An Open Guide

The Open Guide to Amazon Web Services

Chat Slack   Join us!

Credits   Contributing guidelines

Table of Contents

Purpose

- Why an Open Guide?
- Scope
- Legend

AWS in General

- General Information
- Learning and Career Development
- Managing AWS
- Managing Servers and Applications

Specific AWS Services	Basics	Tips	Gotchas
Security and IAM			
S3			
EC2			
CloudWatch			
AMIs			
Auto Scaling			
EBS			
EFS			
Load Balancers			
CLB (ELB)			
ALB			
Elastic IPs			
Glacier			
RDS			
RDS MySQL and MariaDB			
RDS Aurora			
RDS SQL Server			
DynamoDB			
ECS			
Lambda			
API Gateway			
Route 53			

Specific AWS Services	Basics	Tips	Gotchas
CloudFormation			
VPCs, Network Security, and Security Groups			
KMS			
CloudFront			
DirectConnect			
Redshift			
EMR			
Kinesis Streams			
Device Farm			
IoT			
SES			
Certificate Manager			

Special Topics

- High Availability
- Billing and Cost Management
- Further Reading

Legal

- Disclaimer
- License

Figures and Tables

Tools and Services Market Landscape AWS Data Transfer Costs

- Figure: Tools and Services Market Landscape: A selection of third-party companies/products
- Figure: AWS Data Transfer Costs: Visual overview of data transfer costs
- Table: Service Matrix: How AWS services compare to alternatives
- Table: AWS Product Maturity and Releases: AWS product releases
- Table: Storage Durability, Availability, and Price: A quantitative comparison

Why an Open Guide?

A lot of information on AWS is already written. Most people learn AWS by reading a blog or a "getting started guide" and referring to the standard AWS references. Nonetheless, trustworthy and practical information and recommendations aren't easy to come by. AWS's own documentation is a great but sprawling resource few have time to read fully, and it doesn't include anything but official facts, so omits experiences of engineers. The information in blogs or Stack Overflow is also not consistently up to date.

This guide is by and for engineers who use AWS. It aims to be a useful, living reference that consolidates links, tips, gotchas, and best practices. It arose from discussion and editing over beers by several engineers who have used AWS extensively.

Before using the guide, please read the **license** and **disclaimer**.

Please help!

This is an early in-progress draft! It's our first attempt at assembling this information, so is far from comprehensive still, and likely to have omissions or errors.

Chat Slack

Please help by **joining the Slack channel** (we like to talk about AWS in general, even if you only have questions — discussion helps the community and guides improvements) and **contributing to the guide**. This guide is *open to contributions*, so unlike a blog, it can keep improving. Like any open source effort, we combine efforts but also review to ensure high quality.

Scope

- Currently, this guide covers selected "core" services, such as EC2, S3, Load Balancers, EBS, and IAM, and partial details and tips around other services. We expect it to expand.
- It is not a tutorial, but rather a collection of information you can read and return to. It is for both beginners and the experienced.
- The goal of this guide is to be:
  - **Brief:** Keep it dense and use links
  - **Practical:** Basic facts, concrete details, advice, gotchas, and other "folk knowledge"
  - **Current:** We can keep updating it, and anyone can contribute improvements
  - **Thoughtful:** The goal is to be helpful rather than present dry facts. Thoughtful opinion with rationale is welcome. Suggestions, notes, and opinions based on real experience can be extremely valuable. (We believe this is both possible with a guide of this format, unlike in some other venues.)
- This guide is not sponsored by AWS or AWS-affiliated vendors. It is written by and for engineers who use AWS.

Legend

- Marks standard/official AWS pages and docs
- Important or often overlooked tip

- "Serious" gotcha (used where risks or time or resource costs are significant: critical security risks, mistakes with significant financial cost, or poor architectural choices that are fundamentally difficult to correct)
- "Regular" gotcha, limitation, or quirk (used where consequences are things not working, breaking, or not scaling gracefully)
- Undocumented feature (folklore)
- Relatively new (and perhaps immature) services or features
- Performance discussions
- Lock-in: Products or decisions that are likely to tie you to AWS in a new or significant way — that is, later moving to a non-AWS alternative would be costly in terms of engineering effort
- Alternative non-AWS options
- Cost issues, discussion, and gotchas
- A mild warning attached to "full solution" or opinionated frameworks that may take significant time to understand and/or might not fit your needs exactly; the opposite of a point solution (the cathedral is a nod to Raymond's metaphor)
- Colors indicate basics, tips, and gotchas, respectively.
- Areas where correction or improvement are needed (possibly with link to an issue — do help!)

General Information

When to Use AWS

- AWS is the dominant public cloud computing provider.
  - In general, "cloud computing" can refer to one of three types of cloud: "public," "private," and "hybrid." AWS is a public cloud provider, since anyone can use it. Private clouds are within a single (usually large) organization. Many companies use a hybrid of private and public clouds.
  - The core features of AWS are infrastructure-as-a-service (IaaS) — that is, virtual machines and supporting infrastructure. Other cloud service models include platform-as-a-service (PaaS), which typically are more fully managed services that deploy customers' applications, or software-as-a-service (SaaS), which are cloud-based applications. AWS does offer a few products that fit into these other models, too.
  - In business terms, with infrastructure-as-a-service you have a variable cost model — it is OpEx, not CapEx (though some pre-purchased contracts are still CapEx).
- AWS's annual revenue was **$7.88 billion** as of 2015 according to their SEC 10-K filing, or roughly **7%** of Amazon.com's total 2015 revenue.
- **Main reasons to use AWS:**
  - If your company is building systems or products that may need to scale
  - and you have technical know-how

- – and you want the most flexible tools
  - – and you're not significantly tied into different infrastructure already
  - – and you don't have internal, regulatory, or compliance reasons you can't use a public cloud-based solution
  - – and you're not on a Microsoft-first tech stack
  - – and you don't have a specific reason to use Google Cloud
  - – and you can afford, manage, or negotiate its somewhat higher costs
  - – ... then AWS is likely a good option for your company.
- Each of those reasons above might point to situations where other services are preferable. In practice, many, if not most, tech startups as well as a number of modern large companies can or already do benefit from using AWS. Many large enterprises are partly migrating internal infrastructure to Azure, Google Cloud, and AWS.
- **Costs:** Billing and cost management are such big topics that we have an entire section on this.
- **EC2 vs. other services:** Most users of AWS are most familiar with EC2, AWS' flagship virtual server product, and possibly a few others like S3 and CLBs. But AWS products now extend far beyond basic IaaS, and often companies do not properly understand or appreciate all the many AWS services and how they can be applied, due to the sharply growing number of services, their novelty and complexity, branding confusion, and fear of lock-in to proprietary AWS technology. Although a bit daunting, it's important for technical decision-makers in companies to understand the breadth of the AWS services and make informed decisions. (We hope this guide will help.)
- **AWS vs. other cloud providers:** While AWS is the dominant IaaS provider (31% market share in this 2016 estimate), there is significant competition and alternatives that are better suited to some companies. This Gartner report has a good overview of the major cloud players :
  - – **Google Cloud**. It arrived later to market than AWS, but has vast resources and is now used widely by many companies, including a few large ones. It is gaining market share. Not all AWS services have similar or analogous services in Google Cloud. And vice versa: In particular Google offers some more advanced machine learning-based services like the Vision, Speech, and Natural Language APIs. It's not common to switch once you're up and running, but it does happen: Spotify migrated from AWS to Google Cloud. There is more discussion on Quora about relative benefits.
  - – **Microsoft Azure** is the de facto choice for companies and teams that are focused on a Microsoft stack, and it has now placed significant emphasis on Linux as well
  - – In **China**, AWS' footprint is relatively small. The market is dominated by Alibaba's Aliyun.
  - – Companies at (very) large scale may want to reduce costs by managing their own infrastructure. For example, Dropbox migrated to their own infrastructure.

- Other cloud providers such as Digital Ocean offer similar services, sometimes with greater ease of use, more personalized support, or lower cost. However, none of these match the breadth of products, mind-share, and market domination AWS now enjoys.
- Traditional managed hosting providers such as Rackspace offer cloud solutions as well.
- **AWS vs. PaaS:** If your goal is just to put up a single service that does something relatively simple, and you're trying to minimize time managing operations engineering, consider a platform-as-a-service such as Heroku. The AWS approach to PaaS, Elastic Beanstalk, is arguably more complex, especially for simple use cases.
- **AWS vs. web hosting:** If your main goal is to host a website or blog, and you don't expect to be building an app or more complex service, you may wish consider one of the myriad of web hosting services.
- **AWS vs. managed hosting:** Traditionally, many companies pay managed hosting providers to maintain physical servers for them, then build and deploy their software on top of the rented hardware. This makes sense for businesses who want direct control over hardware, due to legacy, performance, or special compliance constraints, but is usually considered old fashioned or unnecessary by many developer-centric startups and younger tech companies.
- **Complexity:** AWS will let you build and scale systems to the size of the largest companies, but the complexity of the services when used at scale requires significant depth of knowledge and experience. Even very simple use cases often require more knowledge to do "right" in AWS than in a simpler environment like Heroku or Digital Ocean. (This guide may help!)
- **Geographic locations:** AWS has data centers in over a dozen geographic locations, known as **regions**, in Europe, East Asia, North and South America, and now Australia and India. It also has many more **edge locations** globally for reduced latency of services like CloudFront.
  - See the current list of regions and edge locations, including upcoming ones.
  - If your infrastructure needs to be in close physical proximity to another service for latency or throughput reasons (for example, latency to an ad exchange), viability of AWS may depend on the location.
- **Lock-in:** As you use AWS, it's important to be aware when you are depending on AWS services that do not have equivalents elsewhere.
  - Lock-in may be completely fine for your company, or a significant risk. It's important from a business perspective to make this choice explicitly, and consider the cost, operational, business continuity, and competitive risks of being tied to AWS. AWS is such a dominant and reliable vendor, many companies are comfortable with using AWS to its full extent. Others can tell stories about the dangers of "cloud jail" when costs spiral.
  - Generally, the more AWS services you use, the more lock-in you have to AWS — that is, the more engineering resources (time and money)

it will take to change to other providers in the future.

- Basic services like virtual servers and standard databases are usually easy to migrate to other providers or on premises. Others like load balancers and IAM are specific to AWS but have close equivalents from other providers. The key thing to consider is whether engineers are architecting systems around specific AWS services that are not open source or relatively interchangeable. For example, Lambda, API Gateway, Kinesis, Redshift, and DynamoDB do not have substantially equivalent open source or commercial service equivalents, while EC2, RDS (MySQL or Postgres), EMR, and ElastiCache more or less do. (See more below, where these are noted with .)

- **Combining AWS and other cloud providers:** Many customers combine AWS with other non-AWS services. For example, legacy systems or secure data might be in a managed hosting provider, while other systems are AWS. Or a company might only use S3 with another provider doing everything else. However small startups or projects starting fresh will typically stick to AWS or Google Cloud only.

- **Hybrid cloud:** In larger enterprises, it is common to have hybrid deployments encompassing private cloud or on-premises servers and AWS — or other enterprise cloud providers like IBM/Bluemix, Microsoft/Azure, NetApp, or EMC.

- **Major customers:** Who uses AWS and Google Cloud?
  - AWS's list of customers includes large numbers of mainstream online properties and major brands, such as Netflix, Pinterest, Spotify (moving to Google Cloud), Airbnb, Expedia, Yelp, Zynga, Comcast, Nokia, and Bristol-Myers Squibb.
    * Azure's [list of customers](https://azure.microsoft.com/en-us/case-studies/)
  - Google Cloud's list of customers is large as well, and includes a few mainstream sites, such as Snapchat, Best Buy, Domino's, and Sony Music.

Which Services to Use

- AWS offers a *lot* of different services — about fifty at last count.
- Most customers use a few services heavily, a few services lightly, and the rest not at all. What services you'll use depends on your use cases. Choices differ substantially from company to company.
- **Immature and unpopular services:** Just because AWS has a service that sounds promising, it doesn't mean you should use it. Some services are very narrow in use case, not mature, are overly opinionated, or have limitations, so building your own solution may be better. We try to give a sense for this by breaking products into categories.
- **Must-know infrastructure:** Most typical small to medium-size users will focus on the following services first. If you manage use of AWS systems, you likely need to know at least a little about all of these. (Even if you don't use them, you should learn enough to make that choice intelligently.)

- IAM: User accounts and identities (you need to think about accounts early on!)
- EC2: Virtual servers and associated components, including:
  * AMIs: Machine Images
  * Load Balancers: CLBs and ALBs
  * Autoscaling: Capacity scaling (adding and removing servers based on load)
  * EBS: Network-attached disks
  * Elastic IPs: Assigned IP addresses
- S3: Storage of files
- Route 53: DNS and domain registration
- VPC: Virtual networking, network security, and co-location; you automatically use
- CloudFront: CDN for hosting content
- CloudWatch: Alerts, paging, monitoring

- **Managed services:** Existing software solutions you could run on your own, but with managed deployment:
  - RDS: Managed relational databases (managed MySQL, Postgres, and Amazon's own Aurora database)
  - EMR: Managed Hadoop
  - Elasticsearch: Managed Elasticsearch
  - ElastiCache: Managed Redis and Memcached

- **Optional but important infrastructure:** These are key and useful infrastructure components that are less widely known and used. You may have legitimate reasons to prefer alternatives, so evaluate with care to be sure they fit your needs:
  - Lambda: Running small, fully managed tasks "serverless"
  - CloudTrail: AWS API logging and audit (often neglected but important)
  - CloudFormation: Templatized configuration of collections of AWS resources
  - Elastic Beanstalk: Fully managed (PaaS) deployment of packaged Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker applications
  - EFS: Network filesystem compatible with NFSv4.1
  - ECS: Docker container/cluster management (note Docker can also be used directly, without ECS)
  - ECR: Hosted private Docker registry
  - Config: AWS configuration inventory, history, change notifications

- **Special-purpose infrastructure:** These services are focused on specific use cases and should be evaluated if they apply to your situation. Many also are proprietary architectures, so tend to tie you to AWS.
  - DynamoDB: Low-latency NoSQL key-value store
  - Glacier: Slow and cheap alternative to S3
  - Kinesis: Streaming (distributed log) service
  - SQS: Message queueing service

- Redshift: Data warehouse
- QuickSight: Business intelligence service
- SES: Send and receive e-mail for marketing or transactions
- API Gateway: Proxy, manage, and secure API calls
- IoT: Manage bidirectional communication over HTTP, WebSockets, and MQTT between AWS and clients (often but not necessarily "things" like appliances or sensors)
- WAF: Web firewall for CloudFront to deflect attacks
- KMS: Store and manage encryption keys securely
- Inspector: Security audit
- Trusted Advisor: Automated tips on reducing cost or making improvements
- Certificate Manager: Manage SSL/TLS certificates for AWS services
- **Compound services:** These are similarly specific, but are full-blown services that tackle complex problems and may tie you in. Usefulness depends on your requirements. If you have large or significant need, you may have these already managed by in-house systems and engineering teams.
  - Machine Learning: Machine learning model training and classification
  - Data Pipeline: Managed ETL service
  - SWF: Managed state tracker for distributed polyglot job workflow
  - Lumberyard: 3D game engine
- **Mobile/app development:**
  - SNS: Manage app push notifications and other end-user notifications
  - Cognito: User authentication via Facebook, Twitter, etc.
  - Device Farm: Cloud-based device testing
  - Mobile Analytics: Analytics solution for app usage
  - Mobile Hub: Comprehensive, managed mobile app framework
- **Enterprise services:** These are relevant if you have significant corporate cloud-based or hybrid needs. Many smaller companies and startups use other solutions, like Google Apps or Box. Larger companies may also have their own non-AWS IT solutions.
  - AppStream: Windows apps in the cloud, with access from many devices
  - Workspaces: Windows desktop in the cloud, with access from many devices
  - WorkDocs (formerly Zocalo): Enterprise document sharing
  - WorkMail: Enterprise managed e-mail and calendaring service
  - Directory Service: Microsoft Active Directory in the cloud
  - Direct Connect: Dedicated network connection between office or data center and AWS
  - Storage Gateway: Bridge between on-premises IT and cloud storage
  - Service Catalog: IT service approval and compliance
- **Probably-don't-need-to-know services:** Bottom line, our informal polling indicates these services are just not broadly used — and often for

good reasons:
- – Snowball: If you want to ship petabytes of data into or out of Amazon using a physical appliance, read on.
- – CodeCommit: Git service. You're probably already using GitHub or your own solution (Stackshare has informal stats).
- – CodePipeline: Continuous integration. You likely have another solution already.
- – CodeDeploy: Deployment of code to EC2 servers. Again, you likely have another solution.
- – OpsWorks: Management of your deployments using Chef. While Chef is popular, it seems few people use OpsWorks, since it involves going in on a whole different code deployment framework.
- AWS in Plain English offers more friendly explanation of what all the other different services are.

