pros and cons of Amazon cloud infrastructure. We focused on following AWS computing and networking products:

1. **Elastic Compute Cloud (EC2) for application deployment:** EC2 provides basic virtual machine services such as create images and launch instances. It offers a variety of instance types and configurations, highlighting flexible and elastic computing capacity for user applications.
2. **Elastic Load Balancing (ELB) for high availability:** while EC2 provides multiple regions and availability zones to launch instances and protect user applications from single point failure, ELB further enhances application availability by seamlessly distributing incoming application traffic across multiple instances. It also detects unhealthy instances and automatically reroute traffic as necessary.
3. **Auto Scaling (AS) with CloudWatch for on-demand scalability:** AS can automatically increase or decrease EC2 instances according to predefined conditions such as Amazon CloudWatch monitor metrics to respond to demand spikes and lulls. It can detect and replace unhealthy instances, and be configured to work with ELBs.
4. **CloudFront for global content distribution (CDN):** CloudFront is a Content Delivery Network (CDN) that enhances application performance by delivering entire web contents through a global network of edge locations.
5. **CloudFormation for resource management:** a collection of related AWS resources as listed above can be easily managed, provisioned and updated through CloudFomation templates. The JSON-format, text-based file describes all the AWS resources needed to deploy and is instantiated as a stack.

Typical steps to deploy a load-balanced and auto-scaled application on Amazon cloud included:

1. Creating customized AMI based on a fully installed and loaded instance;
2. Creating ELB for listeners and health check conditions;
3. Creating auto scaling group using the ELB;
4. Configuring scale (up and down) policies for corresponding CloudWatch alarms;
5. Configuring CloudFront distributions (optional);
6. Assembling all in a CloudFormation template.

Test Plan

We deployed parcel service in two platforms, one using ArcGIS server and the other using GeoServer. The two were not used to compare the performance but to test the ease of installation and ease of use and perception of using AWS for large GIS datasets. Only GeoServer was used to test performance. Geoserver was chosen for the testing because of OCG compliant API capabilities. GeoServer wms API had the capability to issue cql_filter parameter for filtering data using a REST API call. This allowed us to develop an automated test application that queried the parcels dataset using a randomized buffer query.

Details of the GeoServer Server Development

The GeoServer instance is a M1 Medium Instance of Ubuntu Server 12.04.1 LTS. State parcels data were loaded into PostgreSQL, and published with GeoServer. Figure 2 shows accessing WMS service with desktop GIS.

-- GIS Server: GeoServer 2.2.1:

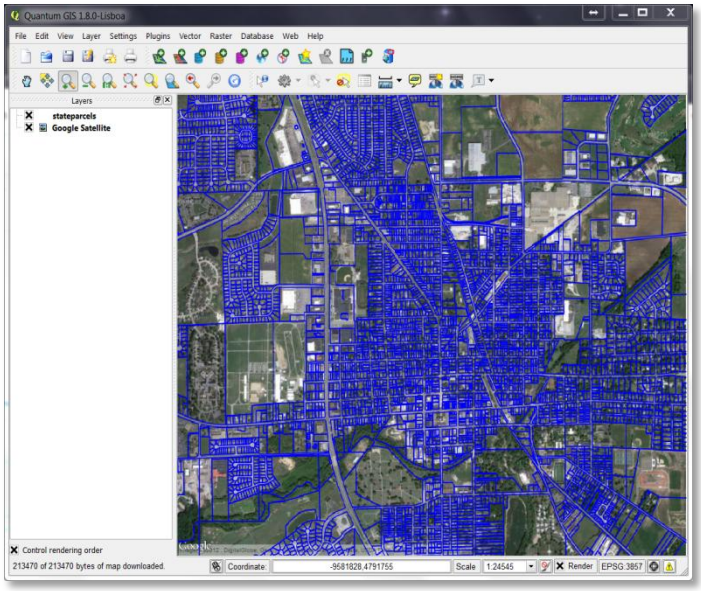
-- Geospatial database: PostgreSQL 9.1 with PostGIS 1.5 extension

WMS service: http://54.243.214.240:8080/geoserver/cloudgis/wms

WFS service: http://54.243.214.240:8080/geoserver/cloudgis/wfs

Layer preview: http://54.243.214.240:8080/geoserver/web/

Figure 2: Access WMS service with desktop QGIS

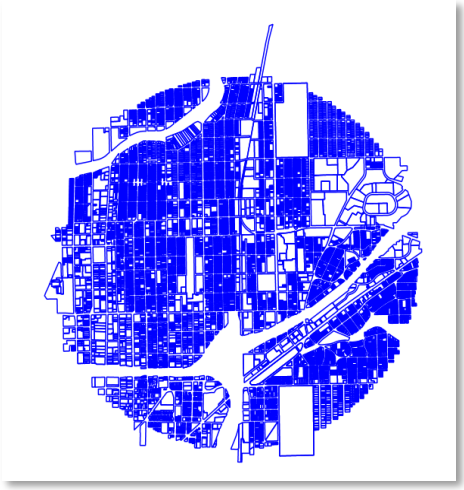


Geoserver Query Development: The team prototyped and tested example parcel search queries with the Geoserver deployment. Querying parcels by defining a circular buffer around a given point was accomplished with CQL_Filter support for both WMS and WCS services. Two example queries were developed.

Example 1: query parcels within 2000 meters of the point (573570.19322 4405157.43581)

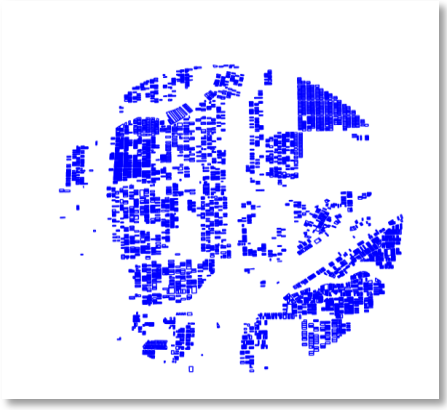
http://54.243.214.240:8080/geoserver/cloudgis/wms?service=WMS&version=1.1.0&request=GetMap&layers=cloudgis:stateparcels&bbox=571000,4402800,576000,4408000&width=600&height=600&srs=EPSG:26916&format=image/png&cql_filter=DWithin(geom, Point(573570.19322 4405157.43581),2000,meters). The output is shown in Figure 3.

