Global Knowledge®

Expert Reference Series of White Papers

Tips and Tricks to Speed AWS Deployment

Tips and Tricks to Speed AWS Deployment

Rich Morrow, Global Knowledge Instructor, Cloud and Big Data Analyst

Introduction

Like many organizations, yours probably has investigated some of the services offered by Amazon Web Services (AWS), and you've probably identified several that you would like to use. Although the benefits of increased agility, developer productivity, pay-as-you-go pricing, and overall cost savings are extremely attractive, you might wonder where to start. What pitfalls exist, and how can you avoid them? How can you best save time and money?

If you're a developer, a development manager, or a CIO who'd like to launch an AWS-hosted service like a pro, keep reading.

Low-Risk, High-Reward AWS Deployments

Trying to run before you can walk almost always ends in cuts and bruises. Your first project to launch on AWS should be something that is either easy to migrate or easy to build from scratch.

Greenfield projects (brand new ones that don't have any legacy) are by far the easiest to launch on any platform. When launching a greenfield project on AWS, you'll want to leverage existing architectures, Amazon Machine Images(AMIs), which are pre-built virtual machine images), and/or CloudFormation Templates (pre-built stacks of services that deploy from a template). The latter two are discussed in detail below.

Amazon publishes dozens of whitepapers and architecture diagrams, both of which you'll definitely want to review. If you're trying to build a standard system like a CMS, e-commerce website, ad server, or batch-processing engine, then you're likely to find a CloudFormation template that will get you 90% or more of the way, and you could be up and running in as soon as a day. With a greenfield project, you have the luxury of architecting "The AWS way," which is loosely coupled, highly-available, and scaled on demand.

Note: In the following paragraphs, we may be introducing some AWS terms or services that you're not familiar with. These services are covered later in the document, so feel free to skip ahead and come back for context if you need.

If you're looking to migrate all or part of an existing app, you'll want to focus on something that will let you get AWS experience with low risk. Content and media serving, internal CMS or collaboration engines, and proof-of-concept projects are all excellent choices for a first effort. The most tried and true path to get one of these apps moved from your systems to AWS is by doing it in three distinct phases: Forklift –> Integrate –> Optimize.

Forklift refers to just picking up your existing stack (maybe a combination of load balancers, servers, and databases) and moving that entire stack into AWS. In the forklift phase, you're single goal is to take what you have and deploy on AWS. You're simply trying to match your existing hardware and software to what AWS offers, and migrate over. In this phase, you're focusing on the AMIs, and matching your systems (like custom built load balancers and databases) to AWS resources (like Elastic Load Balancers and RDS—the database product).

Once you've forklifted an application over, you can easily validate its operation by running your existing regression, load, and security tests against the AWS stack. If you don't have these tests, you need to make their creation a priority, and ensure that they truly validate your existing stack's operation. After your AWS stack has passed those tests, you can do a formal cutover with DNS, shut down your old stack, and begin the "Integrate" phase on your AWS stack.

In the Integrate phase, you're now focused on truly integrating your app into AWS. You'll be looking for opportunities to do high availability or "HA" (ensuring that no single points of failure exist in your stack), save costs, and extend your security and disaster recovery capabilities. During this phase, you'll begin utilizing AWS features like autoscaling groups for your web and app servers, S3, and CloudFront for low-latency, cost effective serving of static content, and RDS multi-AZ hot failover for the database layer. You'll also begin fine-tuning your security, and may even leverage technologies like VPC, VPNs, and/or Identity and Access Management (IAM) (which is AWS' fine-grained access controls for users, groups, and servers/services).

After you've completed the integration phase, you may want to hyper-optimize your application in the Optimization phase. In Optimization, you're trying to squeeze out every last efficiency and cost savings that AWS offers. Some common activities in this phase are:

- Code refactors (which probably push out some functionality like queues, email, and notifications to AWS services like SQS, SES, and SNS)
- Advanced EC2 strategies like parallel processing, leveraging reserved, and spot instances
- Integration with more esoteric AWS services like Simple Workflow (SWF) and Elastic Map Reduce (EMR) –hosted Hadoop

By going through the Optimization phase with a low-risk product, you'll not only have fully integrated one full product into AWS, but you'll have acquired valuable learning that you can then apply to more complicated or risky efforts.