Tools and Services Market Landscape

There are now enough cloud and "big data" enterprise companies and products that few can keep up with the market landscape. (See the Big Data Evolving Landscape – 2016 for one attempt at this.)

We've assembled a landscape of a few of the services. This is far from complete, but tries to emphasize services that are popular with AWS practitioners — services that specifically help with AWS, or a complementary, or tools almost anyone using AWS must learn.

Popular Tools and Services for AWS Practitioners

Suggestions to improve this figure? Please file an issue.

Common Concepts

- The AWS **General Reference** covers a bunch of common concepts that are relevant for multiple services.
- AWS allows deployments in **regions**, which are isolated geographic locations that help you reduce latency or offer additional redundancy (though typically availability zones are the first tool of choice for high availability).
- Each service has API **endpoints** for each region. Endpoints differ from service to service and not all services are available in each region, as listed in these tables.
- **Amazon Resource Names (ARNs)** are specially formatted identifiers for identifying resources. They start with 'arn:' and are used in many services, and in particular for IAM policies.

Service Matrix

Many services within AWS can at least be compared with Google Cloud offerings or with internal Google services. And often times you could assemble the same

thing yourself with open source software. This table is an effort at listing these rough correspondences. (Remember that this table is imperfect as in almost every case there are subtle differences of features!)

Service	AWS	Google Cloud	Google Internal	Microsoft Azure	Other providers	Open source "build your own"
Virtual server	EC2	Compute Engine (GCE)		Virtual Machine	DigitalOcean	OpenStack
PaaS	Elastic Beanstalk	App Engine	App Engine	Web Apps	Heroku, App-Fog, OpenShift	Meteor, AppScale, Cloud Foundry, Convox
Serverless, microservices	Lambda, API Gateway	Functions		Function Apps	PubNub Blocks, Auth0 Webtask	Kong, Tyk
Container cluster manager	ECS	Container Engine, Kubernetes	Borg or Omega	Container Service		Kubernetes, Mesos, Aurora
File storage	S3	Cloud Storage	GFS	Storage Account		Swift, HDFS
Block storage	EBS	Persistent Disk		Storage Account		NFS
SQL datastore	RDS	Cloud SQL		SQL Database		MySQL, PostgreSQL
Sharded RDBMS			F1, Spanner			Crate.io, CockroachDB
Bigtable		Cloud Bigtable	Bigtable			HBase
Key-value store, column store	DynamoDB	Cloud Datastore	Megastore	Tables, DocumentDB		Cassandra, CouchDB, RethinkDB, Redis
Memory cache	ElastiCache	App Engine Memcache		Redis Cache		Memcached, Redis

Service	AWS	Google Cloud	Google Internal	Microsoft Azure	Other providers	Open source "build your own"
Search	CloudSearch, Elasticsearch (managed)			Search	Algolia, QBox	Elasticsearch, Solr
Data warehouse	Redshift	BigQuery	Dremel	SQL Data Warehouse	Oracle, IBM, SAP, HP, many others	Greenplum
Business intelligence	QuickSight	Data Studio 360		Power BI	Tableau	
Lock manager	DynamoDB (weak)		Chubby	Lease blobs in Storage Account		ZooKeeper, Etcd, Consul
Message broker	SQS, SNS, IoT	Pub/Sub	PubSub2	Service Bus		RabbitMQ, Kafka, 0MQ
Streaming distributed log	Kinesis	Dataflow	PubSub2	Event Hubs		Kafka Streams, Apex, Flink, Spark Streaming, Storm
MapReduce	EMR	Dataproc	MapReduce	HDInsight, Data-Lake Analytics	Qubole	Hadoop
Monitoring Metric management	CloudWatch	Monitoring	Borgmon Borgmon TSDB	Monitor Application Insights		Prometheus(?) Graphite, InfluxDB, OpenTSDB, Grafana, Riemann, Prometheus
CDN	CloudFront	Cloud CDN		CDN		Apache Traffic Server

Service	AWS	Google Cloud	Google Internal	Microsoft Azure	Other providers	Open source "build your own"
Load balancer	CLB/ALB	Load Balancing	GFE	Load Balancer, Application Gateway		nginx, HAProxy, Apache Traffic Server
DNS	Route53	DNS		DNS		bind
Email	SES				Sendgrid, Mandrill, Postmark	
Git hosting	CodeCommit	Cloud Source Repositories		Visual Studio Team Services	GitHub, BitBucket	GitLab
User authentication	Cognito			Azure Active Directory		oauth.io
Mobile app analytics	Mobile Analytics	Firebase Analytics		HockeyApp	Mixpanel	
Mobile app testing	Device Farm	Firebase Test Lab		Xamarin Test Cloud	BrowserStack, Sauce Labs, Testdroid	
Managing SSL/TLS certificates	Certificate Manager				Let's Encrypt, Comodo, Symantec, GlobalSign	

Please help fill this table in.

Selected resources with more detail on this chart:

- Google internal: MapReduce, Bigtable, Spanner, F1 vs Spanner, Bigtable vs Megastore

AWS Product Maturity and Releases

It's important to know the maturity of each AWS product. Here is a mostly complete list of first release date, with links to the release notes. Most recently released services are first. Not all services are available in all regions; see this table.

Service	Original release	Availability	CLI Support
Database Migration Service	2016-03	General	
Certificate Manager	2016-01	General	
IoT	2015-08	General	
WAF	2015-10	General	
Data Pipeline	2015-10	General	
Elasticsearch	2015-10	General	
Service Catalog	2015-07	General	
Device Farm	2015-07	General	
CodePipeline	2015-07	General	
CodeCommit	2015-07	General	
API Gateway	2015-07	General	
Config	2015-06	General	
EFS	2015-05	General	
Machine Learning	2015-04	General	
Lambda	2014-11	General	
ECS	2014-11	General	
KMS	2014-11	General	
CodeDeploy	2014-11	General	
Kinesis	2013-12	General	
CloudTrail	2013-11	General	
AppStream	2013-11	Preview	
CloudHSM	2013-03	General	
Silk	2013-03	Obsolete?	
OpsWorks	2013-02	General	
Redshift	2013-02	General	
Elastic Transcoder	2013-01	General	
Glacier	2012-08	General	
CloudSearch	2012-04	General	
SWF	2012-02	General	
Storage Gateway	2012-01	General	
DynamoDB	2012-01	General	
DirectConnect	2011-08	General	
ElastiCache	2011-08	General	
CloudFormation	2011-04	General	
SES	2011-01	General	
Elastic Beanstalk	2010-12	General	
Route 53	2010-10	General	
IAM	2010-09	General	

Service	Original release	Availability	CLI Support
SNS	2010-04	General	
EMR	2010-04	General	
RDS	2009-12	General	
VPC	2009-08	General	
Snowball	2009-05	General	
CloudWatch	2009-05	General	
CloudFront	2008-11	General	
Fulfillment Web Service	2008-03	Obsolete?	
SimpleDB	2007-12	Nearly obsolete	
DevPay	2007-12	General	
Flexible Payments Service	2007-08	Retired	
EC2	2006-08	General	
SQS	2006-07	General	
S3	2006-03	General	
Alexa Top Sites	2006-01	General	HTTP-only
Alexa Web Information Service	2005-10	General	HTTP-only

Compliance

- Many applications have strict requirements around reliability, security, or data privacy. The AWS Compliance page has details about AWS's certifications, which include **PCI DSS Level 1**, **SOC 3**, and **ISO 9001**.
- Security in the cloud is a complex topic, based on a shared responsibility model, where some elements of compliance are provided by AWS, and some are provided by your company.
- Several third-party vendors offer assistance with compliance, security, and auditing on AWS. If you have substantial needs in these areas, assistance is a good idea.
- From inside **China**, AWS services outside China are generally accessible, though there are at times breakages in service. There are also AWS services inside China.

Getting Help and Support

- **Forums:** For many problems, it's worth searching or asking for help in the discussion forums to see if it's a known issue.
- **Premium support:** AWS offers several levels of premium support.
    - The first tier, called "Developer support" lets you file support tickets with 12 to 24 hour turnaround time, it starts at $29 but once your monthly spend reaches around $1000 it changes to a 3% surcharge on your bill.
    - The higher-level support services are quite expensive — and increase your bill by up to 10%. Many large and effective companies never pay for this level of support. They are usually more helpful for midsize

or larger companies needing rapid turnaround on deeper or more perplexing problems.

– Keep in mind, a flexible architecture can reduce need for support. You shouldn't be relying on AWS to solve your problems often. For example, if you can easily re-provision a new server, it may not be urgent to solve a rare kernel-level issue unique to one EC2 instance. If your EBS volumes have recent snapshots, you may be able to restore a volume before support can rectify the issue with the old volume. If your services have an issue in one availability zone, you should in any case be able to rely on a redundant zone or migrate services to another zone.

– Larger customers also get access to AWS Enterprise support, with dedicated technical account managers (TAMs) and shorter response time SLAs.

– There is definitely some controversy about how useful the paid support is. The support staff don't always seem to have the information and authority to solve the problems that are brought to their attention. Often your ability to have a problem solved may depend on your relationship with your account rep.

- **Account manager:** If you are at significant levels of spend (thousands of US dollars plus per month), you may be assigned (or may wish to ask for) a dedicated account manager.

  – These are a great resource, even if you're not paying for premium support. Build a good relationship with them and make use of them, for questions, problems, and guidance.

  – Assign a single point of contact on your company's side, to avoid confusing or overwhelming them.

- **Contact:** The main web contact point for AWS is here. Many technical requests can be made via these channels.

- **Consulting and managed services:** For more hands-on assistance, AWS has established relationships with many consulting partners and managed service partners. The big consultants won't be cheap, but depending on your needs, may save you costs long term by helping you set up your architecture more effectively, or offering specific expertise, e.g. security. Managed service providers provide longer-term full-service management of cloud resources.

- **AWS Professional Services:** AWS provides consulting services alone or in combination with partners.

Restrictions and Other Notes

- Lots of resources in Amazon have **limits** on them. This is actually helpful, so you don't incur large costs accidentally. You have to request that quotas be increased by opening support tickets. Some limits are easy to raise, and some are not. (Some of these are noted in sections below.)

- **AWS terms of service** are extensive. Much is expected boilerplate, but

it does contain important notes and restrictions on each service. In particular, there are restrictions against using many AWS services in **safety-critical systems**. (Those appreciative of legal humor may wish to review clause 57.10.)

Related Topics

- OpenStack is a private cloud alternative to AWS used by large companies that wish to avoid public cloud offerings.

Learning and Career Development

Certifications

- **Certifications:** AWS offers **certifications** for IT professionals who want to demonstrate their knowledge.
- Certified Solutions Architect Associate
- Certified Developer Associate
- Certified SysOps Administrator Associate
- Certified Solutions Architect Professional
- Certified DevOps Engineer Professional
- **Getting certified:** If you're interested in studying for and getting certifications, this practical overview tells you a lot of what you need to know. The official page is here and there is an FAQ.
- **Do you need a certification?** Especially in consulting companies or when working in key tech roles in large non-tech companies, certifications are important credentials. In others, including in many tech companies and startups, certifications are not common or considered necessary. (In fact, fairly or not, some Silicon Valley hiring managers and engineers see them as a "negative" signal on a resume.)

Managing AWS

Managing Infrastructure State and Change

A great challenge in using AWS to build complex systems (and with DevOps in general) is to manage infrastructure state effectively over time. In general, this boils down to three broad goals for the state of your infrastructure:

- **Visibility**: Do you know the state of your infrastructure (what services you are using, and exactly how)? Do you also know when you — and anyone on your team — make changes? Can you detect misconfigurations, problems, and incidents with your service?
- **Automation**: Can you reconfigure your infrastructure to reproduce past configurations or scale up existing ones without a lot of extra manual work, or requiring knowledge that's only in someone's head? Can you respond to incidents easily or automatically?

- **Flexibility**: Can you improve your configurations and scale up in new ways without significant effort? Can you add more complexity using the same tools? Do you share, review, and improve your configurations within your team?

Much of what we discuss below is really about how to improve the answers to these questions.

There are several approaches to deploying infrastructure with AWS, from the console to complex automation tools, to third-party services, all of which attempt to help achieve visibility, automation, and flexibility.

AWS Configuration Management

The first way most people experiment with AWS is via its web interface, the AWS Console. But using the Console is a highly manual process, and often works against automation or flexibility.

So if you're not going to manage your AWS configurations manually, what should you do? Sadly, there are no simple, universal answers — each approach has pros and cons, and the approaches taken by different companies vary widely, and include directly using APIs (and building tooling on top yourself), using command-line tools, and using third-party tools and services.