Figure 3: Example of spatial query with a circular buffer



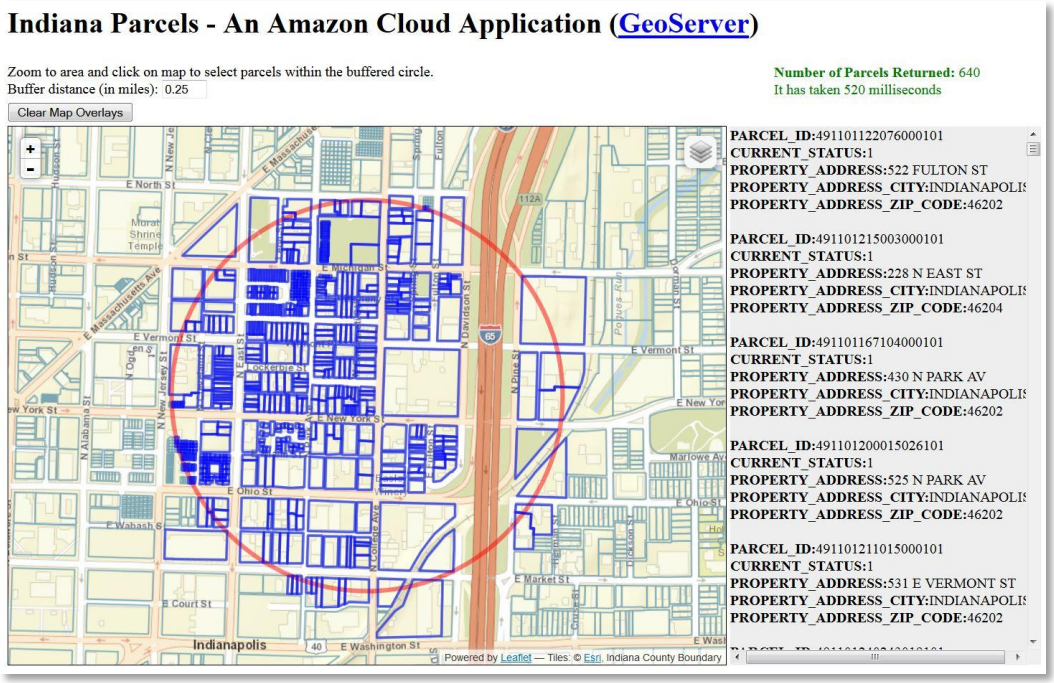
Example 2: Query single family residential within 2000 meters of the point (573570.19322 4405157.43581) single family residential is defined by propery_class_code =510-515.
http://54.243.214.240:8080/geoserver/cloudgis/wms?service=WMS&version=1.1.0&request=GetMap&layers=cloudgis:stateparcels&bbox=571000,4402800,576000,4408000&width=600&height=600&srs=EPSG:26916&format=image/png&cql_filter=DWithin(geom, Point(573570.19322 4405157.43581),2000,meters) and property_class_code between 510 and 515 The output is shown in Figure 4.

Figure 4: Example of spatial and attribute query with a circular buffer



GeoServer Web interface Development: The web interface, Figure 5, is built with open-source JavaScript mapping library: leaflet (http://leafletjs.com/). Users are able to set the query center by clicking on the map. The client pulls parcels records inside the user-specific search distance as GeoJSON from GeoServer through WFS (Web Feature Service). Number of parcels returned, the query time, and the details of the parcels are displayed on the right side of Map. Due to the limited processing power inside the web browser, the upper limit of returned records is set at 10000.

Figure 5: Example of Web Interface



GeoServer Query Testing: We conducted load tests on Geoserver deployment to demonstrate functional features of Amazon ELB and AS, using Indiana University Intelligent Infrastructure (IUII) VM gf8.ucs.indiana.edu for reference. The Amazon EC2 VM is a m1.medium instance with 3.75GB memory and 2 EC2 compute unit. IUII VM contains 2 CPUs and 8 GB memory. The load test issues parcel search queries of random point locations and buffer radius. We used Apache Jmeter (http://jmeter.apache.org) software to simulate concurrent access of multiple users: ramp up 256 threads in 30 seconds, hold load for 5 or 10 minutes, then gradually stop all threads in 30 seconds. We focused the performance measure on query response time and overall throughput.

GeoServer Query Testing Amazon ELB result: ELB automatically distributes incoming application traffic across multiple EC2 instances. Figure 6 shows the average response time (in seconds) during the 6 minutes testing period on two VM ELB, one EC2 VM, and IUII VM. Figure 7 shows the corresponding results for throughput, measured as number of estimated transactions per second. It demonstrates that even though single EC2 instance cannot compete with IUII VM, which is relatively more powerful and has better local

network connection to the testing client, the performance of two VM ELB becomes pretty comparable.

Figure 6: Response time of two VM ELB, one EC2 VM, and IUII V

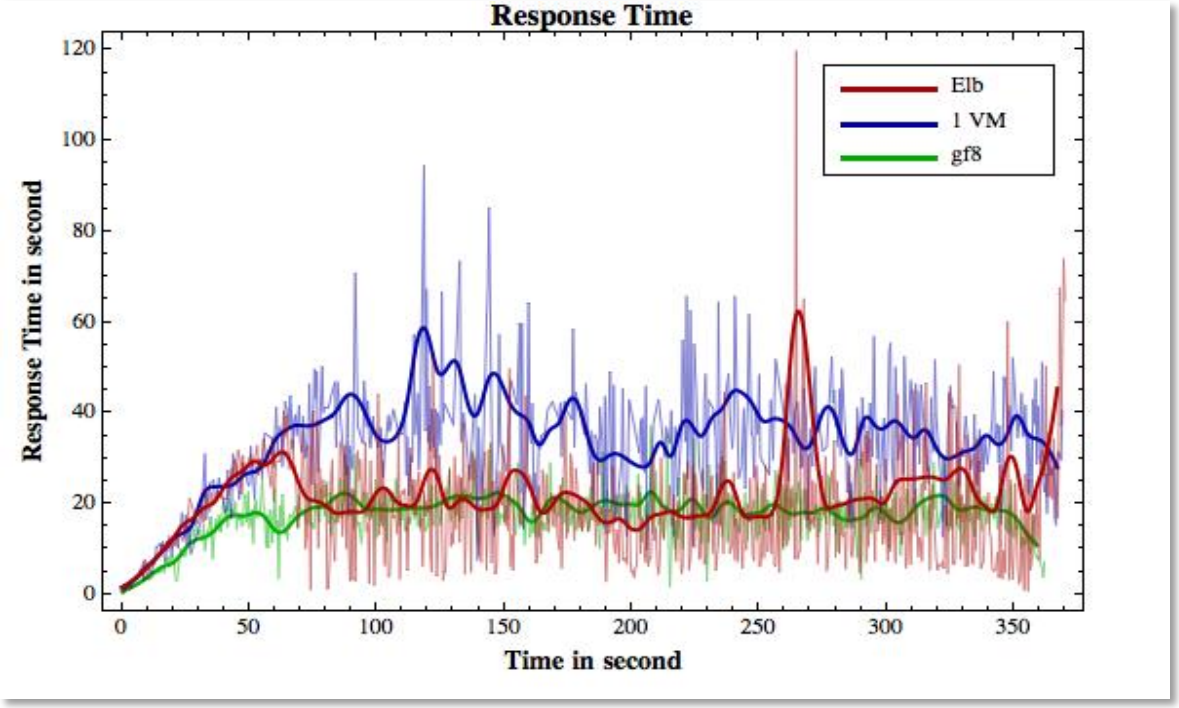
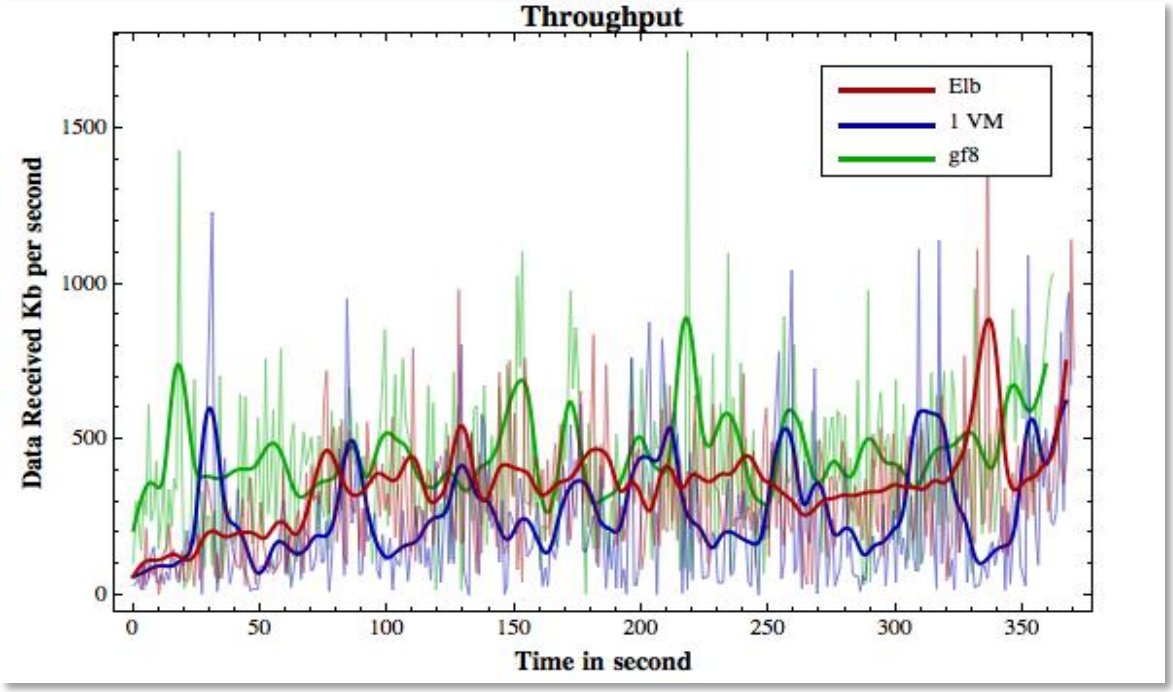