EC2—The Bulk of What You'll Do

One of the first and most important services you'll be leveraging in AWS is EC2—virtual servers that come in various hardware footprints. Although the RAM and disk footprints are easy to understand, AWS uses the moniker of Elastic Compute Unit or "ECU" for the CPU baseline. An ECU equates to a 2007 1.0-1.2GHz Opteron or Xeon processor—an important fact to keep in mind if you're trying to match existing server specs to EC2.

EC2s have two aspects to them, the instance type or "hardware footprint" that was covered previously, and the AMI or "software footprint," which is the particular OS, applications (like Apache, JBoss, or custom PHP or Python code), and all configuration. Instance types and AMIs are independent of each other—you can launch most AMIs into most instance types. When deciding on an AMI, you definitely want to consult both the community AMI library as well as the AMI marketplace, because there's a good chance that the exact software you want to deploy exists in one or the other. The only caveat to using either is to carefully vet the provider of the AMI—make sure it comes from a reputable company or individual you trust.

AMIs from the marketplace are an excellent way to test drive both open source and commercial software—you can spin the servers up in just a few minutes, use them for as little as an hour, and pay as little as a few pennies. As with nearly all AWS services, there is no up-front cost and you only pay for the hours that the software is kept running. You can also load your own applications and custom code into a base AMI (OS only), and then burn your own AMI, which now contains all your modifications and can be used to spin up exact copies of that server in minutes.

If you're like most customers, when you start running AWS at scale, you'll notice that 70% or more of your average monthly bill is attributed to EC2 costs alone. To rein in those costs, most customers use a combination of autoscaling (which is completely free and lets you expand and contract your EC2 farms based on real-time user demand), and alternative billing models like reservations and spot.

All AWS services which utilize resizable VMs (like EC2, RDS, and Elasticache) can be paid for using "reservations." A reservation functions a lot like a stock option—you pay an up-front fee for the option (but not obligation) to purchase a given VM for a lower nominal hourly cost than the "no-obligation" on-demand price. If your server demand is well known (reservations are for specific instance types in specific data centers), purchasing a one or three year reservation for your "always on" needs can save 30% or more over the entire term of the reservation.

A much more advanced use case is spot instances. With spot instances, you're bidding on unused EC2 capacity—you put in a bid for a specific instance and if the spot price goes below your bid, you get the instance. The flip side of that is that if the spot price goes *above* your bid, *the instance is taken away from you without warning and given to other customers.* Although that constraint makes spot appear unusable, there are many cases where it makes perfect sense—dev and test environments, batch processing architectures, and offline data analysis to name just a few.

It's worth noting that whether you're using on-demand, reserved, or spot instances, the underlying server is the exact same—the three billing models only dictate how you pay for that server (and in the case of spot, that it can go away at any time).

A very important aspect of EC2 instances is security. To help you manage security, AWS provides two important tools that you'll want to leverage ASAP—Security Groups and public/private EC2 key pairs.

AWS Security Groups (SGs) can be applied to the same resources which support reserved billing models—EC2, RDS, and Elasticache. SGs are point-and-click stateful firewalls that can be defined once and applied to many instances. A single instance can also have one or more SGs applied to it, and they can be modified at runtime to add and remove access to that server. Like everything AWS does, SGs are locked down to "no access"—you'll need to create a new SG (do NOT modify the "default" SG) for at least each tier of your application—perhaps opening port 80 and 443 to the world for the web servers and only port 8080 (from the web server's SG) on the app tier servers.

In addition to SGs, AWS requires public-private key pairs for at least the first Secure Shell (SSH) access to your EC2 server. It's highly recommended to both leave this key pair authentication on, and rotate key pairs at least every two to three months, or when an employee leaves the organization.