AWS Console

- The AWS Console lets you control much (but not all) functionality of AWS via a web interface.
- Ideally, you should only use the AWS Console in a few specific situations:
    - It's great for read-only usage. If you're trying to understand the state of your system, logging in and browsing it is very helpful.
    - It is also reasonably workable for very small systems and teams (for example, one engineer setting up one server that doesn't change often).
    - It can be useful for operations you're only going to do rarely, like less than once a month (for example, a one-time VPC setup you probably won't revisit for a year). In this case using the console can be the simplest approach.
- **Think before you use the console:** The AWS Console is convenient, but also the enemy of automation, reproducibility, and team communication. If you're likely to be making the same change multiple times, avoid the console. Favor some sort of automation, or at least have a path toward automation, as discussed next. Not only does using the console preclude automation, which wastes time later, but it prevents documentation, clarity, and standardization around processes for yourself and your team.

Command-Line tools

- The **aws command-line interface** (CLI), used via the **aws** command, is the most basic way to save and automate AWS operations.
- Don't underestimate its power. It also has the advantage of being well-maintained — it covers a large proportion of all AWS services, and is up to date.
- In general, whenever you can, prefer the command line to the AWS Console for performing operations.
- Even in absence of fancier tools, you can **write simple Bash scripts** that invoke *aws* with specific arguments, and check these into Git. This is a primitive but effective way to document operations you've performed. It improves automation, allows code review and sharing on a team, and gives others a starting point for future work.
- For use that is primarily interactive, and not scripted, consider instead using the **aws-shell** tool from AWS. It is easier to use, with auto-completion and a colorful UI, but still works on the command line. If you're using SAWS, a previous version of the program, you should migrate to aws-shell.

APIs and SDKs

- **SDKs** for using AWS APIs are available in most major languages, with Go, iOS, Java, JavaScript, Python, Ruby, and PHP being most heavily used. AWS maintains a short list, but the awesome-aws list is the most comprehensive and current. Note support for C++ is still new.
- **Retry logic:** An important aspect to consider whenever using SDKs is error handling; under heavy use, a wide variety of failures, from programming errors to throttling to AWS-related outages or failures, can be expected to occur. SDKs typically implement **exponential backoff** to address this, but this may need to be understood and adjusted over time for some applications. For example, it is often helpful to alert on some error codes and not on others.
- Don't use APIs directly. Although AWS documentation includes lots of API details, it's better to use the SDKs for your preferred language to access APIs. SDKs are more mature, robust, and well-maintained than something you'd write yourself.

Boto

- A good way to automate operations in a custom way is **Boto3**, also known as the Amazon SDK for Python. **Boto2**, the previous version of this library, has been in wide use for years, but now there is a newer version with official support from Amazon, so prefer Boto3 for new projects.
- If you find yourself writing a Bash script with more than one or two CLI commands, you're probably doing it wrong. Stop, and consider writing a Boto script instead. This has the advantages that you can:

- Check return codes easily so success of each step depends on success of past steps.
- Grab interesting bits of data from responses, like instance ids or DNS names.
- Add useful environment information (for example, tag your instances with git revisions, or inject the latest build identifier into your initialization script).

General Visibility

- **Tagging resources** is an essential practice, especially as organizations grow, to better understand your resource usage. For example, through automation or convention, you can add tags:
  - For the org or developer that "owns" that resource
  - For the product that resource supports
  - To label lifecycles, such as temporary resources or one that should be deprovisioned in the future
  - To distinguish production-critical infrastructure (e.g. serving systems vs backend pipelines)
  - To distinguish resources with special security or compliance requirements

Managing Servers and Applications

AWS vs Server Configuration

This guide is about AWS, not DevOps or server configuration management in general. But before getting into AWS in detail, it's worth noting that in addition to the configuration management for your AWS resources, there is the long-standing problem of configuration management for servers themselves.

Philosophy

- Heroku's **Twelve-Factor App** principles list some established general best practices for deploying applications.
- **Pets vs cattle:** Treat servers like cattle, not pets. That is, design systems so infrastructure is disposable. It should be minimally worrisome if a server is unexpectedly destroyed.
- The concept of **immutable infrastructure** is an extension of this idea.
- Minimize application state on EC2 instances. In general, instances should be able to be killed or die unexpectedly with minimal impact. State that is in your application should quickly move to RDS, S3, DynamoDB, EFS, or other data stores not on that instance. EBS is also an option, though it generally should not be the bootable volume, and EBS will require manual or automated re-mounting.

Server Configuration Management

- There is a large set of open source tools for managing configuration of server instances.
- These are generally not dependent on any particular cloud infrastructure, and work with any variety of Linux (or in many cases, a variety of operating systems).
- Leading configuration management tools are Puppet, Chef, Ansible, and Saltstack. These aren't the focus of this guide, but we may mention them as they relate to AWS.

Containers and AWS

- Docker and the containerization trend are changing the way many servers and services are deployed in general.
- Containers are designed as a way to package up your application(s) and all of their dependencies in a known way. When you build a container, you are including every library or binary your application needs, outside of the kernel. A big advantage of this approach is that it's easy to test and validate a container locally without worrying about some difference between your computer and the servers you deploy on.
- A consequence of this is that you need fewer AMIs and boot scripts; for most deployments, the only boot script you need is a template that fetches an exported docker image and runs it.
- Companies that are embracing microservice architectures will often turn to container-based deployments.
- AWS launched ECS as a service to manage clusters via Docker in late 2014, though many people still deploy Docker directly themselves. See the ECS section for more details.

Visibility

- Store and track instance metadata (such as instance id, availability zone, etc.) and deployment info (application build id, Git revision, etc.) in your logs or reports. The **instance metadata service** can help collect some of the AWS data you'll need.
- **Use log management services:** Be sure to set up a way to view and manage logs externally from servers.
  - Cloud-based services such as Sumo Logic, Splunk Cloud, Scalyr, and Loggly are the easiest to set up and use (and also the most expensive, which may be a factor depending on how much log data you have).
  - Major open source alternatives include Elasticsearch, Logstash, and Kibana (the "Elastic Stack") and Graylog.
  - If you can afford it (you have little data or lots of money) and don't have special needs, it makes sense to use hosted services whenever possible, since setting up your own scalable log processing systems is notoriously time consuming.

- **Track and graph metrics:** The AWS Console can show you simple graphs from CloudWatch, you typically will want to track and graph many kinds of metrics, from CloudWatch and your applications. Collect and export helpful metrics everywhere you can (and as long as volume is manageable enough you can afford it).
  - Services like Librato, KeenIO, and Datadog have fancier features or better user interfaces that can save a lot of time. (A more detailed comparison is here.)
  - Use Prometheus or Graphite as timeseries databases for your metrics (both are open source).
  - Grafana can visualize with dashboards the stored metrics of both timeseries databases (also open source).

Tips for Managing Servers

- **Timezone settings on servers**: unless *absolutely necessary*, always **set the timezone on servers to UTC** (see instructions for your distribution, such as Ubuntu, CentOS or Amazon Linux). Numerous distributed systems rely on time for synchronization and coordination and UTC provides the universal reference plane: it is not subject to daylight savings changes and adjustments in local time. It will also save you a lot of headache debugging elusive timezone issues and provide coherent timeline of events in your logging and audit systems.
- **NTP and accurate time:** If you are not using Amazon Linux (which comes preconfigured), you should confirm your servers configure NTP correctly, to avoid insidious time drift (which can then cause all sorts of issues, from breaking API calls to misleading logs). This should be part of your automatic configuration for every server. If time has already drifted substantially (generally >1000 seconds), remember NTP won't shift it back, so you may need to remediate manually (for example, like this on Ubuntu).
- **Testing immutable infrastructure:** If you want to be proactive about testing your service's ability to cope with instance termination or failure, it can be helpful to introduce random instance termination during business hours, which will expose any such issues at a time when engineers are available to identify and fix them. Netflix's Simian Army (specifically, Chaos Monkey) is a popular tool for this. Alternatively, chaos-lambda by the BBC is a lightweight option which runs on AWS Lambda.

Security and IAM

We cover security basics first, since configuring user accounts is something you usually have to do early on when setting up your system.

Security and IAM Basics

- IAM Homepage   User guide   FAQ

- The AWS Security Blog is one of the best sources of news and information on AWS security.
- **IAM** is the service you use to manage accounts and permissioning for AWS.
- Managing security and access control with AWS is critical, so every AWS administrator needs to use and understand IAM, at least at a basic level.

- IAM identities include users (people or services that are using AWS), groups (containers for sets of users and their permissions), and roles (containers for permissions assigned to AWS service instances). Permissions for these identities are governed by policies You can use AWS pre-defined policies or custom policies that you create.
- IAM manages various kinds of authentication, for both users and for software services that may need to authenticate with AWS, including:
    - **Passwords** to log into the console. These are a username and password for real users.
    - **Access keys**, which you may use with command-line tools. These are two strings, one the "id", which is an upper-case alphabetic string of the form 'AXXXXXXXXXXXXXXXXXXX', and the other is the secret, which is a 40-character mixed-case base64-style string. These are often set up for services, not just users.
        * Access keys that start with AKIA are normal keys. Access keys that start with ASIA are session/temporary keys from STS, and will require an additional "SessionToken" parameter to be sent along with the id and secret.
    - **Multi-factor authentication (MFA)**, which is the highly recommended practice of using a keychain fob or smartphone app as a second layer of protection for user authentication.
- IAM allows complex and fine-grained control of permissions, dividing users into groups, assigning permissions to roles, and so on. There is a policy language that can be used to customize security policies in a fine-grained way.
    - The policy language has a complex and error-prone JSON syntax that's quite confusing, so unless you are an expert, it is wise to base yours off trusted examples or AWS' own pre-defined managed policies.
- At the beginning, IAM policy may be very simple, but for large systems, it will grow in complexity, and need to be managed with care.
    - Make sure one person (perhaps with a backup) in your organization is formally assigned ownership of managing IAM policies, make sure every administrator works with that person to have changes reviewed. This goes a long way to avoiding accidental and serious misconfigurations.
- It is best to give each user or service the minimum privileges needed to perform their duties. This is the principle of least privilege, one of the foundations of good security. Organize all IAM users and groups according

23

to levels of access they need.
- IAM has the permission hierarchy of:
  1. Explicit deny: The most restrictive policy wins.
  2. Explicit allow: Access permissions to any resource has to be explicitly given.
  3. Implicit deny: All permissions are implicitly denied by default.
- You can test policy permissions via the AWS IAM policy simulator tool tool. This is particularly useful if you write custom policies.

Security and IAM Tips

- Use IAM to create individual user accounts and **use IAM accounts for all users from the beginning**. This is slightly more work, but not that much.
  – That way, you define different users, and groups with different levels of privilege (if you want, choose from Amazon's default suggestions, of administrator, power user, etc.).
  – This allows credential revocation, which is critical in some situations. If an employee leaves, or a key is compromised, you can revoke credentials with little effort.
  – You can set up Active Directory federation to use organizational accounts in AWS.
- **Enable MFA** on your account.
  – You should always use MFA, and the sooner the better — enabling it when you already have many users is extra work.
  – Unfortunately it can't be enforced in software, so an administrative policy has to be established.
  – Most users can use the Google Authenticator app (on iOS or Android) to support two-factor authentication. For the root account, consider a hardware fob.
- **Turn on CloudTrail:** One of the first things you should do is enable CloudTrail. Even if you are not a security hawk, there is little reason not to do this from the beginning, so you have data on what has been happening in your AWS account should you need that information. You'll likely also want to set up a log management service to search and access these logs.
- **Use IAM roles for EC2:** Rather than assign IAM users to applications like services and then sharing the sensitive credentials, define and assign roles to EC2 instances and have applications retrieve credentials from the instance metadata.
- Assign IAM roles by realm — for example, to development, staging, and production. If you're setting up a role, it should be tied to a specific realm so you have clean separation. This prevents, for example, a development instance from connecting to a production database.
- **Best practices:** AWS' list of best practices is worth reading in full up front.

- **Multiple accounts:** Decide on whether you want to use multiple AWS accounts and research how to organize access across them. Factors to consider:
  - Number of users
  - Importance of isolation
    * Resource Limits
    * Permission granularity
    * Security
    * API Limits
  - Regulatory issues
  - Workload
  - Size of infrastructure
  - Cost of multi-account "overhead": Internal AWS service management tools may need to be custom built or adapted.
  - It can help to use separate AWS accounts for independent parts of your infrastructure if you expect a high rate of AWS API calls, since AWS throttles calls at the AWS account level.
- **Inspector** is an automated security assessment service from AWS that helps identify common security risks. This allows validation that you adhere to certain security practices and may help with compliance.
- **Trusted Advisor** addresses a variety of best practices, but also offers some basic security checks around IAM usage, security group configurations, and MFA.
- **Use KMS for managing keys**: AWS offers KMS for securely managing encryption keys, which is usually a far better option than handling key security yourself. See below.
- **AWS WAF** is a web application firewall to help you protect your applications from common attack patterns.
- **Security auditing:**
  - Security Monkey is an open source tool that is designed to assist with security audits.
  - Scout2 is an open source tool that uses AWS APIs to assess an environment's security posture. Scout2 is stable and actively maintained.
  - **Export and audit security settings:** You can audit security policies simply by exporting settings using AWS APIs, e.g. using a Boto script like SecConfig.py (from this 2013 talk) and then reviewing and monitoring changes manually or automatically.

Security and IAM Gotchas and Limitations

- **Don't share user credentials:** It's remarkably common for first-time AWS users to create one account and one set of credentials (access key or password), and then use them for a while, sharing among engineers and others within a company. This is easy. But *don't do this.* This is an insecure practice for many reasons, but in particular, if you do, you will have reduced ability to revoke credentials on a per-user or per-service

basis (for example, if an employee leaves or a key is compromised), which can lead to serious complications.

- **Instance metadata throttling:** The instance metadata service has rate limiting on API calls. If you deploy IAM roles widely (as you should!) and have lots of services, you may hit global account limits easily.
  - One solution is to have code or scripts cache and reuse the credentials locally for a short period (say 2 minutes). For example, they can be put into the ~/.aws/credentials file but must also be refreshed automatically.
  - But be careful not to cache credentials for too long, as they expire. (Note the other dynamic metadata also changes over time and should not be cached a long time, either.)
- Some IAM operations are slower than other API calls (many seconds), since AWS needs to propagate these globally across regions.
- The uptime of IAM's API has historically been lower than that of the instance metadata API. Be wary of incorporating a dependency on IAM's API into critical paths or subsystems — for example, if you validate a user's IAM group membership when they log into an instance and aren't careful about precaching group membership or maintaining a back door, you might end up locking users out altogether when the API isn't available.
- **Don't check in AWS credentials or secrets to a git repository.** There are bots that scan GitHub looking for credentials. Use scripts or tools, such as git-secrets to prevent anyone on your team from checking in sensitive information to your git repositories.