Figure 7: Throughput of two VM ELB, one EC2 VM and IUII VM



Testing GeoServer Amazon Auto-Scaling result: Auto Scaling allows users to scale the number of active EC2 instances up or down automatically based on predefined CloudWatch metrics. For testing purpose, our scaling policy is set to increase the number of instances by one when the CPU utilization is higher than 70% for 2 minutes, and decrease the number of instances by one when the CPU utilization is lower than 50% for 10 minutes. We also configured the auto scaling group behind an ELB to evenly distribute server loads. Figure 8 and Figure 9 demonstrate auto-scaling effect on the response time and throughput over the 11 minutes testing period: we started with a single EC2 instance, as the server load increases and about 3 minutes into the test, the query response time quickly hits the ELB default 60 seconds timeout, with one retry, capping the response time at 120 seconds; auto-scaling policy now enters the evaluation and about 2 minutes later, at 5 minutes into the test, a new instance was started to share the load, bringing down the response time to 60 seconds or so; as auto-scaling evaluation continues, yet another scale up happened 2 more minutes later, dropping the response time further down to 30 seconds or so. The same effect is also reflected on the throughput results: throughput sank to nearly zero when the response time capped at timeout value, then jumped back up twice responding to the scale up events.

Figure 8: Response time of Amazon auto-scaling test

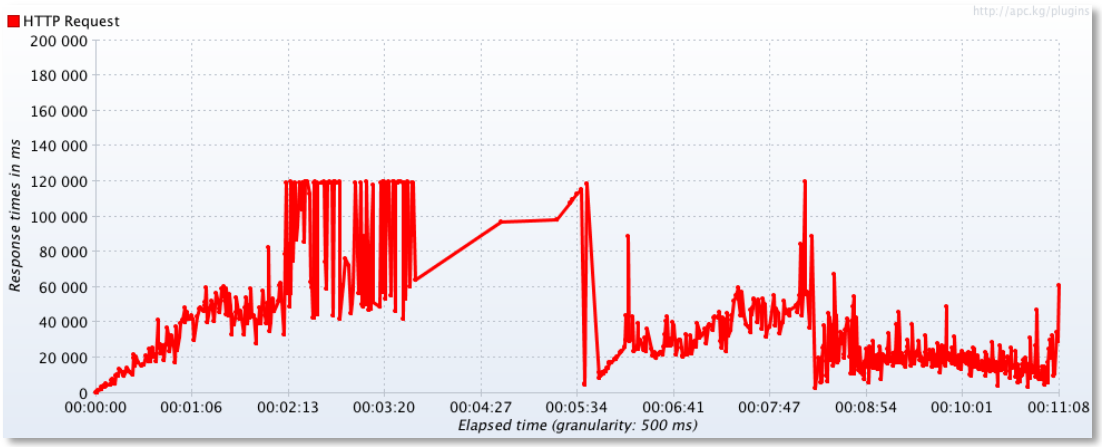
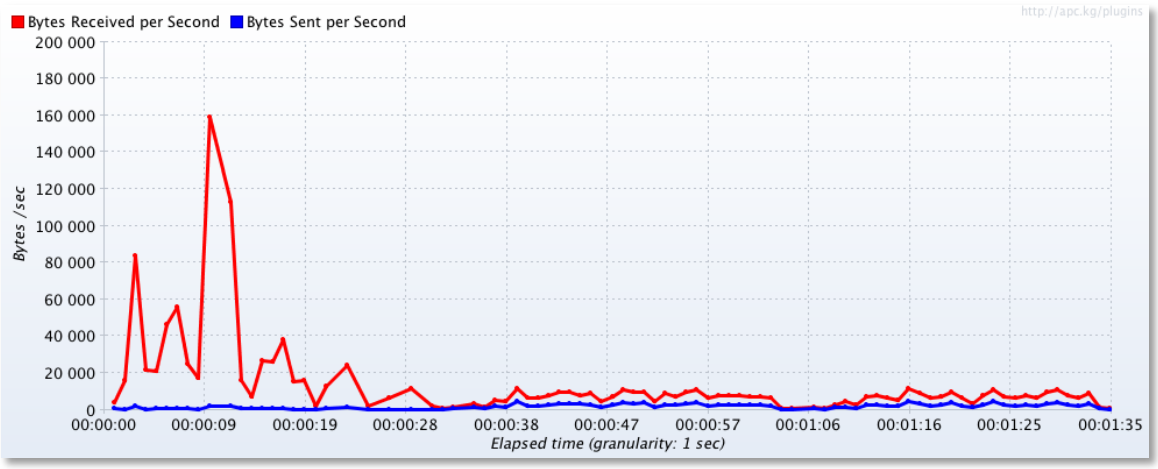


Figure 9: Throughput of Amazon CloudFront Test



Challenges and Solutions: During the GeoServer web interface development, we encountered the issue of cross-domain queries for the JavaScript client in web browser environment. By default GeoServer runs on port 8080, which is considered a different domain from the JavaScript user interface on port 80. The same-origin policy of typical web browsers prohibits the direct data access between the two. While AJAX JSONP provides the communication technique for such cases, its support in earlier versions of GeoServer was scatter. At early development stage, we setup Apache proxy forwarding for GeoServer to be accessible in the same domain as the user interface to solve the problem. As GeoServer releases more stable support for JSONP later on, we switched to the proper JQuery implementations without port forwarding.