Security

If you're used to managing only a handful of servers, you probably allow outside SSH access to each. In AWS, where you will likely have dozens of servers running at any given time, you'll probably want to leverage a bastion host or "jump station" server. This bastion host (or set of bastion hosts) becomes the only machine that allows outside SSH access, and to get to other machines in your farm, you first have to SSH into the bastion and then SSH to the other machines. Your bastion host machine(s) then become the one(s) on which you lock down security, possibly installing OS-level firewalls, anti-virus, and intrusion detection systems (IDSes) such as Snort or Tripwire. Again, SGs provide an easy way to enforce bastion host (outside access) and internal server (access only from bastion host) SSH.

Another important tool to use for security in AWS is the IAM service. IAM, like autoscaling, is totally free to use, and it provides at least "action" level control (e.g.: **get**Object and **put**Object in S3) for every service. For some services (such as S3 and EC2), you can even now control access to individual resources like *launch Instance* **for only specific AMIs or instance types**—extremely useful to sandbox new users and ensure that they cannot run up your bill. Each user in IAM has their own credentials (which can be revoked independently), and you can cherry pick whether they have API access, Web Console access or both. One of the smartest ways to use IAM accounts by enforcing one of the most important paradigms in security—"the rule of least access." Since IAM is a "default deny" model, the only resources and actions that a user gets are the ones you, as the account admin grant to them.

Savvy AWS users will actually take their root account (the main account that has complete access to all actions on all resources), burn its credentials on a USB and then lock it in a company safe (after, of course, they used it to create all of the IAM accounts that they needed). Whatever you do, make sure that individual users and other EC2 instances never use that root account for interaction with AWS—if that account becomes compromised, *everything* you've built with AWS is compromised.

A final important security tool available to you (which is 100% free) is the Virtual Private Cloud (VPC). With VPC, AWS users can cordon off a private, virtual cloud in AWS. This private cloud has "edge level" control over who can get into and out of the whole set of resources you deploy into it. It also gives you:

- Detailed control over public and private IP addresses
- Egress as well as ingress rules for SGs
- The ability to connect a Virtual Private Gateway (VPG) which allows for secured VPN connections to your on-premise data center
- The ability to set up routing rules which direct traffic based on source, destination, and traffic type

VPC provides a ton of security for only a small additional amount of complexity, and I try to use it in every client project I launch in AWS.

CloudFormation for POCs and Quick Deploys of Full Stacks

Although AMIs allow you to easily spin up a single server, pre-packaged with the running software you need, you often need an entire stack of machines and services to build out a full production release (for instance, a standard 3-tier web app which needs a load balancer, web and app tiers—each with their own farm of EC2s, auto scaling groups, load balancers between—and maybe a multi-AZ RDS deployment to store the data). For this need, AWS offers another service—CloudFormation, which again, comes with zero cost.

With CloudFormation, you define all the resources you need in a JSON template, and then you can deploy that template in any Region. Because the template is a single JSON document, you can even check the template into version control, allowing it to be the single source of documentation for what you deploy. Those who have used commercial configuration management tools like Puppet, Chef, or Ansible will be right at home in CloudFormation's syntax. The only possible downside to CloudFormation is that it is an AWS-only tool, whereas other general-purpose tools will work both in and out of AWS.

Storage Options

At the storage layer, AWS shines exceptionally bright—giving you several options to choose from, each with various use cases.

First and foremost is S3, which many customers use simply to serve popular content to many concurrent users. One of the best ways to use S3 is to serve any static web-based content such as JavaScript, CSS, static HTML, and images. S3 is a "Write Once Read Many" (WORM) file system, meaning it is especially well suited for low-latency, high concurrent volume read patterns—exactly the situation you have with these web assets in which your Apache web server is probably being (incorrectly) used. By moving this popular content out to S3, you not only free up your web server to do more complicated tasks faster, you also greatly increase the number of users who can concurrently use your website, while mostly likely making the overall experience for these web and mobile users much faster.