S3

S3 Basics

- Homepage   Developer guide   FAQ   Pricing
- **S3** (Simple Storage Service) is AWS' standard cloud storage service, offering file (opaque "blob") storage of arbitrary numbers of files of almost any size, from 0 to **5TB**. (Prior to 2011 the maximum size was 5 GB; larger sizes are now well supported via multipart support.)
- Items, or **objects**, are placed into named **buckets** stored with names which are usually called **keys**. The main content is the **value**.
- Objects are created, deleted, or updated. Large objects can be streamed, but you cannot access or modify parts of a value; you need to update the whole object.
- Every object also has **metadata**, which includes arbitrary key-value pairs, and is used in a way similar to HTTP headers. Some metadata is system-defined, some are significant when serving HTTP content from buckets or CloudFront, and you can also define arbitrary metadata for your own use.
- **S3 URIs:** Although often bucket and key names are provided in APIs individually, it's also common practice to write an S3 location in the form 's3://bucket-name/path/to/key' (where the key here is 'path/to/key').

(You'll also see 's3n://' and 's3a://' prefixes in Hadoop systems.)

- **S3 vs Glacier, EBS, and EFS:** AWS offers many storage services, and several besides S3 offer file-type abstractions. Glacier is for cheaper and infrequently accessed archival storage. EBS, unlike S3, allows random access to file contents via a traditional filesystem, but can only be attached to one EC2 instance at a time. EFS is a network filesystem many instances can connect to, but at higher cost. See the comparison table.

S3 Tips

- For most practical purposes, you can consider S3 capacity unlimited, both in total size of files and number of objects. The number of objects in a bucket is essentially also unlimited. Customers routinely have millions of objects.
- **Bucket naming:** Buckets are chosen from a global namespace (across all regions, even though S3 itself stores data in whichever S3 region you select), so you'll find many bucket names are already taken. Creating a bucket means taking ownership of the name until you delete it. Bucket names have a few restrictions on them.
  - Bucket names can be used as part of the hostname when accessing the bucket or its contents, like `<bucket_name>.s3-us-east-1.amazonaws.com`, as long as the name is DNS compliant.
  - A common practice is to use the company name acronym or abbreviation to prefix (or suffix, if you prefer DNS-style hierarchy) all bucket names (but please, don't use a check on this as a security measure — this is highly insecure and easily circumvented!).
  - Bucket names with '.' (periods) in them can cause certificate mismatches when used with SSL. Use '-' instead, since this then conforms with both SSL expectations and is DNS compliant.
- **Versioning:** S3 has optional versioning support, so that all versions of objects are preserved on a bucket. This is mostly useful if you want an archive of changes or the ability to back out mistakes (it has none of the features of full version control systems like Git).
- **Durability:** Durability of S3 is extremely high, since internally it keeps several replicas. If you don't delete it by accident, you can count on S3 not losing your data. (AWS offers the seemingly improbable durability rate of 99.999999999%, but this is a mathematical calculation based on independent failure rates and levels of replication — not a true probability estimate. Either way, S3 has had a very good record of durability.) Note this is *much* higher durability than EBS! If durability is less important for your application, you can use S3 Reduced Redundancy Storage, which lowers the cost per GB, as well as the redundancy.
- **S3 pricing** depends on storage, requests, and transfer.
  - For transfer, putting data into AWS is free, but you'll pay on the way out. Transfer from S3 to EC2 in the *same region* is free. Transfer to other regions or the Internet in general is not free.

27

- Deletes are free.
- **S3 Reduced Redundancy and Infrequent Access:** Most people use the Standard storage class in S3, but there are other storage classes with lower cost:
  - Reduced Redundancy Storage (RRS) has lower durability (99.99%, so just four nines). That is, there's a small chance you'll lose data. For some data sets where data has value in a statistical way (losing say half a percent of your objects isn't a big deal) this is a reasonable trade-off.
  - Infrequent Access (IA) lets you get cheaper storage in exchange for more expensive access. This is great for archives like logs you already processed, but might want to look at later. To get an idea of the cost savings when using Infrequent Access (IA), you can use this S3 Infrequent Access Calculator.
  - Glacier is a third alternative discussed as a separate product.
  - See the comparison table.
- **Performance:** Maximizing S3 performance means improving overall throughput in terms of bandwidth and number of operations per second.
  - S3 is highly scalable, so in principle you can get arbitrarily high throughput. (A good example of this is S3DistCp.)
  - But usually you are constrained by the pipe between the source and S3 and/or the level of concurrency of operations.
  - Throughput is of course highest from within AWS to S3, and between EC2 instances and S3 buckets that are in the same region.
  - Bandwidth from EC2 depends on instance type. See the "Network Performance" column at ec2instances.info.
  - Throughput of many objects is extremely high when data is accessed in a distributed way, from many EC2 instances. It's possible to read or write objects from S3 from hundreds or thousands of instances at once.
  - However, throughput is very limited when objects accessed sequentially from a single instance. Individual operations take many milliseconds, and bandwidth to and from instances is limited.
  - Therefore, to perform large numbers of operations, it's necessary to use multiple worker threads and connections on individual instances, and for larger jobs, multiple EC2 instances as well.
  - **Multi-part uploads:** For large objects you want to take advantage of the multi-part uploading capabilities (starting with minimum chunk sizes of 5 MB).
  - **Large downloads:** Also you can download chunks of a single large object in parallel by exploiting the HTTP GET range-header capability.
  - **List pagination:** Listing contents happens at 1000 responses per request, so for buckets with many millions of objects listings will take time.
  - **Key prefixes:** In addition, latency on operations is highly depen-

dent on prefix similarities among key names. If you have need for high volumes of operations, it is essential to consider naming schemes with more randomness early in the key name (first 6 or 8 characters) in order to avoid "hot spots".

  * We list this as a major gotcha since it's often painful to do large-scale renames.
  * Note that sadly, the advice about random key names goes against having a consistent layout with common prefixes to manage data lifecycles in an automated way.
- For data outside AWS, **DirectConnect** and **S3 Transfer Acceleration** can help. For S3 Transfer Acceleration, you pay about the equivalent of 1-2 months of storage for the transfer in either direction for using nearer endpoints.

- **Command-line applications:** There are a few ways to use S3 from the command line:
  - Originally, **s3cmd** was the best tool for the job. It's still used heavily by many.
  - The regular **aws** command-line interface now supports S3 well, and is useful for most situations.
  - **s4cmd** is a replacement, with greater emphasis on performance via multi-threading, which is helpful for large files and large sets of files, and also offers Unix-like globbing support.

- **GUI applications:** You may prefer a GUI, or wish to support GUI access for less technical users. Some options:
  - The AWS Console does offer a graphical way to use S3. Use caution telling non-technical people to use it, however, since without tight permissions, it offers access to many other AWS features.
  - Transmit is a good option on OS X for basic use cases. Uses legacy AWS2 signatures for authentication and is missing multipart upload support.
  - Cyberduck is a good option on OS X and Windows with support for multipart uploads, ACLs, versioning, lifecycle configuration, storage classes and server side encryption (SSE-S3 and SSE-KMS).

- **S3 and CloudFront:** S3 is tightly integrated with the CloudFront CDN. See the CloudFront section for more information, as well as S3 transfer acceleration.

- **Static website hosting:**
  - S3 has a static website hosting option that is simply a setting that enables configurable HTTP index and error pages and HTTP redirect support to public content in S3. It's a simple way to host static assets or a fully static website.
  - Consider using CloudFront in front of most or all assets:
    * Like any CDN, CloudFront improves performance significantly.
    * SSL is only supported on the built-in amazonaws.com domain for S3. S3 supports serving these sites through a custom domain, but not over SSL on a custom domain. However, CloudFront

29

allows you to serve a custom domain over https. Amazon provides free SNI SSL/TLS certificates via Amazon Certificate Manager. SNI does not work on very outdated browsers/operating systems. Alternatively, you can provide your own certificate to use on CloudFront to support all browsers/operating systems.

* If you are including resources across domains, such as fonts inside CSS files, you may need to configure CORS for the bucket serving those resources.
* Since pretty much everything is moving to SSL nowadays, and you likely want control over the domain, you probably want to set up CloudFront with your own certificate in front of S3 (and to ignore the AWS example on this as it is non-SSL only).
* That said, if you do, you'll need to think through invalidation or updates on CloudFront. You may wish to include versions or hashes in filenames so invalidation is not necessary.

- **Permissions:**
  - It's important to manage permissions sensibly on S3 if you have data sensitivities, as fixing this later can be a difficult task if you have a lot of assets and internal users.
  - Do create new buckets if you have different data sensitivities, as this is much less error prone than complex permissions rules.
  - If data is for administrators only, like log data, put it in a bucket that only administrators can access.
  - Limit individual user (or IAM role) access to S3 to the minimal required and catalog the "approved" locations. Otherwise, S3 tends to become the dumping ground where people put data to random locations that are not cleaned up for years, costing you big bucks.

- **Data lifecycles:**
  - When managing data, the understanding the lifecycle of the data is as important as understanding the data itself. When putting data into a bucket, think about its lifecycle — its end of life, not just its beginning.
  - In general, data with different expiration policies should be stored under separate prefixes at the top level. For example, some voluminous logs might need to be deleted automatically monthly, while other data is critical and should never be deleted. Having the former in a separate bucket or at least a separate folder is wise.
  - Thinking about this up front will save you pain. It's very hard to clean up large collections of files created by many engineers with varying lifecycles and no coherent organization.
  - Alternatively you can set a lifecycle policy to archive old data to Glacier. Be careful with archiving large numbers of small objects to Glacier, since it may actually cost more.
  - There is also a storage class called **Infrequent Access** that has the same durability as Standard S3, but is discounted per GB. It is suitable for objects that are infrequently accessed.

- **Data consistency:** Understanding data consistency is critical for any use of S3 where there are multiple producers and consumers of data.
  - Creation of individual objects in S3 is atomic. You'll never upload a file and have another client see only half the file.
  - Also, if you create a new object, you'll be able to read it instantly, which is called **read-after-write consistency**.
    * Well, with the additional caveat that if you do a read on an object before it exists, then create it, you get eventual consistency (not read-after-write).
  - If you overwrite or delete an object, you're only guaranteed eventual consistency.
  - Note that until 2015, 'us-standard' region had had a weaker eventual consistency model, and the other (newer) regions were read-after-write. This was finally corrected — but watch for many old blogs mentioning this!
  - In practice, "eventual consistency" usually means within seconds, but expect rare cases of minutes or hours.
- **S3 as a filesystem:**
  - In general S3's APIs have inherent limitations that make S3 hard to use directly as a POSIX-style filesystem while still preserving S3's own object format. For example, appending to a file requires rewriting, which cripples performance, and atomic rename of directories, mutual exclusion on opening files, and hardlinks are impossible.
  - s3fs is a FUSE filesystem that goes ahead and tries anyway, but it has performance limitations and surprises for these reasons.
  - Riofs (C) and Goofys (Go) are more recent efforts that attempt adopt a different data storage format to address those issues, and so are likely improvements on s3fs.
  - S3QL (discussion) is a Python implementation that offers data de-duplication, snap-shotting, and encryption, but only one client at a time.
  - ObjectiveFS (discussion) is a commercial solution that supports filesystem features and concurrent clients.
- If you are primarily using a VPC, consider setting up a VPC Endpoint for S3 in order to allow your VPC-hosted resources to easily access it without the need for extra network configuration or hops.
- **Cross-region replication:** S3 has a feature for replicating a bucket between one region and another. Note that S3 is already highly replicated within one region, so usually this isn't necessary for durability, but it could be useful for compliance (geographically distributed data storage), lower latency, or as a strategy to reduce region-to-region bandwidth costs by mirroring heavily used data in a second region.
- **IPv4 vs IPv6:** For a long time S3 only supported IPv4 at the default endpoint `https://BUCKET.s3.amazonaws.com`. However, as of Aug 11, 2016 it now supports both IPv4 & IPv6! To use both, you have to enable dualstack either in your preferred API client or by directly using this url

scheme `https://BUCKET.s3.dualstack.REGION.amazonaws.com`.
- **S3 event notifications:** S3 can be configured to send an SNS notification, SQS message, or AWS Lambda function on bucket events.

S3 Gotchas and Limitations

- For many years, there was a notorious **100-bucket limit** per account, which could not be raised and caused many companies significant pain. As of 2015, you can request increases. You can ask to increase the limit, but it will still be capped (generally below ~1000 per account).
- Be careful not to make implicit assumptions about transactionality or sequencing of updates to objects. Never assume that if you modify a sequence of objects, the clients will see the same modifications in the same sequence, or if you upload a whole bunch of files, that they will all appear at once to all clients.
- S3 has an **SLA** with 99.9% uptime. If you use S3 heavily, you'll inevitably see occasional error accessing or storing data as disks or other infrastructure fail. Availability is usually restored in seconds or minutes. Although availability is not extremely high, as mentioned above, durability is excellent.
- After uploading, any change that you make to the object causes a full rewrite of the object, so avoid appending-like behavior with regular files.
- Eventual data consistency, as discussed above, can be surprising sometimes. If S3 suffers from internal replication issues, an object may be visible from a subset of the machines, depending on which S3 endpoint they hit. Those usually resolve within seconds; however, we've seen isolated cases when the issue lingered for 20-30 hours.
- **MD5s and multi-part uploads:** In S3, the ETag header in S3 is a hash on the object. And in many cases, it is the MD5 hash. However, this is not the case in general when you use multi-part uploads. One workaround is to compute MD5s yourself and put them in a custom header (such as is done by s4cmd).
- **Incomplete multi-part upload costs:** Incomplete multi-part uploads accrue storage charges even if the upload fails and no S3 object is created. Amazon (and others) recommend using a lifecycle policy to clean up incomplete uploads and save on storage costs.
- **US Standard region:** Previously, the us-east-1 region (also known as the US Standard region) was replicated across coasts, which led to greater variability of latency. Effective Jun 19, 2015 this is no longer the case. All Amazon S3 regions now support read-after-write consistency. Amazon S3 also renamed the US Standard region to the US East (N. Virginia) region to be consistent with AWS regional naming conventions.
- When configuring ACLs on who can access the bucket and contents, a predefined group exists called **Authenticated Users**. This group is often used, incorrectly, to restrict S3 resource access to authenticated users of the owning account. If granted, the AuthenticatedUsers group will allow

S3 resource access to **all authenticated users, across all AWS accounts**. A typical use case of this ACL is used in conjunction with the requester pays functionality of S3.

- **S3 authentication versions and regions:** In newer regions, S3 only supports the latest authentication. If an S3 file operation using CLI or SDK doesn't work in one region, but works correctly in another region, make sure you are using the latest authentication signature.

Storage Durability, Availability, and Price

As an illustration of comparative features and price, the table below gives S3 Standard, RRS, IA, in comparison with Glacier, EBS, and EFS, using **Virginia region** as of **August 2016**.

