AWS Startups Blog

Scaling on AWS Part I: A Primer

by AWS Admin | on 25 NOV 2015 | in Startup | Permalink | Share

https://aws.amazon.com/blogs/startups/scaling-on-aws-part

By David Kuo, Solutions Architect, AWS

Scaling an on-premise infrastructure is hard. You need to plan for peak capacity, wait for equipment to arrive, configure the hardware and software, and hope you get everything right the first time. You heard that deploying in the cloud can address these headaches. So either your boss or you shopped around and decided to give Amazon Web Services (AWS) a try.

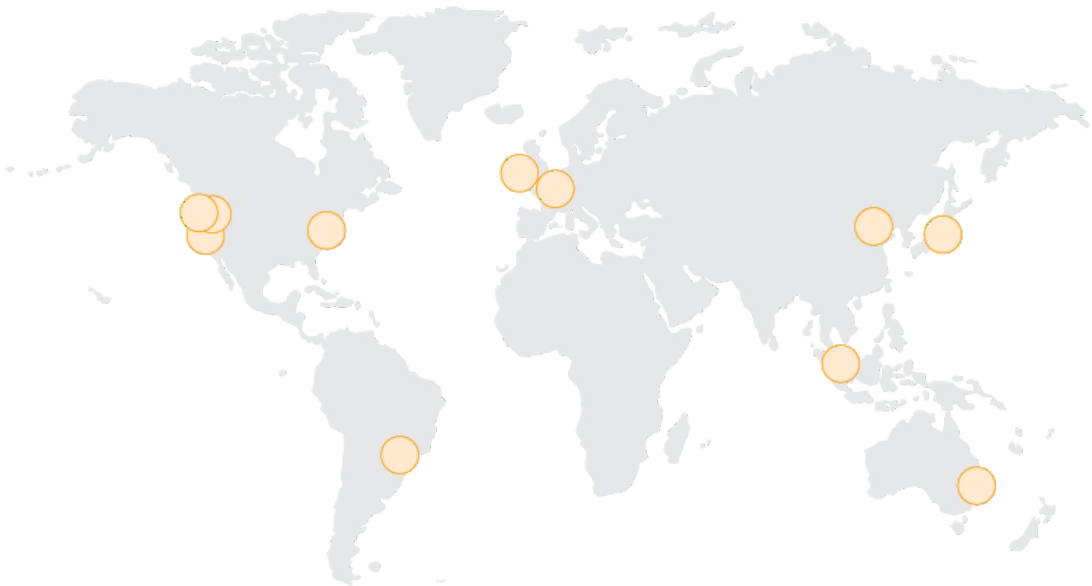
Now it's your job to scale this AWS infrastructure properly and you wonder if there are any best practices or general guidelines when it comes to scaling on AWS. I wrote this blog series with the intention to provide just that. Scaling on AWS is indeed simple compared to traditional infrastructure if you know the relevant AWS services and provision them as you need them. I will walk through a scaling example as an application developer and discuss not only why I introduced certain AWS services into my architecture but also the motivation behind the decisions. Let the adventure begin!

AWS Background Knowledge

Before going through a scaling example, let's see how AWS global infrastructure is designed to support highly scalable workloads.

Regions

Starting from a high-level view of AWS infrastructure, the first thing to notice are regions. Regions are physical geographical locations where AWS has clusters of data centers.



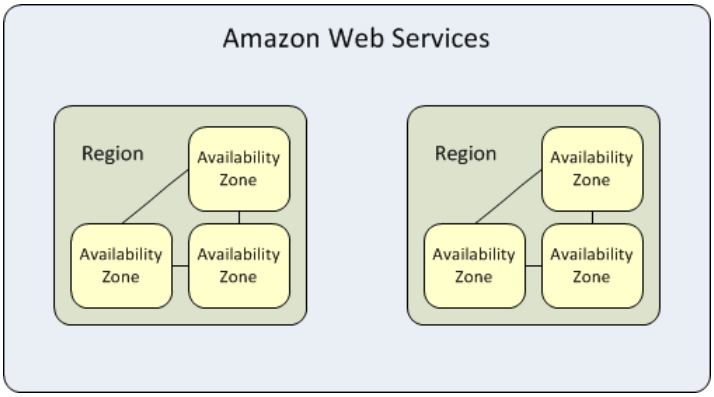
Currently, AWS has 11 regions. And we'll be adding five more: (Korea, India, Ohio, UK, and a second region in China). Customers may choose a region based on several criteria:

1. Proximity to end user
2. Data regulatory requirements
3. Expansion strategy
4. Cost
5. Service availability

In addition, AWS has made it easy to migrate your application and data across regions. With a utility billing model and the tools and services to help you migrate between regions, you have a lot of options when you need to relocate or right-size existing AWS infrastructure later on.

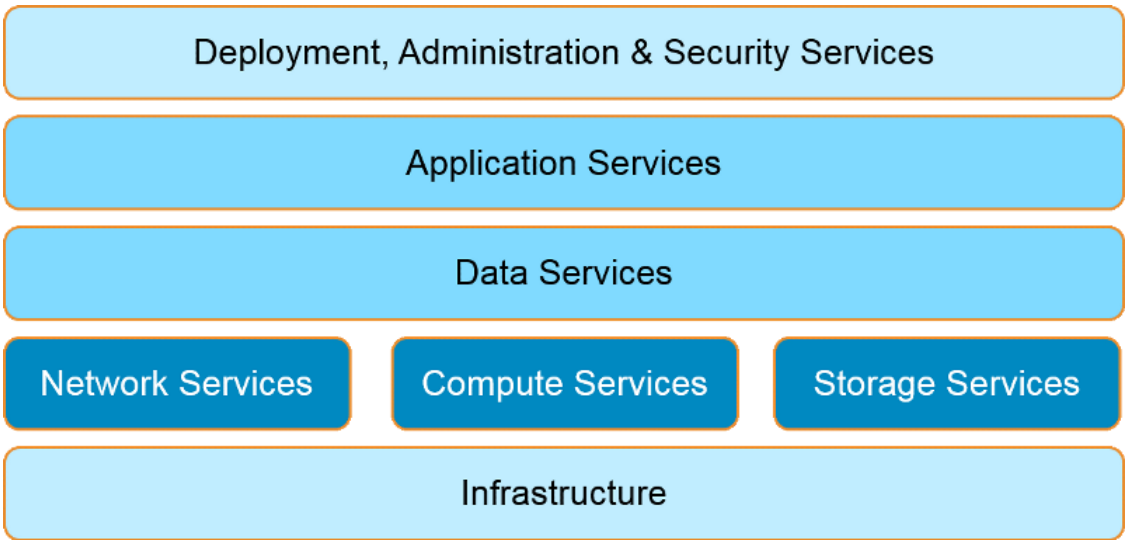
In general, it is desirable to deploy in a region closest to your end users to decrease network latency. For my application, I will assume a potential user base spread across North America. AWS has three regions in North America: N. Virginia, N. California, or Oregon. I will deploy in Oregon since it is a little cheaper and close to where I live.

Availability Zones
Inside a region, there are two or more Availability Zones, also commonly referred to as AZs. AZs are independent clusters of data centers designed to be insulated from failures in other AZs. AZs use independent power grids, sit on different fault lines and flood plains, and provide single digit millisecond connectivity to other AZs in the same region.



While architecting for high availability may not be critical when you first get started on AWS, you eventually want to deploy your application over multiple AZs so it can withstand various types of failures—including rare location outages. Just remember that the AWS infrastructure supports highly available architectures and we'll revisit this topic in a future blog post.

Overview of the AWS Platform
Since 2006, AWS has grown into a rich set of services that act as building blocks to help you construct a scalable, cost-efficient, and secure infrastructure. The following diagram shows broad categories of AWS services.



Global Infrastructure includes regions and Availability Zones that we just discussed. This global footprint gives you the ability to deploy near your customers and migrate your workload as needed.

On top of global infrastructure, AWS has services in compute, storage, networking, and security and administration. Using services in these core categories, you can launch EC2 instances, (Amazon EC2) which are analogous to virtual machines, secure and monitor AWS resources (AWS Identity and Access Management or IAM, Amazon CloudWatch), and define unique network topologies (Amazon VPC). These services let you build a virtual data center in the AWS cloud.