To even further speed up your serving of these assets, you could use CloudFront—AWS's Content Delivery Network (CDN). Although it's not as feature rich as competitor solutions like Akamai, CloudFront provides the same basic functionality—pushing static content closer to users, greatly speeding their interaction with your web product. Like almost all services AWS offers, CloudFront is zero cost up-front and zero commitment. You can try it to decide whether it's suitable for you, and turn it back off, all in a day—something Akamai will likely never let you do.

No discussion of storage would be complete without talking about a Relational Database Management System (RDBMS). The heart of your real-time web or mobile product is likely to be an RDBMS like MySQL, Oracle, Postgres, or SQL Server.

For this layer, AWS offers the RDS product which is a fully managed RDBMS, complete with client-level control (you get no SSH into the server) to create, select, insert, and perform all other operations that you're used to from a command line client like the MySQL client.

RDS should be your go-to tool when trying to implement a database in AWS, but it definitely comes with some limitations that may not make it suitable for you. Although it makes the patching, backup, and multi-AZ HA as simple as a few mouse clicks—in exchange for that convenience, you give up considerable control. You cannot, for instance, adjust my.cnf-level properties, or tune anything at the OS level. There is no ability to SSH into the server, and although RDS comes in re-sizeable footprints (such as EC2), your database must fit in 3TB or less.

For this reason, some AWS users must go the route of building their own database on EC2, probably using a set of striped Elastic Block Store or EBS volumes, and possibly even using provisioned Input/Output Operations Per Second (IOPS) to make the database extremely responsive (albeit at a higher cost). Whenever possible, one should always use the AWS service, which will likely be more cost effective and feature rich for the common operations you'd like to perform.

A final storage product that's worth mentioning is DynamoDB, which many users have found extremely useful (as evidenced by it being the fastest growing service ever in AWS history). DynamoDB is a managed, SSD-backed, distributed Key-Value NoSQL engine—essentially providing the same base functionality as Memcached, but with much higher performance and a richer feature set. Aside from a 64KB row limit, DynamoDB really has no limits—it can serve any read or write volume, and there are no limits to number of tables, number of rows per table, or raw storage volume.

DynamoDB is an excellent fit for data like user web sessions, or "hot tables," that your RDBMS is struggling to serve, or systems where extremely low latency is needed.

Although I love Memcached and have used it in many production environments, when building systems in AWS, one should really try to avoid their ElastiCache product (which provides a hosted Memcached cluster), and instead lean towards DynamoDB. ElastiCache still only runs in a single AZ, making it a potential single point of failure for the critical areas where you want to deploy it.

Learn More

Learn more about how you can improve productivity, enhance efficiency, and sharpen your competitive edge through training.

- AWS Essentials
- Architecting on AWS
- Developing on AWS
- Systems Operations on AWS

Visit **www.globalknowledge.com** or call **1-800-COURSES (1-800-268-7737)** to speak with a Global Knowledge training advisor.

About the Author

Rich is a full-stack generalist with particular in-depth knowledge and love for Cloud and Big Data technologies. When he's not flying around the country for Global Knowledge providing training on AWS and Hadoop to the Fortune 500, he's probably writing or speaking about Cloud, Big Data, and Mobile topics for GigaOM.

Additional Reading

I hope the information provided in this white paper helps make your AWS deployments less risky and more rewarding. If you're inspired to learn more about AWS, a few links to some helpful resources are provided below.

- AWS Regions, AZs and Edge Locations Prezi
- AWS Documentation
- EC2 Documentation
- AutoScaling Documentation
- Identity and Access Management (IAM) Documentation
- CloudFormation Documentation
- Elastic MapReduce (EMR) Documentation
- ELB Documentation
- S3 Documentation
- RDS Documentation
- DynamoDB Documentation
- Simple Workflow Documentation
- Global Knowledge AWS Training