	Durability (per year)	Availability "designed"	Availability SLA	Storage (per TB per month)	GET or retrieve (per million)	Write or archive (per million)
Glacier	Eleven 9s	Slooow	–	$7	$50	$50
S3 IA	Eleven 9s	99.9%	**99%**	$12.50	$1	$10
S3 RRS	**99.99%**	99.99%	99.9%	$24	$0.40	$5
S3 Standard	Eleven 9s	99.99%	99.9%	$30	$0.40	$5
EBS	**99.8%**	Unstated	99.95%	$25/$45/**$100**/$125+ (sc1/st1/**gp2**/io1)		
EFS	"High"	"High"	–	$300		

Especially notable items are in **boldface**. Sources: S3 pricing, S3 SLA, S3 FAQ, RRS info, Glacier pricing, EBS availability and durability, EBS pricing, EFS pricing, EC2 SLA

EC2

EC2 Basics

- Homepage Documentation FAQ Pricing (see also ec2instances.info)
- **EC2** (Elastic Compute Cloud) is AWS' offering of the most fundamental piece of cloud computing: A virtual private server. These "instances" can run most Linux, BSD, and Windows operating systems. Internally, they use Xen virtualization.

- The term "EC2" is sometimes used to refer to the servers themselves, but technically refers more broadly to a whole collection of supporting services, too, like load balancing (CLBs/ALBs), IP addresses (EIPs), bootable images (AMIs), security groups, and network drives (EBS) (which we discuss individually in this guide).
- **EC2 pricing** and **cost management** is a complicated topic. It can range from free (on the AWS free tier) to a lot, depending on your usage. Pricing is by instance type, by hour and changes depending on AWS region and whether you are purchasing your instances On-Demand, on the Spot market or pre-purchasing (Reserved Instances).
- **Network Performance:** For some instance types, AWS uses general terms like Low, Medium, and High to refer to network performance. Users have done benchmarking to provide expectations for what these terms can mean.

EC2 Alternatives and Lock-In

- Running EC2 is akin to running a set of physical servers, as long as you don't do automatic scaling or tooled cluster setup. If you just run a set of static instances, migrating to another VPS or dedicated server provider should not be too hard.
- **Alternatives to EC2:** The direct alternatives are Google Cloud, Microsoft Azure, Rackspace, DigitalOcean and other VPS providers, some of which offer similar API for setting up and removing instances. (See the comparisons above.)
- **Should you use Amazon Linux?** AWS encourages use of their own Amazon Linux, which is evolved from Red Hat Enterprise Linux (RHEL) and CentOS. It's used by many, but others are skeptical. Whatever you do, think this decision through carefully. It's true Amazon Linux is heavily tested and better supported in the unlikely event you have deeper issues with OS and virtualization on EC2. But in general, many companies do just fine using a standard, non-Amazon Linux distribution, such as Ubuntu or CentOS. Using a standard Linux distribution means you have an exactly replicable environment should you use another hosting provider instead of (or in addition to) AWS. It's also helpful if you wish to test deployments on local developer machines running the same standard Linux distribution (a practice that's getting more common with Docker, too, and not currently possible with Amazon Linux).
- **EC2 costs:** See the section on this.

EC2 Tips

- **Picking regions:** When you first set up, consider which regions you want to use first. Many people in North America just automatically set up in the us-east-1 (N. Virginia) region, which is the default, but it's worth considering if this is best up front. For example, you might find

34

it preferable to start in us-west-1 (N. California) or us-west-2 (Oregon) if you're in California and latency matters. Some services are not available in all regions. Baseline costs also vary by region, up to 10-30% (generally lowest in us-east-1).

- **Instance types:** EC2 instances come in many types, corresponding to the capabilities of the virtual machine in CPU architecture and speed, RAM, disk sizes and types (SSD or magnetic), and network bandwidth.

    - Selecting instance types is complex since there are so many types. Additionally, there are different generations, released over the years.
    - Use the list at **ec2instances.info** to review costs and features. Amazon's own list of instance types is hard to use, and doesn't list features and price together, which makes it doubly difficult.
    - Prices vary a lot, so use **ec2instances.info** to determine the set of machines that meet your needs and **ec2price.com** to find the cheapest type in the region you're working in. Depending on the timing and region, it might be much cheaper to rent an instance with *more* memory or CPU than the bare minimum.

- **Turn off** your instances when they aren't in use. For many situations such as testing or staging resources, you may not need your instances on 24/7, and you won't need to pay EC2 hourly costs when they are suspended. Given that costs are calculated based on hourly usage, this is a simple mechanism for cost savings. This can be achieved using Lambda and CloudWatch, an open source solution like Scalr or a SaaS provider like GorillaStack. (Note: if you turn off instances with an ephemeral root volume, any state will be lost when the instance is turned off. Therefore, for stateful applications it is safer to turn off EBS backed instances).

- **Dedicated instances** and **dedicated hosts** are assigned hardware, instead of usual virtual instances. They are more expensive than virtual instances but can be preferable for performance, compliance, or licensing reasons.

- **32 bit vs 64 bit:** A few micro, small, and medium instances are still available to use as 32-bit architecture. You'll be using 64-bit EC2 ("amd64") instances nowadays, though smaller instances still support 32 bit ("i386"). Use 64 bit unless you have legacy constraints or other good reasons to use 32.

- **HVM vs PV:** There are two kinds of virtualization technology used by EC2, hardware virtual machine (HVM) and paravirtual (PV). Historically, PV was the usual type, but now HVM is becoming the standard. If you want to use the newest instance types, you must use HVM. See the instance type matrix for details.

- **Operating system:** To use EC2, you'll need to pick a base operating system. It can be Windows or Linux, such as Ubuntu or Amazon Linux.

You do this with AMIs, which are covered in more detail in their own section below.

- **Limits:** You can't create arbitrary numbers of instances. Default limits on numbers of EC2 instances per account vary by instance type, as described in this list.

- **Use termination protection:** For any instances that are important and long-lived (in particular, aren't part of auto-scaling), enable termination protection. This is an important line of defense against user mistakes, such as accidentally terminating many instances instead of just one due to human error.

- **SSH key management:**
  - When you start an instance, you need to have at least one ssh key pair set up, to bootstrap, i.e., allow you to ssh in the first time.
  - Aside from bootstrapping, you should manage keys yourself on the instances, assigning individual keys to individual users or services as appropriate.
  - Avoid reusing the original boot keys except by administrators when creating new instances.
  - Avoid sharing keys and add individual ssh keys for individual users.

- **GPU support:** You can rent GPU-enabled instances on EC2 for use in machine learning or graphics rendering workloads.
  - There are three generations of GPU-enabled instances available:
    * Third generation P2 series offers NVIDIA K80 GPUs in 1, 8 and 16 GPU configurations targeting machine learning and scientific workloads.
    * Second generation G2 series offers NVIDIA K520 GPUs in 1 or 4 GPU configurations targeting graphics and video encoding.
    * First generation CG1 instances are still available in some regions in a single configuration with a NVIDIA M2050 GPU.
  - AWS offers an AMI (based on Amazon Linux) with most NVIDIA drivers and ancillary software (CUDA, CUBLAS, CuDNN, TensorFlow) installed to lower the barrier to usage. Note, however, that this leads to lock-in due to Amazon Linux and the fact that you have no direct access to software configuration or versioning.
  - As with any expensive EC2 instance types, Spot instances can offer significant savings with GPU workloads when interruptions are tolerable.

EC2 Gotchas and Limitations

- Never use ssh passwords. Just don't do it; they are too insecure, and consequences of compromise too severe. Use keys instead. Read up on this and fully disable ssh password access to your ssh server by making

36

sure 'PasswordAuthentication no' is in your /etc/ssh/sshd_config file. If you're careful about managing ssh private keys everywhere they are stored, it is a major improvement on security over password-based authentication.

- For all newer instance types, when selecting the AMI to use, be sure you select the HVM AMI, or it just won't work.

- When creating an instance and using a new ssh key pair, make sure the ssh key permissions are correct.

- Sometimes certain EC2 instances can get scheduled for retirement by AWS due to "detected degradation of the underlying hardware," in which case you are given a couple of weeks to migrate to a new instance

  - If your instance root device is an EBS volume, you can typically stop and then start the instance which moves it to healthy host hardware, giving you control over timing of this event. Note however that you will lose any instance store volume data (ephemeral drives) if your instance type has instance store volumes.
  - The instance public IP (if it has one) will likely change unless you're using Elastic IPs. This could be a problem if other systems depend on the IP address.

- Periodically you may find that your server or load balancer is receiving traffic for (presumably) a previous EC2 server that was running at the same IP address that you are handed out now (this may not matter, or it can be fixed by migrating to another new instance).

- If the EC2 API itself is a critical dependency of your infrastructure (e.g. for automated server replacement, custom scaling algorithms, etc.) and you are running at a large scale or making many EC2 API calls, make sure that you understand when they might fail (calls to it are rate limited and the limits are not published and subject to change) and code and test against that possibility.

- Many newer EC2 instance types are EBS-only. Make sure to factor in EBS performance and costs when planning to use them.

- Instances come in two types: **Fixed Performance Instances** (e.g. M3, C3, and R3) and **Burstable Performance Instances** (e.g. T2). A T2 instance receives CPU credits continuously, the rate of which depends on the instance size. T2 instances accrue CPU credits when they are idle, and use CPU credits when they are active. However, once an instance runs out of credits, you'll notice a severe degradation in performance. If you need consistently high CPU performance for applications such as video encoding, high volume websites or HPC applications, it is recommended to use Fixed Performance Instances.

- An IAM role can be assigned to an EC2 instance only at launch time. You cannot assign to a running instance.

- Instance user-data is [limited to 16 KB](http://docs.aws.amazon.com/AWSEC2/latest/Us

- Very new accounts may not be able to launch some instance types, such as GPU instanc

CloudWatch

CloudWatch Basics

- Homepage Documentation FAQ Pricing
- **CloudWatch** monitors resources and applications, captures logs, and
  sends events.
- CloudWatch monitoring is the standard mechanism for keeping tabs on
  AWS resources. A wide range of **metrics and dimensions** are available
  via CloudWatch, allowing you to create time based graphs, **alarms**, and
  dashboards.
    - Alarms are the most practical use of CloudWatch, allowing you to
      trigger notifications from any given metric.
    - Alarms can trigger SNS notifications, Auto Scaling actions, or EC2
      actions.
    - Publish and share graphs of metrics by creating customizable dash-
      board views.
        * Monitor and report on EC2 instance system check failure alarms.
- **Using CloudWatch Events:**
    - Events create a mechanism to automate actions in various services
      on AWS. You can create event rules from instance states, AWS APIs,
      Auto Scaling, Run commands, deployments or time-based schedules
      (think Cron).
    - Triggered events can can invoke Lambda functions, send
      SNS/SQS/Kinesis messages, or perform instance actions (terminate,
      restart, stop, or snapshot volumes).
    - Custom payloads can be sent to targets in JSON format, this is
      especially useful when triggering Lambdas.
- **Using CloudWatch Logs:**
    - CloudWatch Logs is a streaming log storage system. By storing logs
      within AWS you have access to unlimited paid storage, but you also
      have the option of streaming logs directly to ElasticSearch or custom
      Lambdas.
    - A log agent installed on your servers will process logs over time and
      send them to CloudWatch Logs.
    - You can export logged data to S3 or stream results to other AWS
      services.
- **Detailed monitoring:** Detailed monitoring for EC2 instances must be
  enabled to get granular metrics, and is billed under CloudWatch.

CloudWatch Alternatives and Lock-In

- CloudWatch offers fairly basic functionality that doesn't create significant (additional) AWS lock-in. Most of the metrics provided by the service can be obtained through APIs that can be imported into other aggregation or visualization tools or services (many specifically provide CloudWatch data import services).
- Alternatives to CloudWatch monitoring services include NewRelic, Datadog, Sumo Logic, Zabbix, Nagios, Ruxit, Elastic Stack, open source options such as StatsD or collectd with Graphite, and many others.
- CloudWatch Log alternatives include Splunk, Sumo Logic, Loggly, Logstash, Papertrail, Elastic Stack, and other centralized logging solutions.

CloudWatch Tips

- Some very common use cases for CloudWatch are **billing alarms**, **instance or load balancer up/down alarms**, and **disk usage alerts**.
- You can use EC2Config to monitor watch memory and disk metrics on Windows platform instances. For Linux, there are example scripts that do the same thing.
- You can publish your own metrics using the AWS API. Incurs additional cost.
- You can stream directly from CloudWatch Logs to a Lambda or ElasticSearch cluster by creating subscriptions on Log Groups.
- Don't forget to take advantage of the CloudWatch non-expiring free tier.

CloudWatch Gotchas and Limitations

- Metrics in CloudWatch originate on the hypervisor. The hypervisor doesn't have access to OS information, so certain metrics (most notably memory utilization) are not available unless pushed to CloudWatch from inside the instance.
- You can not use more than one metric for an alarm.
- Notifications you receive from alarms will not have any contextual detail; they have only the specifics of the threshold, alarm state, and timing.
- Minimum granularity in CloudWatch is 1 minute. That means that multiple values of a metric that are pushed to CloudWatch within the same minute are aggregated into minimum, maximum, average and total (sum) per minute.
- Data about metrics is kept in CloudWatch for 15 months, starting November 2016 (used to be 14 days). Minimum granularity increases after 15 days.

AMIs

AMI Basics

- User guide
- **AMIs** (Amazon Machine Images) are immutable images that are used to launch preconfigured EC2 instances. They come in both public and private flavors. Access to public AMIs is either freely available (shared/community AMIs) or bought and sold in the **AWS Marketplace**.
- Many operating system vendors publish ready-to-use base AMIs. For Ubuntu, see the Ubuntu AMI Finder. Amazon of course has AMIs for Amazon Linux.

AMI Tips

- AMIs are built independently based on how they will be deployed. You must select AMIs that match your deployment when using them or creating them:
  - EBS or instance store
  - PV or HVM virtualization types
  - 32 bit ("i386") vs 64 bit ("amd64") architecture
- As discussed above, modern deployments will usually be with **64-bit EBS-backed HVM**.
- You can create your own custom AMI by snapshotting the state of an EC2 instance that you have modified.
- AMIs backed by EBS storage have the necessary image data loaded into the EBS volume itself and don't require an extra pull from S3, which results in EBS-backed instances coming up much faster than instance storage-backed ones.
- **AMIs are per region**, so you must look up AMIs in your region, or copy your AMIs between regions with the AMI Copy feature.
- As with other AWS resources, it's wise to use tags to version AMIs and manage their lifecycle.
- If you create your own AMIs, there is always some tension in choosing how much installation and configuration you want to "bake" into them.
  - Baking less into your AMIs (for example, just a configuration management client that downloads, installs, and configures software on new EC2 instances when they are launched) allows you to minimize time spent automating AMI creation and managing the AMI lifecycle (you will likely be able to use fewer AMIs and will probably not need to update them as frequently), but results in longer waits before new instances are ready for use and results in a higher chance of launch-time installation or configuration failures.
  - Baking more into your AMIs (for example, pre-installing but not fully configuring common software along with a configuration management client that loads configuration settings at launch time) results in a faster launch time and fewer opportunities for your software installa-

tion and configuration to break at instance launch time but increases the need for you to create and manage a robust AMI creation pipeline.

– Baking even more into your AMIs (for example, installing all required software as well and potentially also environment-specific configuration information) results in fast launch times and a much lower chance of instance launch-time failures but (without additional re-deployment and re-configuration considerations) can require time consuming AMI updates in order to update software or configuration as well as more complex AMI creation automation processes.