In the data services category, AWS offers managed relational and non-relational database services such as Amazon Relational Database Service (RDS) and Amazon DynamoDB. Managed services performs administrative tasks for you so you don't need to spend resources on backups, software maintenance, or manage failovers. There is also Amazon RedShift, which is a managed database warehouse service that uses columnar storage to achieve high efficiency and can cost as little as less than $1000 per year per TB of storage. If you need to process and analyze big data, AWS platform also offers Amazon Elastic Map Reduce (EMR), a hosted map reduce service, Amazon Kinesis, a service that enables real time data analytics, and Amazon Machine Learning, a service that makes it easy for you to consume machine learning technology. I will dive deeper on these technologies in a later post.

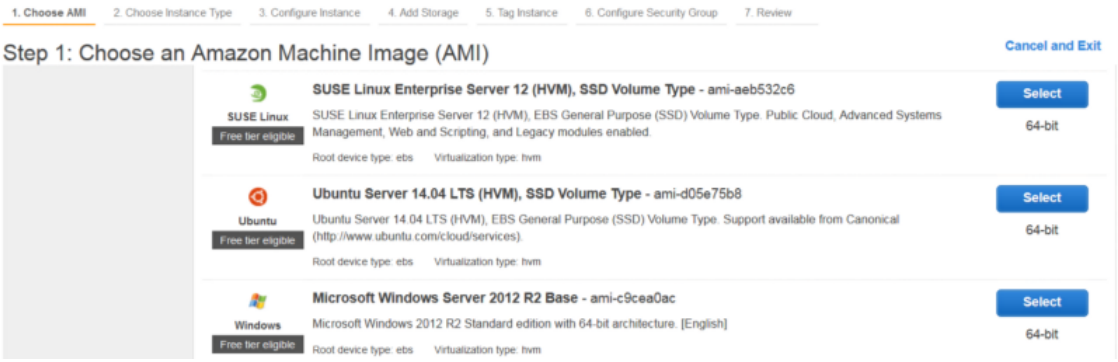
In addition, AWS provides a number of application services. These services are common software components that traditionally have to be homegrown or outsourced but provide no differentiating value for your business. On AWS, you can save time and money by choosing from a variety of generic components in a plug-and-play fashion and pay for them on a utility basis. These generic components include database services, a database warehouse service, a caching

service, queueing service, a notification service, an email service, and many others.

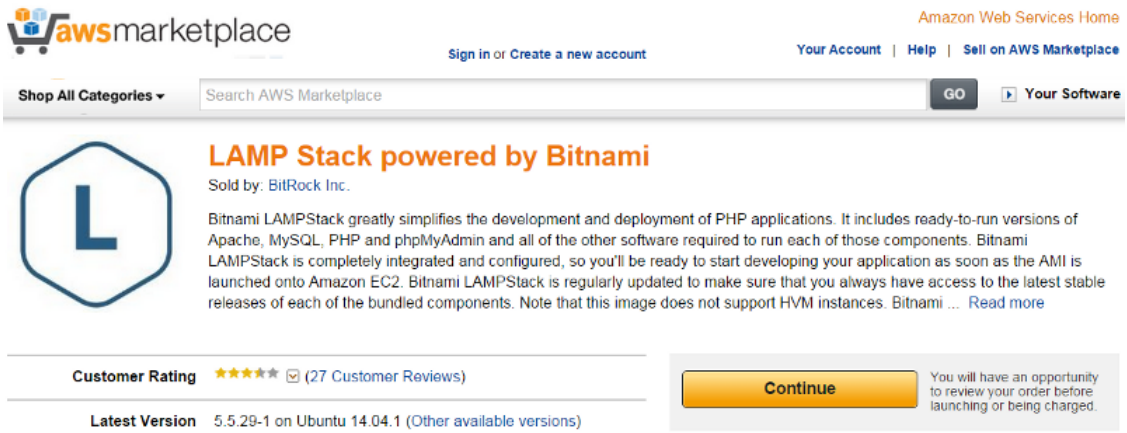
Bottom line is that the AWS platform empowers you with 50+ services so you can build your infrastructure quickly and cost effectively.