While performing load tests for Amazon cloud features, we found it challenging to design best test cases and correctly interpret results, because many uncontrollable external factors can come into play. For example, when too much load is put on the fixed 2 VM ELB configuration, we got response time results as shown in Figure 10. It turns out that Amazon ELB timeouts persistent sock connections at 60 seconds, and our JMeter test plan has a built-in one time retry, making the majority of response time capped at 120 seconds. Figure 11 and 12 demonstrate typical load test results for CloudFront configuration, where the response time quickly runs flat and throughput sinks to nearly zero. It turns out that CloudFront caches the time out response code from origin servers for 5 minutes, making the traditional load test meaningless. Our final cases were designed through research readings on such issues and after rounds of trials and errors.

Figure 10: Response Time of Overloaded 2VM ELB

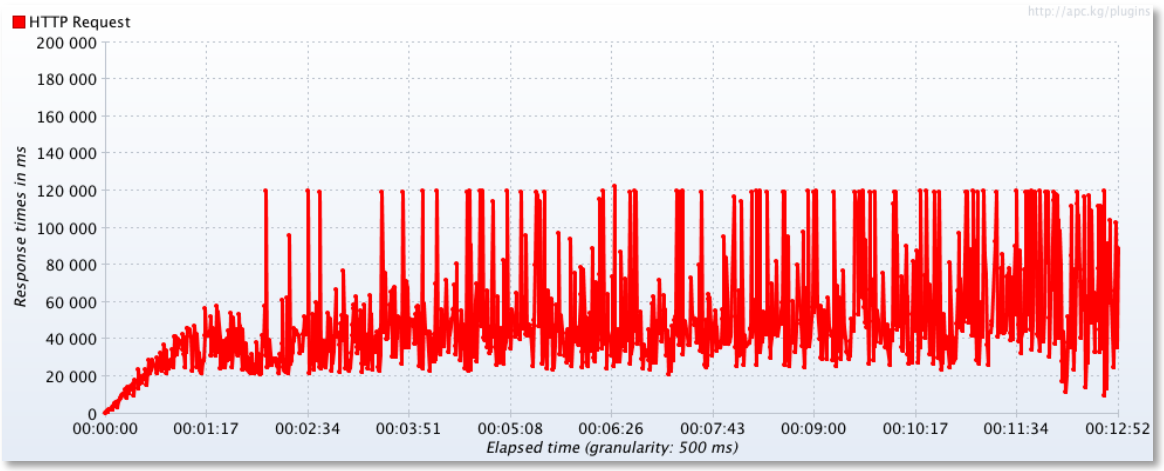


Figure 11: Response Time of Amazon CloudFront Test

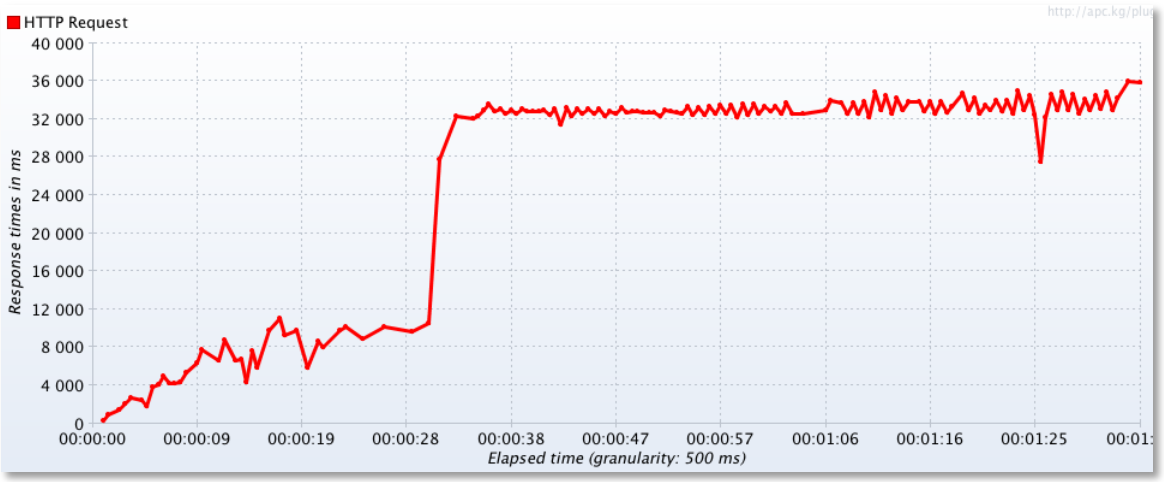
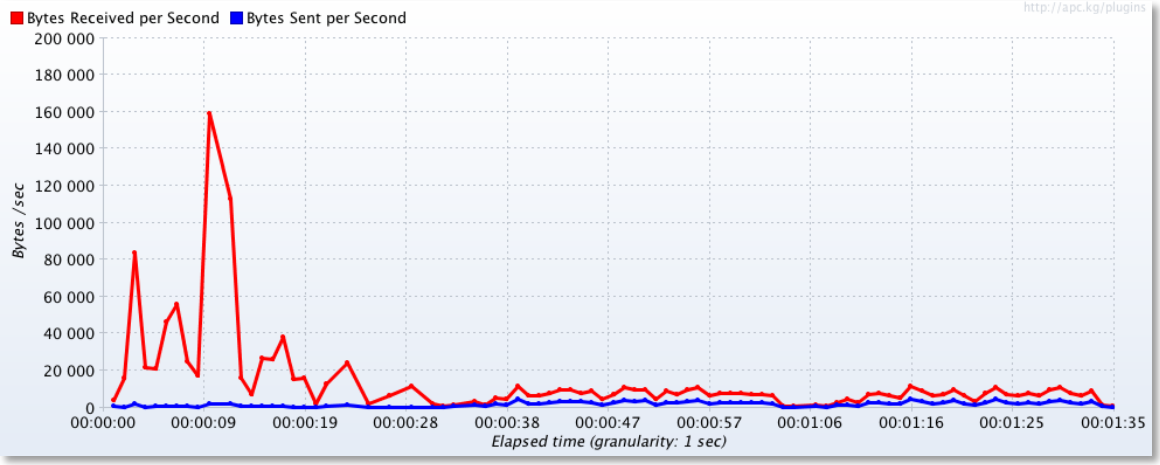