- Which option you favor depends on how quickly you need to scale up capacity, and size and maturity of your team and product.
    – When instances boot fast, auto-scaled services require less spare capacity built in and can more quickly scale up in response to sudden increases in load. When setting up a service with autoscaling, consider baking more into your AMIs and backing them with the EBS storage option.
    – As systems become larger, it common to have more complex AMI management, such as a multi-stage AMI creation process in which few (ideally one) common base AMIs are infrequently regenerated when components that are common to all deployed services are updated and then a more frequently run "service-level" AMI generation process that includes installation and possibly configuration of application-specific software.
- More thinking on AMI creation strategies here.
- Use tools like Packer to simplify and automate AMI creation.

AMI Gotchas and Limitations

- **Amazon Linux package versions:** By default, instances based on Amazon Linux AMIs are configured point to the latest versions of packages in Amazon's package repository. This means that the package versions that get installed are not locked and it is possible for changes, including breaking ones, to appear when applying updates in the future. If you bake your AMIs with updates already applied, this is unlikely to cause problems in running services whose instances are based on those AMIs – breaks will appear at the earlier AMI-baking stage of your build process, and will need to be fixed or worked around before new AMIs can be generated. There is a "lock on launch" feature that allows you to configure Amazon Linux instances to target the repository of a particular major version of the Amazon Linux AMI, reducing the likelihood that breaks caused by Amazon-initiated package version changes will occur at package install time but at the cost of not having updated packages get automatically installed by future update runs. Pairing use of the "lock on launch" feature with a process to advance the Amazon Linux AMI at your discretion can give you tighter control over update behaviors and timings.

Auto Scaling

Auto Scaling Basics

- Homepage User guide FAQ Pricing at no additional charge
- **Auto Scaling Groups (ASGs)** are used to control the number of instances in a service, reducing manual effort to provision or deprovision EC2 instances.
- They can be configured through Scaling Policies to automatically increase or decrease instance counts based on metrics like CPU utilization, or based on a schedule.
- There are three common ways of using ASGs - dynamic (automatically adjust instance count based on metrics for things like CPU utilization), static (maintain a specific instance count at all times), scheduled (maintain different instance counts at different times of day or on days of the week).
- ASGs have no additional charge themselves; you pay for underlying EC2 and CloudWatch services.

Auto Scaling Tips

- Better matching your cluster size to your current resource requirements through use of ASGs can result in significant cost savings for many types of workloads.
- Pairing ASGs with CLBs is a common pattern used to deal with changes in the amount of traffic a service receives.
- Dynamic Auto Scaling is easiest to use with stateless, horizontally scalable services.
- Even if you are not using ASGs to dynamically increase or decrease instance counts, you should seriously consider maintaining all instances inside of ASGs – given a target instance count, the ASG will work to ensure that number of instances running is equal to that target, replacing instances for you if they die or are marked as being unhealthy. This results in consistent capacity and better stability for your service.
- Autoscalers can be configured to terminate instances that a CLB or ALB has marked as being unhealthy.

Auto Scaling Gotchas and Limitations

- **ReplaceUnhealthy setting:** By default, ASGs will kill instances that the EC2 instance manager considers to be unresponsive. It is possible for instances whose CPU is completely saturated for minutes at a time to appear to be unresponsive, causing an ASG with the default Replace-Unhealthy setting turned on to replace them. When instances that are managed by ASGs are expected to consistently run with very high CPU, consider deactivating this setting. If you do so, however, detecting and killing unhealthy nodes will become your responsibility.

EBS

EBS Basics

- Homepage User guide FAQ Pricing
- **EBS** (Elastic Block Store) provides block level storage. That is, it offers storage volumes that can be attached as filesystems, like traditional network drives.
- EBS volumes can only be attached to one EC2 instance at a time. In contrast, EFS can be shared but has a much higher price point (a comparison).

EBS Tips

- **RAID:** Use RAID drives for increased performance.
- A worthy read is AWS' post on EBS IO characteristics as well as their performance tips.
- One can provision IOPS (that is, pay for a specific level of I/O operations per second) to ensure a particular level of performance for a disk.
- A single EBS volume allows 10k IOPS max. To get the maximum performance out of an EBS volume, it has to be of a maximum size and attached to an EBS-optimized EC2 instance.
- A standard block size for an EBS volume is 16kb.

EBS Gotchas and Limitations

- EBS durability is reasonably good for a regular hardware drive (annual failure rate of between 0.1% - 0.2%). On the other hand, that is very poor if you don't have backups! By contrast, S3 durability is extremely high. *If you care about your data, back it up to S3 with snapshots.*
- EBS has an **SLA** with **99.95%** uptime. See notes on high availability below.
- EBS volumes have a **volume type** indicating the physical storage type. The types called "standard" (**st1** or **sc1**) are actually old spinning-platter disks, which deliver only hundreds of IOPS — not what you want unless you're really trying to cut costs. Modern SSD-based **gp2** or **io1** are typically the options you want.

EFS

EFS Basics

- Homepage User guide FAQ Pricing
- **EFS** is Amazon's new (general release 2016) network filesystem.
- It is designed to be highly available and durable and each EFS file system object is redundantly stored across multiple availability zones.
- EFS is designed to be used as a shared network drive and it can automatically scale up to petabytes of stored data and thousands of instances

attached to it.

- It's presented as an NFSv4.1 server, so any compatible NFS client can mount it.
- EFS can offer higher throughput (multiple gigabytes per second) and better durability and availability than EBS (see the comparison table), but with higher latency.
- EFS is priced based on the amount of data stored and it costs much more than EBS, about three times as much compared to general purpose gp2 EBS volumes.
- Performance depends on the amount of data stored on it, which also determines the price:
  - Like EBS, EFS uses a credit based system. Credits are earned at a rate of 50 KiB/s per GiB of storage and consumed in bursts during reading/writing files or metadata. Unlike EBS, operations on metadata (file size, owner, date, etc.) also consume credits. The BurstCreditBalance metric in CloudWatch should be monitored to make sure the file system doesn't run out of credits.
  - Throughput capacity during bursts is also dependant on size. Under 1 TiB, throughput can go up to 100 MiB/s. Above that, 100 MiB/s is added for each stored TiB. So a file system storing 5 TiB would be able to burst at a rate of 500 MiB/s. Maximum throughput per EC2 instance is 250 MiB/s.
  - EFS has two performance modes that can only be set when a file system is created. One is "General Purpose", the other is "Max I/O". Max I/O scales higher, but at the cost of higher latency. When in doubt, use General Purpose, which is also the default. If the PercentIOLimit metric in CloudWatch hovers around 100%, Max I/O is recommended. Changing performance mode means creating a new EFS and migrating data.
- High availability is achieved by having mount targets in different subnets / availability zones.

EFS Tips

- With EFS being based on NFSv4.1, any directory on the EFS can be mounted directly, it doesn't have to be the root directory. One application could mount *fs-12345678:/prog1*, another *fs-12345678:/prog2*.

- User and group level permissions can be used to control access to certain directories on the EFS file system.

- **Sharing EFS filesystems:** One EFS filesystem can be used for multiple applications or services, but it should be considered carefully:

  Pros:

  - Because performance is based on total size of stored files, having everything on one drive will increase performance for everyone. One

44

application consuming credits faster than it can accumulate might be offset by another application that just stores files on EFS and rarely accesses them.

Cons:

- Since credits are shared, if one application over-consumes them, it will affect the others.
- A compromise is made with regards to security. All clients will have to have network access to the drive. Someone with root access on one client instance can mount any directory on the EFS and they have read-write access to all files on the drive, even if they don't have access to the applications hosted on other clients.

EFS Gotchas and Limitations

- A number of NFSv4.1 features are not supported and there are some limits to the service.
- As of 2016-11, EFS does not offer disk level encryption, though it is on the roadmap.
- Some applications, like SQLite and IPython, might not work properly on EFS when accessed from multiple clients. This is because lock upgrades and downgrades are not supported. There might be workarounds for some issues.
- Mounting EFS over VPN connection, VPC peering, or AWS Direct Connect is not supported.
- Using an EFS volume on Windows is not supported, apparently due to Microsoft implementing NFS differently.
- When a file is uploaded to EFS, it can take hours for EFS to update the details for billing and burst credit purposes.
- Metadata operations can be costly in terms of burst credit consumption. Recursively traversing a tree containing thousands of files can easily ramp up to tens or even hundreds of megabytes of burst credits being consumed, even if no file is being touched. Commands like `find` or `chown -R` can have an adverse impact on performace if run periodically.
- Mount points are AZ-based. In an Auto scaling group spread across zones, you can end up with instances in one zone mounting EFS from a different zone. That might decrease performance and would create an unintended single point of failure. One way to fix it would be a shell script that runs before network drives are mounted and edits /etc/fstab with the proper AZ.

Load Balancers

Load Balancer Basics

- AWS has 2 load balancing products - "Classic Load Balancers" (CLBs) and "Application Load Balancers" (ALBs).

- Before the introduction of ALBs, "Classic Load Balancers" were known as "Elastic Load Balancers" (ELBs), so older documentation, tooling, and blog posts may still reference "ELBs".
- CLBs have been around since 2009 while ALBs are a recent (2016) addition to AWS.
- CLBs support TCP and HTTP load balancing while ALBs support HTTP load balancing only.
- Both can optionally handle termination for a single SSL certificate.
- Both can optionally perform active health checks of instances and remove them from the destination pool if they become unhealthy.
- CLBs don't support complex / rule-based routing, while ALBs support a (currently small) set of rule-based routing features.
- CLBs can only forward traffic to a single globally configured port on destination instances, while ALBs can forward to ports that are configured on a per-instance basis, better supporting routing to services on shared clusters with dynamic port assignment (like ECS or Mesos).
- CLBs are supported in EC2 Classic as well as in VPCs while ALBs are supported in VPCs only.

Load Balancer Tips

- If you don't have opinions on your load balancing up front, and don't have complex load balancing needs like application-specific routing of requests, it's reasonable just to use an CLB or ALB for load balancing instead.
- Even if you don't want to think about load balancing at all, because your architecture is so simple (say, just one server), put a load balancer in front of it anyway. This gives you more flexibility when upgrading, since you won't have to change any DNS settings that will be slow to propagate, and also it lets you do a few things like terminate SSL more easily.
- **CLBs and ALBs have many IPs:** Internally, an AWS load balancer is simply a collection of individual software load balancers hosted within EC2, with DNS load balancing traffic among them. The pool can contain many IPs, at least one per availability zone, and depending on traffic levels. They also support SSL termination, which is very convenient.
- **Scaling:** CLBs and ALBs can scale to very high throughput, but scaling up is not instantaneous. If you're expecting to be hit with a lot of traffic suddenly, it can make sense to load test them so they scale up in advance. You can also contact Amazon and have them "pre-warm" the load balancer.
- **Client IPs:** In general, if servers want to know true client IP addresses, load balancers must forward this information somehow. CLBs add the standard X-Forwarded-For header. When using an CLB as an HTTP load balancer, it's possible to get the client's IP address from this.
- **Using load balancers when deploying:** One common pattern is to swap instances in the load balancer after spinning up a new stack with your latest version, keep old stack running for one or two hours, and either

flip back to old stack in case of problems or tear it down.

Load Balancer Gotchas and Limitations

- CLBs and ALBs have **no fixed external IP** that all clients see. For most consumer apps this doesn't matter, but enterprise customers of yours may want this. IPs will be different for each user, and will vary unpredictably for a single client over time (within the standard EC2 IP ranges). And similarly, never resolve an CLB name to an IP and put it as the value of an A record — it will work for a while, then break!
- Some web clients or reverse proxies cache DNS lookups for a long time, which is problematic for CLBs and ALBs, since they change their IPs. This means after a few minutes, hours, or days, your client will stop working, unless you disable DNS caching. Watch out for Java's settings and be sure to adjust them properly. Another example is nginx as a reverse proxy, which normally resolves backends only at start-up (although there is a way to get around this).
- It's not unheard of for IPs to be recycled between customers without a long cool-off period. So as a client, if you cache an IP and are not using SSL (to verify the server), you might get not just errors, but responses from completely different services or companies!
- As an operator of a service behind an CLB or ALB, the latter phenomenon means you can also see puzzling or erroneous requests by clients of other companies. This is most common with clients using back-end APIs (since web browsers typically cache for a limited period).
- CLBs and ALBs take time to scale up, it does not handle sudden spikes in traffic well. Therefore, if you anticipate a spike, you need to "pre-warm" the load balancer by gradually sending an increasing amount of traffic.
- Tune your healthchecks carefully — if you are too aggressive about deciding when to remove an instance and conservative about adding it back into the pool, the service that your load balancer is fronting may become inaccessible for seconds or minutes at a time. Be extra careful about this when an autoscaler is configured to terminate instances that are marked as being unhealthy by a managed load balancer.
- CLB HTTPS listeners don't support Server Name Indication (SNI). If you need SNI, you can work around this limitation by either providing a certificate with Subject Alternative Names (SANs) or by using TCP listeners and terminating SSL at your backend.

CLB

CLB Basics

- Homepage   User guide   FAQ   Pricing
- Classic Load Balancers, formerly known as Elastic Load Balancers, are HTTP and TCP load balancers that are managed and scaled for you by Amazon.

CLB Tips

- **Best practices:** This article is a must-read if you use CLBs heavily, and has a lot more detail.

CLB Gotchas and Limitations

- In general, CLBs are not as "smart" as some load balancers, and don't have fancy features or fine-grained control a traditional hardware load balancer would offer. For most common cases involving sessionless apps or cookie-based sessions over HTTP, or SSL termination, they work well.

- `By default, CLBs will refuse to route traffic from a load balancer in one Availabili`

- Complex rules for directing traffic are not supported. For example, you can't direct traffic based on a regular expression in the URL, like HAProxy offers.

- **Apex DNS names:** Once upon a time, you couldn't assign an CLB to an apex DNS record (i.e. example.com instead of foo.example.com) because it needed to be an A record instead of a CNAME. This is now possible with a Route 53 alias record directly pointing to the load balancer.

- CLBs use HTTP keep-alives on the internal side. This can cause an unexpected side effect: Requests from different clients, each in their own TCP connection on the external side, can end up on the same TCP connection on the internal side. Never assume that multiple requests on the same TCP connection are from the same client!

- Traffic between CLBs and back-end instances in the same subnet **will** have Network ACL rules evaluated (EC2 to EC2 traffic in the same subnet would not have Network ACL rules evaluated). If the default '0.0.0.0/0 ALLOW' rule is removed from the Network ACL applied to the subnet, a rule that allows traffic on both the health check port and any listener port must be added.

ALB

ALB Basics

- Homepage User guide FAQ Pricing
- **Websockets and HTTP/2** are now supported.
- Prior to the Application Load Balancer, you were advised to use TCP instead of HTTP as the protocol to make it work (as described here) and use the obscure but useful Proxy Protocol (more on this) to pass client IPs over a TCP load balancer.