Day One, User One

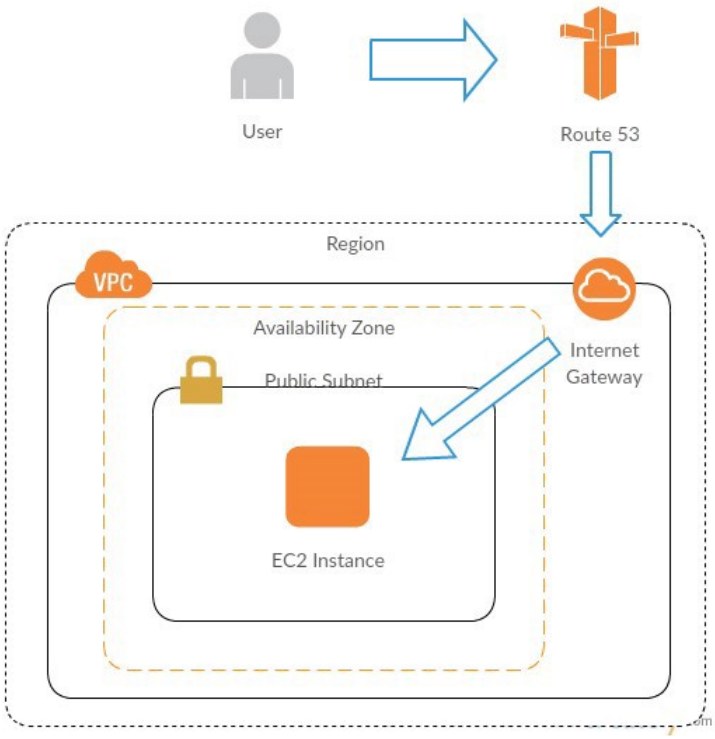
Now that you know AWS has a vast infrastructure and a host of services to help you build a scalable application, where do you start? Well if you just want to get your feet wet, you could start by deploying a simple application on a single box. For example, you may be a LAMP stack developer or a .NET developer and decide to host your web server, application server, and database on the same EC2 instance. AWS makes it very simple for you to create virtual machines by providing commonly used technology stacks in the form of Amazon Machine Images (AMI). AMIs are analogous to snapshots of virtual machines. When you launch your virtual machines using the AWS Management Console or AWS CLI, you can choose to launch a Linux- or Windows-based server by selecting a particular AMI.



In addition, you can choose from a variety of AMI on AWS Marketplace. In fact, a commonly used stack such as LAMP is available.



Next, you want to deploy this EC2 instance somewhere inside a secure virtual network, such as a public subnet inside a VPC. To host DNS records for your server, you could use Amazon Route53.



I know lots of acronyms were just used without explanation. Let's see some definitions before moving on.

Amazon Elastic Compute Cloud (Amazon EC2)

- An EC2 instance is equivalent to a virtual machine. Instances are grouped into families with different ratios of compute, storage, and network resources to optimally serve your unique workload needs. In general, it is good to start with an instance type from the General purpose instance family, which has a balanced ratio of resources. Through monitoring workload resource needs, I'll have opportunities to switch to more appropriate families for better performance and cost efficiency. In my initial architecture, I will deploy a t2.medium into a public subnet inside a virtual private cloud (VPC).

Amazon Virtual Private Cloud (Amazon VPC)

- Amazon VPC lets you provision a logically isolated section of the AWS cloud where you can launch AWS resources in a virtual network that you define. You have complete control over IP address range, subnets, and routing rules.

VPC subnet

- A subnet is a slice of the overall VPC IP address range and tied to an Availability Zone. Virtual machines deployed into a public subnet can have randomly assigned public IPs and are visible to other Internet hosts. To make a subnet public, you need to create a routing rule for the subnet and direct internet-bound traffic to egress through an internet gateway, which is a horizontally scaled, redundant, and highly available VPC component that allow communication between instances in VPC and the internet.