Figure 12: Throughput of Amazon CloudFront Test



Details of the ArcGIS Server Development

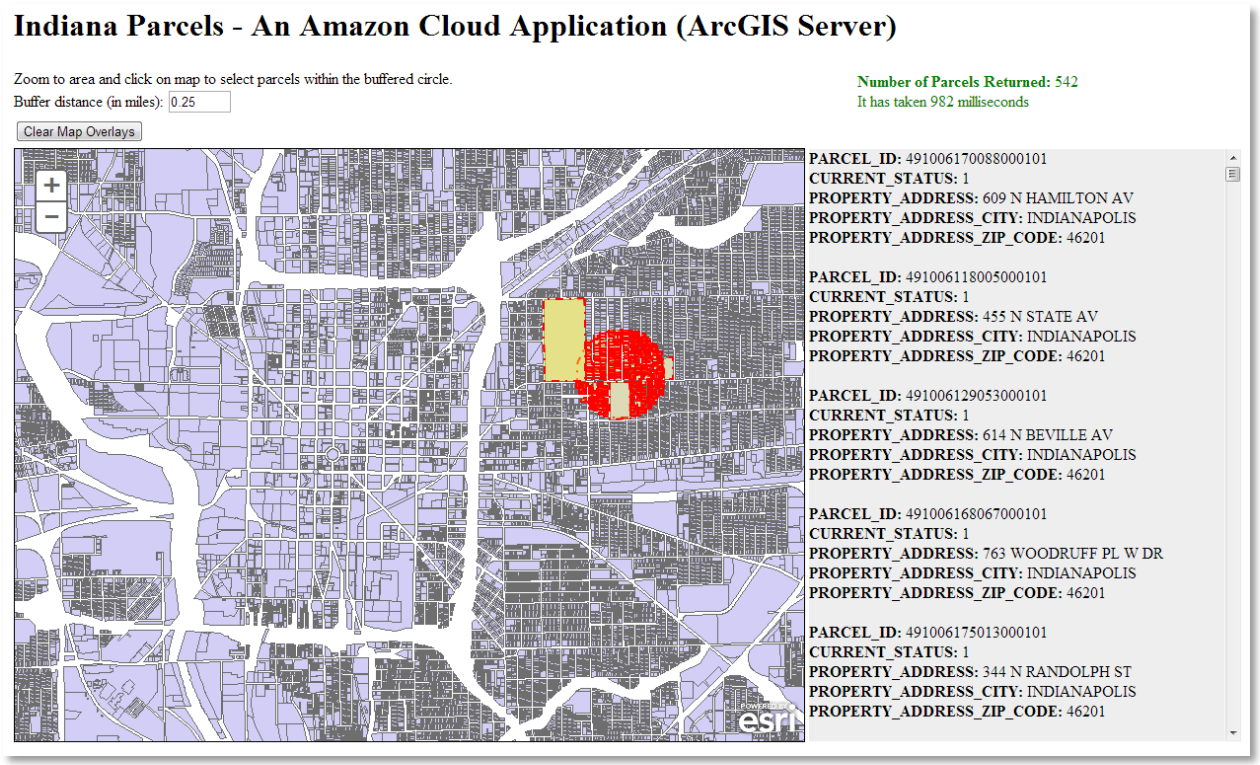
The ArcGIS server instance is a M1 Large Instance of Windows Server 2008, running ArcGIS Server 10.0 SP1. The team created a service using ArcGIS server management application. The instance was created using the Amazon Machine Image (AMI) provided by ESRI.

-- GIS Server: ArcGIS Server 10.0 SP1
-- Geospatial Data: ESRI file geodatabase
-- Parcel service API's
http://54.243.151.133/ArcGIS/services/MIBOR_Statewide_Parcels/MapServer
http://54.243.151.133/ArcGIS/services/MIBOR_Statewide_Parcels/MapServer/WMSServer
http://54.243.151.133/ArcGIS/services/MIBOR_Statewide_Parcels/MapServer/KmlServer
http://54.243.151.133/ArcGIS/services/MIBOR_Statewide_Parcels/MapServer/WFSServer
- Parcel viewer applicationhttp://54.242.136.230/cap2/bufferQuery.html

ArcGIS Server Query Development: The team prototyped and tested example parcel search queries with the ArcGIS Server deployment. The application was developed using ESRI ArcGIS server JavaScript API. Querying parcels by defining a circular buffer around a given point was accomplished with an ArcGIS server geometry service. Attribute query was developed using ArcGIS server query task. The attributes used in the applications are "PARCEL_ID", "CURRENT_STATUS", "PROPERTY_ADDRESS", "PROPERTY_ADDRESS_CITY" and "PROPERTY_ADDRESS_ZIP_CODE".

The application has the capability to zoom and pan around the Indiana map. The user can define the buffer radius in miles. Clicking on any part of the map will create a circle that is used to query the data. The data is displayed in a separate panel. Figure 13 depicts the application with the map and the query results.

Figure13: ArcGIS server



Installation and Provisioning of ArcGIS server on Amazon Cloud: Authorization for the Amazon Web Service (AWS) was easy and intuitive. The process only required a credit card. Charges were only applied when services were used. After the initial learning curve of using AWS, the process to create a windows 2008 server instance and install ArcGIS server was straight forward. To request an ArcGIS server Amazon Machine Image (AMI) for AWS, we provided our Indiana University provisioning file with ArcGIS server authorization codes and the AWS account id to the ESRI representative.

Starting and stopping server instances on as needed basis was very intuitive. Creating ArcGIS server map services was no different than setting up a typical in house server. Installing the parcel layer was performed by login to the Amazon server using remote desktop and copying the files directly into the server and using ArcGIS server manager to create the map service.

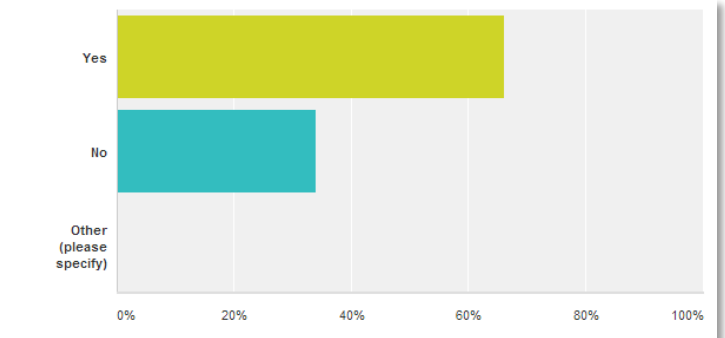
ArcGIS and GeoServer Server Query Development: The team prototyped and tested example parcel search queries with the ArcGIS Server and GeoServer deployments. Indiana Geographic Information Council (IGIC) members queried parcels by defining a circular buffer around a given point. The users were asked to record the number of parcels and the time in milliseconds. The users were asked to perform three queries. The exercise took place on May 23, 2013 during a 15 minute period. A total of 51 IGIC members participated. The results of the queries are listed in Table 1.