ALB Tips

- Use ALBs to route to services that are hosted on shared clusters with dynamic port assignment (like ECS or Mesos).
- ALBs support HTTP path-based routing (send HTTP requests for "/api/" -> *{target-group-1}, "/blog/"* -> {target group 2}).

ALB Gotchas and Limitations

- ALBs only support HTTP/2 over HTTPS (no plain-text HTTP/2).
- ALBs only support HTTP/2 to external clients and not to internal resources (instances/containers).
- ALBs support HTTP routing but not port-based TCP routing.
- ALBs do not (yet) support routing based on HTTP "Host" header or HTTP verb.
- Instances in the ALB's target groups have to either have a single, fixed healthcheck port ("EC2 instance"-level healthcheck) or the healthcheck port for a target has to be the same as its application port ("Application instance"-level healthcheck) - you can't configure a per-target healthcheck port that is different than the application port.
- ALBs are VPC-only (they are not available in EC2 Classic)
- In a target group, if there is no healthy target, all requests are routed to all targets. For example, if you point a listener at a target group containing a single service that has a long initialization phase (during which the health checks would fail), requests will reach the service while it is still starting up.

Elastic IPs

Elastic IP Basics

- Documentation FAQ Pricing
- **Elastic IPs** are static IP addresses you can rent from AWS to assign to EC2 instances.

Elastic IP Tips

- **Prefer load balancers to elastic IPs:** For single-instance deployments, you could just assign elastic IP to an instance, give that IP a DNS name, and consider that your deployment. Most of the time, you should provision a load balancer instead:
  - It's easy to add and remove instances from load balancers. It's also quicker to add or remove instances from a load balancer than to reassign an elastic IP.
  - It's more convenient to point DNS records to load balancers, instead of pointing them to specific IPs you manage manually. They can also be Route 53 aliases, which are easier to change and manage.

– But in some situations, you do need to manage and fix IP addresses of EC2 instances, for example if a customer needs a fixed IP. These situations require elastic IPs.
- Elastic IPs are limited to 5 per account. It's possible to request more.
- If an Elastic IP is not attached to an active resource there is a small hourly fee.
- Elastic IPs are no extra charge as long as you're using them. They have a (small) cost when not in use, which is a mechanism to prevent people from squatting on excessive numbers of IP addresses.

Elastic IP Gotchas and Limitations

- There is officially no way to allocate a contiguous block of IP addresses, something you may desire when giving IPs to external users. Though when allocating at once, you may get lucky and have some be part of the same CIDR block.

Glacier

Glacier Basics

- Homepage Developer guide FAQ Pricing
- **Glacier** is a lower-cost alternative to S3 when data is infrequently accessed, such as for archival purposes.
- It's only useful for data that is rarely accessed. It generally takes 3-5 hours to fulfill a retrieval request.
- AWS has not officially revealed the storage media used by Glacier; it may be low-spin hard drives or even tapes.

Glacier Tips

- You can physically ship your data to Amazon to put on Glacier on a USB or eSATA HDD.

Glacier Gotchas and Limitations

- Getting files off Glacier is glacially slow (typically 3-5 hours or more).
- Due to a fixed overhead per file (you pay per PUT or GET operation), uploading and downloading many small files on/to Glacier might be very expensive. There is also a 32k storage overhead per file. Hence it's a good idea is to archive files before upload.
- Glacier's pricing policy is reportedly pretty complicated: "Glacier data retrievals are priced based on the peak hourly retrieval capacity used within a calendar month." Some more info can be found here and here.
- Be aware of the per-object costs of archiving S3 data to Glacier. It costs $0.05 per 1,000 requests. If you have large numbers of S3 objects of relatively small size, it will take time to reach a break-even point (initial

archiving cost versus lower storage pricing).

RDS

RDS Basics

- Homepage User guide FAQ Pricing (see also ec2instances.info/rds/)
- **RDS** is a managed relational database service, allowing you to deploy and scale databases more easily. It supports Oracle, Microsoft SQL Server, PostgreSQL, MySQL, MariaDB, and Amazon's own Aurora.
- RDS offers out of the box support for high availability and failover for your databases.

RDS Tips

- If you're looking for the managed convenience of RDS for other data stores such as MongoDB or Cassandra, you may wish to consider third-party services from providers such as mLab, Compose, or InstaClustr.
- Make sure to create a new parameter group and option group for your database since the default parameter group does not allow dynamic configuration changes.
- RDS instances start with a default timezone of UTC. If necessary, this can be changed to a different timezone.

RDS Gotchas and Limitations

- RDS instances run on EBS volumes (either general-purpose or provisioned IOPS), and hence are constrained by EBS performance.
- Verify what database features you need, as not everything you might want is available on RDS. For example, if you are using Postgres, check the list of supported features and extensions. If the features you need aren't supported by RDS, you'll have to deploy your database yourself.
- If you use the failover support offered by RDS, keep in mind that it is based on DNS changes, and make sure that your client reacts to these changes appropriately. This is particularly important for Java, given how its DNS resolver's TTL is configured by default.
- **DB migration to RDS:** While importing your database into RDS ensure you take into consideration the maintenance window settings. If a backup is running at the same time, your import can take a considerably longer time than you would have expected.
- Database sizes are limited to **6TB** for all database engines except for SQL Server which has a **4TB** limit and Aurora which supports up to **64TB** databases.

RDS MySQL and MariaDB

RDS MySQL and MariaDB Basics

- RDS offers MySQL versions 5.5, 5.6, and 5.7.

RDS MySQL and MariaDB Tips

- MySQL RDS allows access to binary logs.
- Multi-AZ instances of MySQL transparently replicate data across AZs using DRBD. Automated backups of multi-AZ instances run off the backup instance to reduce latency spikes on the primary.

- **MySQL vs MariaDB vs Aurora:** If you prefer a MySQL-style database but are starting something new, you probably should consider Aurora and MariaDB as well. **Aurora** has increased availability and is the next-generation solution. That said, Aurora may not be as fast relative to MySQL as is sometimes reported, and is more complex to administer. **MariaDB**, the modern community fork of MySQL, likely now has the edge over MySQL for many purposes and is supported by RDS.

RDS MySQL and MariaDB Gotchas and Limitations

- **No SUPER privileges.** RDS provides some stored procedures to perform some tasks that require SUPER privileges such as starting or stopping replication.
- You can replicate to non-RDS instances of MySQL, but replication to these instances will break during AZ failovers.
- There is no ability to manually CHANGE MASTER on replicas, so they must all be rebuilt after a failover of the master.

RDS Aurora

RDS Aurora Basics

- Amazon's proprietary fork of MySQL intended to scale up for high concurrency workloads. Generally speaking, individual query performance under Aurora is not expected to improve significantly relative to MySQL or MariaDB, but Aurora is intended to maintain performance while executing many more queries concurrently than an equivalent MySQL or MariaDB server could handle.
- Notable new features include:
    - Log-structured storage instead of B-trees to improve write performance
    - Out-of-process buffer pool so that databases instances can be restarted without clearing the buffer pool

- The underlying physical storage is a specialized SSD array that automatically maintains 6 copies of your data across 3 AZs.
- Aurora read replicas share the storage layer with the write master which significantly reduces replica lag, eliminates the need for the master to write and distribute the binary log for replication, and allows for zero-data-loss failovers from the master to a replica. The master and all the read replicas that share storage are known collectively as an **Aurora cluster**.

RDS Aurora Tips

- In order to take advantage of Aurora's higher concurrency, applications should be configured with large database connection pools and should execute as many queries concurrently as possible. For example, Aurora servers have been tested to produce increasing performance on some OLTP workloads with up to 5,000 connections.
- Aurora scales well with multiple CPUs and may require a large instance class for optimal performance.

- Because Aurora is based on MySQL 5.6.10, avoiding any MySQL features from 5.7 or later will ease the transition from a MySQL-compatible database into Aurora.
- The easiest migration path to Aurora is restoring a database snapshot from MySQL 5.6. The next easiest method is restoring a dump from a MySQL-compatible database such as MariaDB. For low-downtime migrations from other MySQL-compatible databases, you can set up an Aurora instance as a replica of your existing database. If none of those methods are options, Amazon offers a fee-based data migration service.
- You can replicate from an Aurora cluster to MySQL or to another Aurora cluster. This requires binary logging to be enabled and is not as performant as native Aurora replication.

RDS Aurora Gotchas and Limitations

- Aurora is based on MySQL 5.6.10 with some cherry-picking of later MySQL features. It is missing most 5.7 features as well as some online DDL features introduced in 5.6.17.

RDS SQL Server

RDS SQL Server Basics

- RDS offers SQL Server 2008 R2, 2012, and 2014 including Express, Web, Standard and Enterprise (2008 R2 and 2012 only for Enterprise)

RDS SQL Server Tips

- Recently added support for backup and restore to/from S3 which may make it an attractive DR option for on-premises installations.

RDS SQL Server Gotchas and Limitations

- The user is granted only db_owner privileges for each database on the instance.
- Storage cannot be expanded for existing databases. If you need more space, you must restore your database on a new instance with larger storage.
- There is a **4TB** database size limit for non-Express editions.
- Limited to 30 databases per instance

DynamoDB

DynamoDB Basics

- Homepage Developer guide FAQ Pricing
- **DynamoDB** is a NoSQL database with focuses on speed, flexibility, and scalability.
- DynamoDB is priced on a combination of throughput and storage.

DynamoDB Alternatives and Lock-in

- Unlike the technologies behind many other Amazon products, DynamoDB is a proprietary AWS product with no interface-compatible alternative available as an open source project. If you tightly couple your application to its API and featureset, it will take significant effort to replace.
- The most commonly used alternative to DynamoDB is Cassandra.

DynamoDB Tips

- There is a **local version of DynamoDB** provided for developer use.
- DynamoDB Streams provides an ordered stream of changes to a table. Use it to replicate, back up, or drive events off of data
- DynamoDB can be used as a simple locking service.
- DynamoDB indexing can include **primary keys**, which can either be a single-attribute hash key or a composite hash-key range. You can also query non-primary key attributes using **secondary indexes**.
- **Data Types:** DynamoDB supports three data types – **number**, **string**, and **binary** – in both scalar and multi-valued sets. DynamoDB can also support **JSON**.

DynamoDB Gotchas and Limitations

- DynamoDB doesn't provide an easy way to bulk-load data (it is possible through Data Pipeline) and this has some unfortunate consequences. Since you need to use the regular service APIs to update existing or create new rows, it is common to temporarily turn up a destination table's write throughput to speed import. But when the table's write capacity is increased, DynamoDB may do an irreversible split of the partitions underlying the table, spreading the total table capacity evenly across the new generation of tables. Later, if the capacity is reduced, the capacity for each partition is also reduced but the total number of partitions is not, leaving less capacity for each partition. This leaves the table in a state where it much easier for hotspots to overwhelm individual partitions.
- It is important to make sure that DynamoDB resource limits are compatible with your dataset and workload. For example, the maximum size value that can be added to a DynamoDB table is 400 KB (larger items can be stored in S3 and a URL stored in DynamoDB).
- Dealing with **time series data** in DynamoDB can be challenging. A global secondary index together with down sampling timestamps can be a possible solution as explained here.
- DynamoDB does not allow an empty string as a valid attribute value. The most common work-around is to use a substitute value instead of leaving the field empty.

ECS

ECS Basics

- Homepage Developer guide FAQ Pricing
- **ECS** (EC2 Container Service) is a relatively new service (launched end of 2014) that manages clusters of services deployed via Docker.
- See the Containers and AWS section for more context on containers.
- ECS is growing in adoption, especially for companies that embrace microservices.
- Deploying Docker directly in EC2 yourself is another common approach to using Docker on AWS. Using ECS is not required, and ECS does not (yet) seem to be the predominant way many companies are using Docker on AWS.
- It's also possible to use Elastic Beanstalk with Docker, which is reasonable if you're already using Elastic Beanstalk.
- Using Docker may change the way your services are deployed within EC2 or Elastic Beanstalk, but it does not radically change how most other services are used.
- ECR (EC2 Container Registry) is Amazon's managed Docker registry service. While simpler than running your own registry, it is missing some features that might be desired by some users:
  - Doesn't support cross-region replication of images.

* If you want fast fleet-wide pulls of large images, you'll need to push your image into a region-local registry.
  - Doesn't support custom domains / certificates.
- A container's health is monitored via CLB or ALB. Those can also be used to address a containerized service. When using an ALB you do not need to handle port contention (i.e. services exposing the same port on the same host) since an ALB's target groups can be associated with ECS-based services directly.

ECS Tips

- **Log drivers:** ECS supports multiple log drivers (awslogs, splunk, fluentd, syslog, JSON, …). Use `awslogs` for CloudWatch (make sure a group is made for the logs first). Drivers such as fluentd are not enable by default. To do so, install the agent and enable the driver by adding `ECS_AVAILABLE_LOGGING_DRIVERS='["awslogs","fluentd"]'` to `/etc/ecs/ecs.config`.
- This blog from Convox (and commentary) lists a number of common challenges with ECS as of early 2016.

ECS Alternatives and Lock-in

- Kubernetes: Extensive container platform. Available as a hosted solution on Google Cloud (https://cloud.google.com/container-engine/) and AWS (https://tectonic.com/).
- Nomad: Orchestrator/Scheduler, tightly integrated in the Hashicorp stack (Consul, Vault, etc).

Please help expand this incomplete section.

Lambda

Lambda Basics

- Homepage Developer guide FAQ Pricing
- **Lambda** is a relatively new service (launched at end of 2014) that offers a different type of compute abstraction: A user-defined function that can perform a small operation, where AWS manages provisioning and scheduling how it is run.

Lambda Tips

- **What does "serverless" mean?** This idea of using Lambda for application logic has grown to be called **serverless** since you don't explicitly manage any server instances, as you would with EC2. This term is a bit confusing since the functions themselves do of course run on servers managed by AWS. Serverless, Inc. also uses this word for the name of their

company and their own open source framework, but the term is usually meant more generally.

- The release of Lambda and API Gateway in 2015 triggered a startlingly rapid adoption in 2016, with many people writing about serverless architectures in which many applications traditionally solved by managing EC2 servers can be built without explicitly managing servers at all.
- **Frameworks:** Several frameworks for building and managing serverless deployment are emerging.
- The Awesome Serverless list gives a good set of examples of the relatively new set of tools and frameworks around Lambda.
- The **Serverless framework** is a leading new approach designed to help group and manage Lambda functions. It's approaching version 1 as of August 2016) and is popular among a small number of users.

Lambda Alternatives and Lock-in

- Other clouds offer similar services with different names, including Google Cloud Functions, Azure Functions, and IBM OpenWhisk.

Lambda Gotchas and Limitations

- Lambda is a new technology. As of mid 2016, only a few companies are using it for large-scale production applications.
- Managing lots of Lambda functions is a workflow challenge, and tooling to manage Lambda deployments is still immature.
- AWS' official workflow around managing function versioning and aliases is painful.
- Currently as of October, 2016 Lambda functions can sometimes stop working for 2-3 minutes for failure recovery purposes according to a support ticket answer from Lambda development team. They are working to prevent this in the future.