Amazon Route53

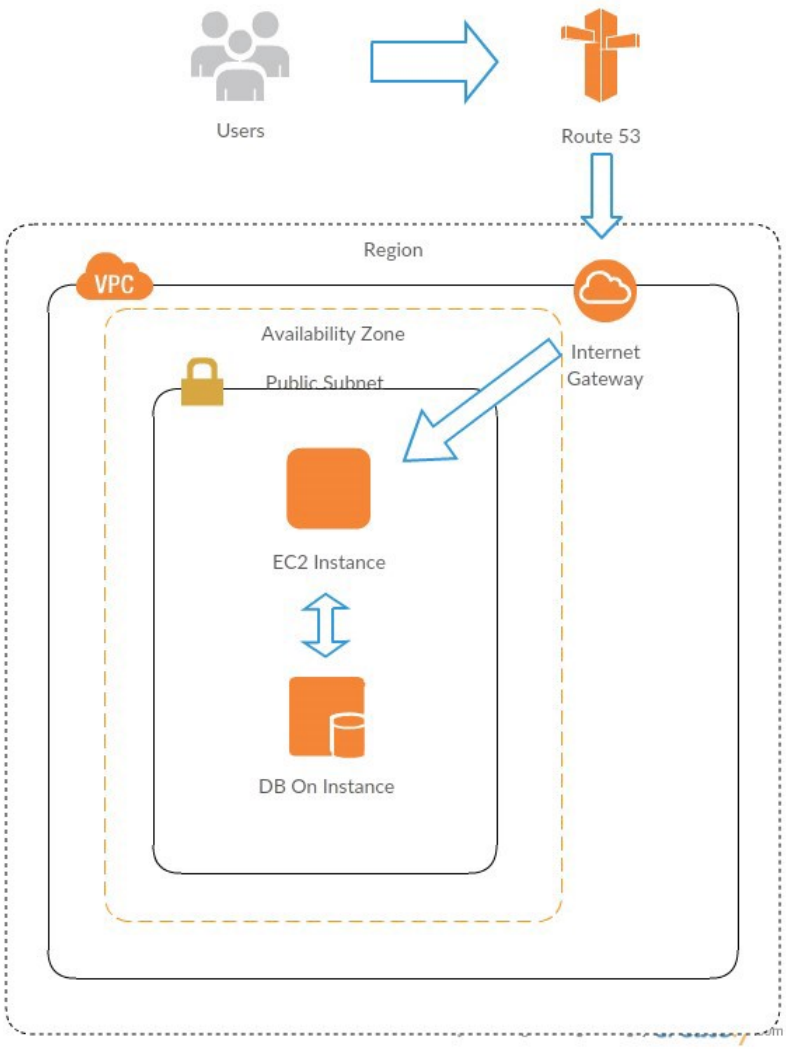
- Route 53 is a highly available and scalable cloud Domain Name System (DNS) web service. It is designed to give developers and businesses an extremely reliable and cost-effective way to route end users to Internet applications by translating names like www.example.com into the numeric IP addresses like 192.0.2.1 that computers use to connect to one another.

Day Two, User > 1

The original architecture is fine until your traffic ramps up. Now you wonder how you can scale. A quick solution is to vertically scale by spinning up a bigger box. EC2 instances can have up to 244 GB of RAM or 40 virtual cores, and upgrading to a more powerful instance is easy. For instances that use Amazon Elastic Block Store (Amazon EBS) as root volume, you would stop an EC2 instance, change instance type, and restart the instance. The entire process should take minutes.