Table 1: Interactive User Parcel Query Results

Test 1		Test 2		Test 3	
# Parcels Queried	Milliseconds	# Parcels Queried	Milliseconds	# Parcels Queried	Milliseconds
81	113	8,118	16,211	1,000	6,041
1,442	436	3,150	5,493	10,000	13,667
542	727	5,111	9,263	10,000	7,258
18	500	629	3,390	Did not query through. Had "Stop running script?" message	None.
332	46,718	4,177	25,406	10,000	1,454
579	5,292	4,214	3,760	6,094	43,354
266	344	3,341	13,259	10,000	6,924
321	903	2,678	44,908	9,649	43,345
160	448	2,031	3,997	10,000	37,904
333	2,656	3,050	5,233	10,000	14,615
262	456	297	3,398	froze machine	had to manually shut down browser
157	2,493	7,332	5,186	1,243	6,765
13	175	588	1,098	10,000	24,896
338	3,015	2,823	53,344	3,238	4,054
479	490	3,749	2,350	10,000	5,086
377	668	5,466	26,023	10,000	15,975
468	1,904	8,033	11,830	-	-
163	988	5,509	12,430	10,000	19,698
608	578	8,057	37,699	10,000	20,004
421	1,579	3,566	3,025	over 10,000	bombed
518	1,638	7,846	47,305	Never pulled back records	Never pulled back records
102	13,608	6,231	49,892	919	1,065
123	234	767	41,383	9,556	8,869
242	41,604	3,889	11,912	10,000	34,447
53	333	654	582	10,000	6,039
256	49,422	7	Unknown; process would not complete	Never got a chance to run, previous test failed to finish after 10+ mins	Never got a chance to run, previous test failed to finish after 10+ mins
151	365	Failed to ifnish after 10+ mins; time for testing expired	Failed to ifnish after 10+ mins; time for testing expired	7	Unknown. Process never completed
267	47,985	2,332	2,770	6,178	7,335
7	Unknown. Process never completed; waited 10 mins	3515	26562	10,000	9,449
1,241	51,030	2,043	2,394	10,000	39,283
222	34,514	4,183	5,055	10,000	6,530
531	538	693	2,156	1,192	3,999
18	828	4,501	4,784	10,000	10,396
263	44,200	4,195	18,584	10,000	2,930
806	51,440	2,577	4,279	10,000+	15,086
726	46,484	7,134	4,949	2,259	3,338
10,000	15,116	5,880	22,593	10,000	15,061
149	705	2,461	4,162	10,000	33,271
375	3,886	619	1,030	10,000	8,040
298	318	5,629	8,984	154	8,884
177	740	3,820	3,239	10,000	317
497	493	580	580	10,000	59,547
277	619	2,179	3,127	10,000	1,646
197	490	3,780	32,602	10,000	6,212
632	1,561	7,965	6,381	6,987	6,255
255	486	2,147	1,877	7,081	7,099
255	452	2,152	2,405	10,000	8,658

The most interesting result of this query exercise was the large range in capabilities of the users. The exercise revealed a large number of variables that were not controlled. Users were located across Indiana with different Internet Service Providers, internal networking environments, client hardware, and operating systems.

The uncontrolled variables make it impossible to comprehensively analyze all of the results. It should be noted however that it was not the intent of this investigation to isolate all variables. Instead, this was designed to provide insight as to the overall quality of the user experience in a wide range of user environments. Given that purpose, we are able to draw conclusions about the user experiences using geospatial data, software, and models in the cloud. Most importantly, approximately 2/3 of the users expressed interest and would consider using the geocloud.



Answer Choices	Responses	
Yes	65.96%	31
No	34.04%	16
Other (please specify)	0%	0
Total		47

Server Hosting Costs: Our analyses Identified the major cost was associated with hosting a server running (24x7). Amazon continues to lower prices on its AWS offerings. Based on the latest pricing, the cost of reserving a single m1.medium Linux server instance as used for our Geoserver test case is under $600 per year ($338 one-time upfront payment + $21 monthly fee = $590). To rent a similarly configured VM from IUII (2 CPU, 4GB memory, 35GB disk) costs about the same ($400 standard VM + $125 additional CPU + 2*$40 additional memory = $605).

Amazon ELB and AS model serves extremely well for computing needs of emergency responses, where the application server usage is generally very light but spikes quickly in case of a natural disaster or emergency. IUII virtual systems, although perform quite well,

doesn't offer such elasticity, and the much more flexible usage-by-hour pricing model from Amazon.

In addition to computing products described above, AWS also offers a variety of storage products. Elastic Block Store (EBS) provides highly available persistent storage for EC2 instances; Simple Storage Service (S3) is designed for Internet data storage and hosting using simple web interface (REST API) for I/O operations; Glacier is optimized for low cost data archival. Our experiment configuration fits best with vector data, where the dataset size is relatively small and can be easily bundled inside the application server for ELB and AS. The additional cost for corresponding EBS persistent storage is minimum. For example, the complete IndianaMap vector dataset is under 10GB, which corresponds to $1 monthly fee.

However, bundling data inside the application server quickly becomes expensive and impractical for AS with large amount of raster data because EBS volumes cannot be shared among multiple instances. One simple solution is to separate application services from data storage by having a dedicated high performance server for data hosting. Application servers can access the data through OGC cascading services while remain flexible for ELB and AS. Our baseline tests of Geoserver WMS services showed no noticeable performance penalty on data access in such configurations. AWS EC2 offers high storage instance hs1.8xlarge that can be used as the data server for just under $20K a year. The complete IndianaMap product including both vector and raster data is about 18TB total, hence the additional archival cost using Amazon Glacier is $2160 annually.

Summing up, the total cost of hosting the complete 18TB IndianaMap data product, with simple backup in Amazon Glacier and a separate Linux application server configured for elastic load balancing and auto scaling, is around $22K/year. Adding on actual on-demand usage fees, the annual cost should be under $25K.

Next Steps: Indiana information technology in general, and the Indiana Geographic Information Office in particular, are exploring options related to working in "the cloud" but have not yet committed to projects that take advantage of these varied opportunities. This project has been beneficial by adding to our understanding of the benefits and challenges of cloud-based geographic information. The test bed successfully provided the knowledge transfer to establish the Amazon-based deployments of GeoServer and ArcGIS Server. The initiative also fostered successful relationships with Indiana Geographic Information Officer (GIO), IGIC and Indiana University to carry this forward.

Additional technical assistance is not required. The Geospatial Platform Cloud Service Test bed documentation enabled the team to easily implement both the ArcGIS and Geoserver deployments.

The Indiana ESRI representative participated in the testing and compiled recommendations for a "controlled variable" test which could be pursued in the future to expand the area of focus beyond this effort. Those recommendations are included in Appendix B.

Feedback on Cooperative Agreements Program: The CAP program provided great technical support both in terms of individual support and, as previously mentioned, documentation. Without the program Indiana would not have had the resources and more importantly the technical support to evaluate the GeoCloud Amazon-based deployments of GeoServer and ArcGIS Server. We have not identified any additional needs or program concerns. If we were to do the program again, we would focus on adding additional vector layers and expanding the analyses capabilities.

What are the CAP Program strengths and weaknesses?

Strengths: The FGDC CAP grant has a long and solid history in Indiana of providing directed seed funding to help us address a broad range of important geospatial topics and technologies. Many of Indiana's statewide geospatial initiatives have had their start or been supported by CAP grants. The ROI of the CAP grant program for GIS initiatives across Indiana is significant.

Weaknesses: Federal funding of the CAP grant program is inconsistent and should be increased not decreased. The 2013 & 2014 CAP grant program has fallen victim to budget cuts related to sequestration, and this is a real shame [and a mistake].

Where did it make a difference?

This CAP Grant made it possible for Indiana's geospatial community to come together to learn, experience and test geoprocessing in two different cloud environments. Without this opportunity, Indiana would not have had the resources and more importantly the technical support to evaluate the GeoCloud Amazon-based deployments of GeoServer and ArcGIS Server. As a result, the unknown is now known, and the costs and benefits are more clearly defined and understood by our GIS community.

Was the assistance you received sufficient or effective?

Yes. The CAP program provided great technical support both in terms of individual support and, as previously mentioned, documentation

Appendix A

Example Amazon CloudFormation template for the load-balanced and auto-scaled IndianaMap GeoServer application:

```
{
 "AWSTemplateFormatVersion" : "2010-09-09",

 "Description" : "AWS CloudFormation Template IndianaMapUI: Create a multi-az, load balanced and auto scaled Indiana Map service through Geoserver, with Leaflet User Interface. The application is configured to span multiple Availability Zones in the region and is Auto-Scaled based on the CPU utilization of the web servers. Notifications will be sent to the operator email address on scaling events. The instances are load balanced with a simple health check against the default web page. The web site is available on port 80, however, the instances can be configured to listen on any port (8080 by default). **WARNING** This template creates one or more Amazon EC2 instances and an Elastic Load Balancer. You will be billed for the AWS resources used if you create a stack from this template.",

 "Parameters" : {
  "InstanceType" : {
   "Description" : "Geoserver EC2 instance type",
   "Type" : "String",
   "Default" : "m1.medium",
   "AllowedValues" : [
"t1.micro","m1.small","m1.medium","m1.large","m1.xlarge","m2.xlarge","m2.2xlarge","m2.4xlarge","m3.xlarge","m3.2xlarge","c1.medium","c1.xlarge","cc1.4xlarge","cc2.8xlarge","cg1.4xlarge"],
    "ConstraintDescription" : "must be a valid EC2 instance type."
  },
  "OperatorEmail": {
   "Description": "Email address to notify if there are any scaling operations",
   "Type": "String"
  },
  "KeyName" : {
   "Description" : "The EC2 Key Pair to allow SSH access to the instances",
   "Type" : "String"
  }
 },

 "Mappings" : {
  "AWSInstanceType2Arch" : {
   "t1.micro"    : { "Arch" : "64" },
```

```
    "m1.small"    : { "Arch" : "64" },
    "m1.medium"   : { "Arch" : "64" },
    "m1.large"    : { "Arch" : "64" },
    "m1.xlarge"   : { "Arch" : "64" },
    "m2.xlarge"   : { "Arch" : "64" },
    "m2.2xlarge"  : { "Arch" : "64" },
    "m2.4xlarge"  : { "Arch" : "64" },
    "m3.xlarge"   : { "Arch" : "64" },
    "m3.2xlarge"  : { "Arch" : "64" },
    "c1.medium"   : { "Arch" : "64" },
    "c1.xlarge"   : { "Arch" : "64" }
   },

   "AWSRegionArch2AMI" : {
    "us-east-1"     : { "32" : "ami-157a177c", "64" : "ami-157a177c" },
    "us-west-1"     : { "32" : "ami-157a177c", "64" : "ami-157a177c" },
    "us-west-2"     : { "32" : "ami-157a177c", "64" : "ami-157a177c" }
   }
  },

  "Resources" : {
   "NotificationTopic": {
    "Type": "AWS::SNS::Topic",
    "Properties": {
     "Subscription": [ {
       "Endpoint": { "Ref": "OperatorEmail" },
       "Protocol": "email" } ]
    }
   },

   "WebServerGroup" : {
    "Type" : "AWS::AutoScaling::AutoScalingGroup",
    "Properties" : {
     "AvailabilityZones" : { "Fn::GetAZs" : ""},
"HealthCheckGracePeriod": "180",
"HealthCheckType": "ELB",
      "LaunchConfigurationName" : { "Ref" : "LaunchConfig" },
      "MinSize" : "1",
      "MaxSize" : "5",
      "LoadBalancerNames" : [ { "Ref" : "ElasticLoadBalancer" } ],
      "NotificationConfiguration" : {
       "TopicARN" : { "Ref" : "NotificationTopic" },
       "NotificationTypes" : [
"autoscaling:EC2_INSTANCE_LAUNCH","autoscaling:EC2_INSTANCE_LAUNCH_ERRO
R","autoscaling:EC2_INSTANCE_TERMINATE",
"autoscaling:EC2_INSTANCE_TERMINATE_ERROR"]
```

```
      }
     }
    },

    "LaunchConfig" : {
     "Type" : "AWS::AutoScaling::LaunchConfiguration",
     "Properties" : {
      "KeyName" : { "Ref" : "KeyName" },
      "ImageId" : { "Fn::FindInMap" : [ "AWSRegionArch2AMI", { "Ref" :
"AWS::Region" },
                                { "Fn::FindInMap" : [ "AWSInstanceType2Arch", { "Ref" :
"InstanceType" },
                                "Arch" ] } ] },
      "SecurityGroups" : [ { "Ref" : "InstanceSecurityGroup" } ],
      "InstanceType" : { "Ref" : "InstanceType" }
     }
    },

    "WebServerScaleUpPolicy" : {
     "Type" : "AWS::AutoScaling::ScalingPolicy",
     "Properties" : {
      "AdjustmentType" : "ChangeInCapacity",
      "AutoScalingGroupName" : { "Ref" : "WebServerGroup" },
      "Cooldown" : "60",
      "ScalingAdjustment" : "1"
     }
    },
    "WebServerScaleDownPolicy" : {
     "Type" : "AWS::AutoScaling::ScalingPolicy",
     "Properties" : {
      "AdjustmentType" : "ChangeInCapacity",
      "AutoScalingGroupName" : { "Ref" : "WebServerGroup" },
      "Cooldown" : "60",
      "ScalingAdjustment" : "-1"
     }
    },

    "CPUAlarmHigh": {
     "Type": "AWS::CloudWatch::Alarm",
     "Properties": {
      "AlarmDescription": "Scale-up if CPU > 70% for 2 minutes",
      "MetricName": "CPUUtilization",
      "Namespace": "AWS/EC2",
      "Statistic": "Average",
      "Period": "60",
      "EvaluationPeriods": "2",
```

```
    "Threshold": "70",
    "AlarmActions": [ { "Ref": "WebServerScaleUpPolicy" } ],
    "Dimensions": [
     {
      "Name": "AutoScalingGroupName",
      "Value": { "Ref": "WebServerGroup" }
     }
    ],
    "ComparisonOperator": "GreaterThanThreshold"
   }
  },
  "CPUAlarmLow": {
   "Type": "AWS::CloudWatch::Alarm",
   "Properties": {
    "AlarmDescription": "Scale-down if CPU < 50% for 10 minutes",
    "MetricName": "CPUUtilization",
    "Namespace": "AWS/EC2",
    "Statistic": "Average",
    "Period": "300",
    "EvaluationPeriods": "2",
    "Threshold": "50",
    "AlarmActions": [ { "Ref": "WebServerScaleDownPolicy" } ],
    "Dimensions": [
     {
      "Name": "AutoScalingGroupName",
      "Value": { "Ref": "WebServerGroup" }
     }
    ],
    "ComparisonOperator": "LessThanThreshold"
   }
  },

  "ElasticLoadBalancer" : {
   "Type" : "AWS::ElasticLoadBalancing::LoadBalancer",
   "Properties" : {
    "AvailabilityZones" : { "Fn::GetAZs" : "" },
    "Listeners" : [ {
     "LoadBalancerPort" : "80",
     "InstancePort" : "80",
     "Protocol" : "HTTP"
    } ],
    "HealthCheck" : {
     "Target" : "HTTP:80/geoserver/web/",
     "HealthyThreshold" : "2",
     "UnhealthyThreshold" : "8",
     "Interval" : "30",
```

```
        "Timeout" : "15"
      }
     }
    },

   "InstanceSecurityGroup" : {
    "Type" : "AWS::EC2::SecurityGroup",
    "Properties" : {
     "GroupDescription" : "Enable SSH access and HTTP from the load balancer
only",
      "SecurityGroupIngress" : [ {
       "IpProtocol" : "tcp",
       "FromPort" : "22",
       "ToPort" : "22",
       "CidrIp" : "0.0.0.0/0"
      },
      {
       "IpProtocol" : "tcp",
       "FromPort" : "80",
       "ToPort" : "80",
       "SourceSecurityGroupOwnerId" : {"Fn::GetAtt" : ["ElasticLoadBalancer",
"SourceSecurityGroup.OwnerAlias"]},
       "SourceSecurityGroupName" : {"Fn::GetAtt" : ["ElasticLoadBalancer",
"SourceSecurityGroup.GroupName"]}
      } ]
    }
   }
  },

  "Outputs" : {
   "URL" : {
    "Description" : "The URL of the website",
    "Value" :  { "Fn::Join" : [ "", [ "http://", { "Fn::GetAtt" : [ "ElasticLoadBalancer",
"DNSName" ]}]]}
   }
  }
}
```