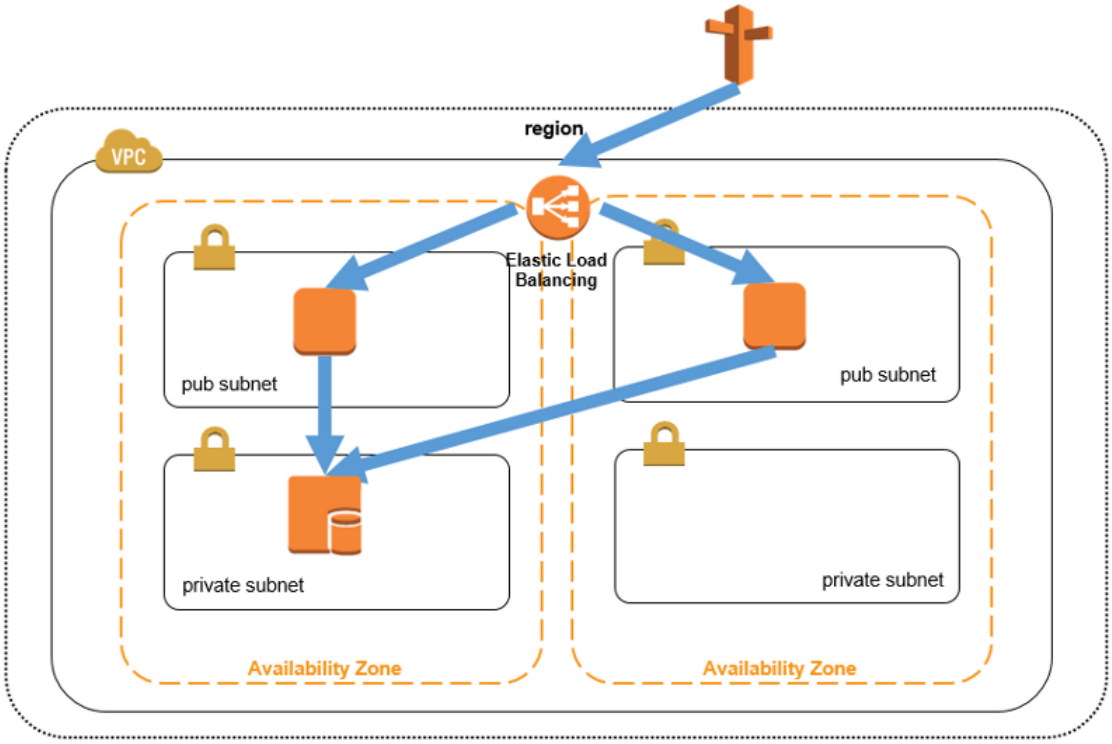
AWS EC2 offers instance types up to 40 vCPUs and 244 GB of memory, but vertical scaling will run into a ceiling sometime. Besides, there are other issues at hand with this architecture. First, you are using a single machine which means you don't have a redundant server. Second, your machine resides in a single AZ, which means your application health is bound to a single location.

To address the vertical scaling challenge, I recommend you start with decoupling your application tiers. Application tiers are likely to have different resource needs and those needs might grow at different rates. By separating the tiers, you can compose each tier using the most appropriate instance type based on different resource needs. When I apply the concept of decoupling to the original architecture, which hosted the entire stack on a single EC2 instance, the new architecture will look like the following diagram.



Next, try to design your application so it can function in a distributed fashion. For example, you should be able to handle a request using any web server and produce the same user experience. Store application state independently so that subsequent requests do not need to be handled by the same server. Once the servers are stateless, you can scale by adding more instances to a tier and load balance incoming requests across EC2 instances using Elastic Load Balancing (ELB). ELB is a load balancing service that distributes incoming application traffic across EC2 instances. It is horizontally scaled, imposes no bandwidth limit, supports SSL termination, and performs health checks so that only healthy instances receive traffic. ELB load balancers can be public facing or closed off to the Internet.

In the following new architecture I am scaling horizontally. It's a cost-effective way to handle high volume traffic. Rather than using exotic, high-performance, and expensive servers, you use a pool of low-cost commodity servers to serve requests.



In the next post, I will continue to decouple and evolve the architecture to serve a growing traffic. Database options and additional AWS services will be discussed.

Summary

1. You can get started on AWS by understanding the global infrastructure and core services around networking and compute such as Amazon VPC and Amazon EC2.
2. Vertical scaling is a viable scaling strategy, but it has a limit. You want to use horizontal scaling to handle high volume traffic cost effectively.
3. To use horizontal scaling, your application should have a distributed architecture and application tiers should be decoupled. This way you are able to avoid over provisioning and maximize the use of different EC2 instance types.

Continue reading: **Scaling on AWS (Part 2): > 10K Users**

References

Scaling Up to Your First 10 Million Users

Follow us: @AWSStartups

TAGS: Advice from AWS Solution Architects, getting started, Technical Use Cases