Appendix B:

Review of the IndianaMap Geospatial Platform Cloud Service Test

By Tom Brenneman, Solution Engineer at Esri

The focus of this test was to gain information on the viability of using a cloud environment for hosting geospatial services. Unfortunately, the flaws in the test make it impossible to draw any meaningful conclusions regarding the viability of deploying geospatial services in the cloud. The main problem with the test is the number of variables that were not controlled. These uncontrolled variables make it impossible to have any confidence in the results.

The text from the User Evaluation Request that went out to all the testers reads as follows:

"Indiana University is working with the Indiana Geographic Information Council (IGIC) and Geographic Information Officer (GIO) to evaluate the deployment of common geospatial services in a commercial Cloud environment. The main outcome of this activity is to gain insight on the logistics and cost models for operational geospatial cloud computing."

The approach used in this test was to implement two similar web applications using different server technologies in the Amazon Web Services EC2 cloud. The server technologies were GeoServer and ArcGIS Server. These web applications were then distributed to members of IGIC to test. The applications reported the time that it took to perform a buffer query of parcels. The testers were given a specific time to test the applications and were asked to report the performance times for each application.

The flaws of this approach are listed below.

1. Differences in the client environments were not controlled. Each variable in the test environment that was not controlled reduces the reliability of the test results. The following items are example of the client configuration that should have been controlled.
    a. Browser
    b. Browser extensions
    c. Internal networking and security environments (Significant performance differences were observed between work and home networks on the same machine).
    d. Client Operating System.
    e. Client Hardware.
    f. Internet Service Provider.
2. Differences in server implementations (GeoServer vs. ArcGIS Server) introduce another variable that is not the focus of this test. The focus of this test was not to compare GeoServer and ArcGIS Server but to compare cloud geospatial services with geospatial services hosted on-premises or in a different hosting environment.

Therefore it would have been more beneficial to vary the hosting environment than it would have been to vary the server technology.

3. The test applications were implemented in very different ways. The ArcGIS Server applications drew all the parcels while the GeoServer application did not. The ArcGIS Server application also used a geometry service to generate the buffer where the GeoServer application generated it in the client application. These differences produced different perceptions of performance by the tester and it produced different loads on the server. These differences should have been controlled.
4. Old technology was used for ArcGIS Server. The test used ArcGIS Server 10.0 instead of 10.1 SP1. Because of the performance improvements in 10.1, the results could have been significantly different. ArcGIS Server 10.1 SP1 was available at the time in a preconfigured EC2 instance, so the testing group had to specifically choose an older version of the technology for the test. This implies a bias in the test environment for the older technology.

Future recommendations

If the goal is to test the viability of geospatial services in a cloud environment then the test should focus on the deployment of the server solution in a cloud vs. an on-premises solution. Other variables should be controlled so that conclusions can be drawn specifically about the cloud deployment. A single server configuration should be deployed to all the test environments. Perhaps the server configuration could be deployed to an on-premises server and an Amazon EC2 machine. Great care should be taken to keep the configurations and hardware specifications as similar as possible and using the latest technology. Using old technology puts the analysis out of date on delivery.

Then identical geospatial services should be deployed to each system. These services should include common geospatial services like: mapping, queries, geocoding, and geoprocessing (analysis). A testing application could then be developed to call these services in a standard way. This application would run in a test harness to ensure that each run of the application is identical. The services in the testing application should be called in a way to mimic a standard user interaction with a web site. For example, the user might view the full extent, then zoom in, and then query the data. This is three web service calls that could be called in succession as a group. This group of web service calls is a logical user workflow. These user workflows should be automated with several testing applications. Each of these testing applications would then test the user workflows against the different deployments of the geospatial services. The test harness should also execute a large number of requests in parallel to test the scalability of each system. The benefit here is the client environment is controlled to the client environment of the test harness. Additionally this test would speak specifically to the viability of geospatial services in the cloud by comparing them to an alternative.

This approach follows more of an industry standard approach to testing applications and web services. Commercially available test harness and scripting environments are available for exactly this purpose. Future tests should take advantage of this technology and a more scientific approach where variables are controlled.