Lambda Code Samples

- Fan-out is an example of using Lambda to "fan-out" or copy data from one service, in this case Kinesis, to multiple other AWS data services. Destinations for fan-out data in the sample include IoT, SQS and more.
- This AWS limit monitor using Lambdas shows use of multiple Lambdas for monitoring.
- This Lambda ECS Worker Pattern shows use of Lambda in a workflow where data from S3 is picked up by the Lambda, pushed to a queue, then sent to ECS for more processing.
- The Secure Pet Store is a sample Java application which uses Lambda and API Gateway with Cognito (for user identity).

Please help expand this incomplete section.

API Gateway

API Gateway Basics

- Homepage Developer guide FAQ Pricing
- **API Gateway** provides a scalable, secured front-end for service APIs, and can work with Lambda, Elastic Beanstalk, or regular EC2 services.
- It allows "serverless" deployment of applications built with Lambda.
- Switching over deployments after upgrades can be tricky. There are no built-in mechanisms to have a single domain name migrate from one API gateway to another one. So it may be necessary to build an additional layer in front (even another API Gateway) to allow smooth migration from one deployment to another.

API Gateway Alternatives and Lock-In

- Kong is an open-source, on-premises API and microservices gateway built on nginx with Lua. Kong is extensible through "plugins".
- Tyk is an open-source API gateway implemented in Go and available in the cloud, on-premises or hybrid.

API Gateway Gotchas and Limitations

- API Gateway only supports encrypted (https) endpoints, and does not support unencrypted HTTP. (This is probably a good thing.)
- API Gateway endpoints are always public, i.e. internet facing, and there is no mechanism to build private endpoints, e.g. for internal use on a VPC but endpoints and their related resources can, optionally, require authentication.
- API Gateway doesn't support multi-region deployments for high availability. It is a service that is deployed in a single region but comes with a global endpoint that is served from AWS edge locations (similar to a CloudFront distribution). You cannot have multiple API Gateways with the same hostname in different AWS regions and use Route 53 to distribute the traffic. More in this forum post.

Please help expand this incomplete section.

Route 53

Route 53 Basics

- Homepage Developer guide FAQ Pricing
- **Route 53** is AWS' DNS service.

Route 53 Alternatives and Lock-In

- Historically, AWS was slow to penetrate the DNS market (as it is often driven by perceived reliability and long-term vendor relationships) but

Route 53 has matured and is becoming the standard option for many companies. Route 53 is cheap by historic DNS standards, as it has a fairly large global network with geographic DNS and other formerly "premium" features. It's convenient if you are already using AWS.
- Generally you don't get locked into a DNS provider for simple use cases, but increasingly become tied in once you use specific features like geographic routing or Route 53's alias records.
- Many alternative DNS providers exist, ranging from long-standing premium brands like UltraDNS and Dyn to less well known, more modestly priced brands like DNSMadeEasy. Most DNS experts will tell you that the market is opaque enough that reliability and performance don't really correlate well with price.
- Route 53 is usually somewhere in the middle of the pack on performance tests, e.g. the SolveDNS reports.

Route 53 Tips

- Know about Route 53's "alias" records:
  - Route 53 supports all the standard DNS record types, but note that **alias resource record sets** are not standard part of DNS, but a specific Route 53 feature. (It's available from other DNS providers too, but each provider has a different name for it.)
  - Aliases are like an internal name (a bit like a CNAME) that is resolved internally on the server side. For example, traditionally you could have a CNAME to the DNS name of a CLB or ALB, but it's often better to make an alias to the same load balancer. The effect is the same, but in the latter case, externally, all a client sees is the target the record points to.
  - It's often wise to use alias record as an alternative to CNAMEs, since they can be updated instantly with an API call, without worrying about DNS propagation.
  - You can use them for CLBs/ALBs or any other resource where AWS supports it.
  - Somewhat confusingly, you can have CNAME and A aliases, depending on the type of the target.
  - Because aliases are extensions to regular DNS records, if exported, the output zone file will have additional non-standard "ALIAS" lines in it.
- **Latency-based routing** allows users around the globe to be automatically directed to the nearest AWS region where you are running, so that latency is reduced.
- Understand that domain registration and DNS management (hosted zones) are two separate Route 53 services. When you buy/transfer a domain, Route 53 automaticaly assigns four name servers to it (e.g. ns-2.awsdns-00.com). Route 53 also offers to automatically create a hosted zone for DNS management, but you are not required do your DNS management

in the same account or even in Route 53; you just need to create an NS record pointing to the servers assigned to your domain in Route 53.

- One use case would be to put your domain registration (very mission critical) in a bastion account while managing the hosted zones within another account which is accessible by your applications.

CloudFormation

CloudFormation Basics

- Homepage Developer guide FAQ Pricing at no additional charge
- **CloudFormation** offers mechanisms to create and update entire **stacks** comprised of many types of AWS resources. These CloudFormation stacks are defined in a **CloudFormation template** which is defined in JSON or YAML.
- CloudFormation itself has no additional charge itself; you pay for the underlying resources.

CloudFormation Alternatives and Lock-In

- Hashicorp's Terraform is a third-party alternative that can support other cloud platforms/providers including Azure and OpenStack.
- Some AWS features may not be available in Terraform (e.g. multi-AZ ElastiCache using Redis), and you may have to resort to embedded CloudFormation templates.

CloudFormation Tips

- Troposphere is a Python library that makes it much easier to create CloudFormation templates.
  - Currently supports AWS and OpenStack resource types.
  - Troposphere does not support all of the resources types you can describe with CloudFormation templates.
  - Built in error checking.
  - A recommended soft dependency is awacs, which allows you to generate AWS access policy in JSON by writing Python code.
- If you are building different stacks with similar layers, it may be useful to build separate templates for each layer that you can reuse using AWS::CloudFormation::Stack.
- Avoid hardcoding resource parameters that can potentially change. Use stack parameters as much as you can, and resort to default parameter values.
- Until 2016, CloudFormation used only an awkward JSON format that makes both reading and debugging difficult. To use it effectively typically involved building additional tooling, including converting it to YAML, but now this is supported directly.

- Wherever possible, export relevant physical IDs from your Stacks by defining Outputs in your CloudFormation Templates. These are the actual names assigned to the resources being created. Outputs can be returned from `DescribeStack` API calls, and get imported to other Stacks as part of the recent addition of cross-stack references.
- CloudFormation can be set up to send SNS notifications upon state changes, enabling programatic handling of situations where stacks fail to build, or simple email alerts so the appropriate people are informed.
- CloudFormation allows the use of **conditionals** when creating a stack.
    - One common way to leverage this capability is in support of multi-environment CloudFormation templates – by configuring them to use 'if-else' statements on the value of a parameter passed in (e.g. "env"), environment-specific values for things like VPC IDs, SecurityGroup IDs, and AMI names can be passed into reusable generic templates.
- **Version control your CloudFormation templates!** In the Cloud, an application is the combination of the code written and the infrastructure it runs on. By version controlling **both**, it is easy to roll back to known good states.

CloudFormation Gotchas and Limitations

- Modifications to stack resources made outside CloudFormation can potentially lead to stacks stuck in UPDATE_ROLLBACK_FAILED mode. Stacks in this state can't be recovered without help from AWS Support.
- CloudFormation is useful but complex and with a variety of pain points. Many companies find alternate solutions, and many companies use it, but only with significant additional tooling.
- CloudFormation can be very slow, especially for items like CloudFront distributions and Route53 CNAME entries.
- It's hard to assemble good CloudFormation configurations from existing state. AWS does offer a trick to do this, but it's very clumsy.
- Many users don't use CloudFormation at all because of its limitations, or because they find other solutions preferable. Often there are other ways to accomplish the same goals, such as local scripts (Boto, Bash, Ansible, etc.) you manage yourself that build infrastructure, or Docker-based solutions (Convox, etc.).

VPCs, Network Security, and Security Groups

VPC Basics

- Homepage User guide FAQ Security groups Pricing
- **VPC** (Virtual Private Cloud) is the virtualized networking layer of your AWS systems.
- Most AWS users should have a basic understanding of VPC concepts, but few need to get into all the details. VPC configurations can be trivial or

extremely complex, depending on the extent of your network and security needs.

- All modern AWS accounts (those created after 2013-12-04) are "EC2-VPC" accounts that support VPCs, and all instances will be in a default VPC. Older accounts may still be using "EC2-Classic" mode. Some features don't work without VPCs, so you probably will want to migrate.

VPC and Network Security Tips

- **Security groups** are your first line of defense for your servers. Be extremely restrictive of what ports are open to all incoming connections. In general, if you use CLBs, ALBs or other load balancing, the only ports that need to be open to incoming traffic would be port 22 and whatever port your application uses. Security groups access policy is 'deny by default'.
- **Port hygiene:** A good habit is to pick unique ports within an unusual range for each different kind of production service. For example, your web frontend might use 3010, your backend services 3020 and 3021, and your Postgres instances the usual 5432. Then make sure you have fine-grained security groups for each set of servers. This makes you disciplined about listing out your services, but also is more error-proof. For example, should you accidentally have an extra Apache server running on the default port 80 on a backend server, it will not be exposed.
- **Migrating from Classic**: For migrating from older EC2-Classic deployments to modern EC2-VPC setup, this article may be of help.
- For basic AWS use, one default VPC may be sufficient. But as you scale up, you should consider mapping out network topology more thoroughly. A good overview of best practices is here.
- Consider controlling access to you private AWS resources through a VPN.
  - You get better visibility into and control of connection and connection attempts.
  - You expose a smaller surface area for attack compared to exposing separate (potentially authenticated) services over the public internet.
    * e.g. A bug in the YAML parser used by the Ruby on Rails admin site is much less serious when the admin site is only visible to the private network and accessed through VPN.
  - Another common pattern (especially as deployments get larger, security or regulatory requirements get more stringent, or team sizes increase) is to provide a bastion host behind a VPN through which all SSH connections need to transit.
- Consider using other security groups as sources for security group rules instead of using CIDRs — that way, all hosts in the source security group and only hosts in that security group are allowed access. This is a much more dynamic and secure way of managing security group rules.
- **VPC Flow Logs** allow you to monitor the network traffic to, from, and within your VPC. Logs are stored in CloudWatch Logs groups, and can be used for security monitoring (with third party tools), performance eval-

uation, and forensic investigation.

  – See the VPC Flow Logs User Guide for basic information.
  – See the flowlogs-reader CLI tool and Python library to retrieve and work with VPC Flow Logs.

VPC and Network Security Gotchas and Limitations

- Security groups are not shared across data centers, so if you have infrastructure in multiple data centers, you should make sure your configuration/deployment tools take that into account.
- Be careful when choosing your VPC IP CIDR block: If you are going to need to make use of ClassicLink, make sure that your private IP range doesn't overlap with that of EC2 Classic.
- If you are going to peer VPCs, carefully consider the cost of data transfer between VPCs, since for some workloads and integrations, this can be prohibitively expensive.
- New RDS instances require a subnet group within your VPC. If you're using the default VPC this isn't a concern, it will contain a subnet for each availability zone in your region. However, if you're creating your own VPC and plan on using RDS, make sure you have at least two subnets within the VPC to act as the subnet group.
- If you delete the default VPC, the only way to create another VPC marked as "default" is to contact AWS technical support. See this note in the documentation.
- Be careful with VPC VPN credentials! If lost or compromised, the VPN endpoint must be deleted and recreated. See the instructions for Replacing Compromised Credentials.

KMS

KMS Basics

- Homepage Developer guide FAQ Pricing
- **KMS** (Key Management Service) is a secure service for creating, storing and auditing usage of cryptographic keys.
- **Service integration:** KMS integrates with other AWS services: EBS, Elastic Transcoder, EMR, Redshift, RDS, SES, S3, WorkMail and Workspaces.
- **Encryption APIs:** The Encrypt and Decrypt API allow you to encrypt and decrypt data on the KMS service side, never exposing the master key contents.
- **Data keys:** The GenerateDataKey API generates a new key off of a master key. The data key contents are exposed to you so you can use it to encrypt and decrypt any size of data in your application layer. KMS does not store, manage or track data keys, you are responsible for this in your application.
- **Auditing:** Turn on CloudTrail to audit all KMS API events.

- **Access:** Use key policies and IAM policies to grant different levels of KMS access. For example, you create an IAM policy that only allows a user to encrypt and decrypt with a specific key.

KMS Tips

- It's very common for companies to manage keys completely via home-grown mechanisms, but it's far preferable to use a service such as KMS from the beginning, as it encourages more secure design and improves policies and processes around managing keys.
- A good motivation and overview is in this AWS presentation.
- The cryptographic details are in this AWS whitepaper.
- This blog from Convox demonstrates why and how to use KMS for encryption at rest.

KMS Gotchas and Limitations

- The Encrypt API only works with < 4KB of data. Larger data requires generating and managing a data key in your application layer.
- KMS audit events are not available in the CloudTrail Lookup Events API. You need to look find them in the raw .json.gz files that CloudTrail saves in S3.

CloudFront

CloudFront Basics

- Homepage Developer guide FAQ Pricing
- **CloudFront** is AWS' content delivery network (CDN).
- Its primary use is improving latency for end users through accessing cacheable content by hosting it at over 60 global edge locations.

CloudFront Alternatives and Lock-in

- CDNs are a highly fragmented market. CloudFront has grown to be a leader, but there are many alternatives that might better suit specific needs.

CloudFront Tips

- **IPv6** is now supported!
- **HTTP/2** is now supported! Clients must support TLS 1.2 and SNI.
- While the most common use is for users to browse and download content (GET or HEAD methods) requests, CloudFront also supports (since 2013) uploaded data (POST, PUT, DELETE, OPTIONS, and PATCH).
    - You must enable this by specifying the allowed HTTP methods when you create the distribution.

- Interestingly, the cost of accepting (uploaded) data is usually less than for sending (downloaded) data.
- In its basic version, CloudFront supports SSL via the SNI extension to TLS, which is supported by all modern web browsers. If you need to support older browsers, you need to pay a few hundred dollars a month for dedicated IPs.
  - Consider invalidation needs carefully. CloudFront does support invalidation of objects from edge locations, but this typically takes many minutes to propagate to edge locations, and costs $0.005 per request after the first 1000 requests. (Some other CDNs support this better.)
- Everyone should use TLS nowadays if possible. Ilya Grigorik's table offers a good summary of features regarding TLS performance features of CloudFront.
- An alternative to invalidation that is often easier to manage, and instant, is to configure the distribution to cache with query strings and then append unique query strings with versions onto assets that are updated frequently.
- For good web performance, it is recommended to enable compression on CloudFront distributions if the origin is S3 or another source that does not already compress.

CloudFront Gotchas and Limitations

- If using S3 as a backing store, remember that the endpoints for website hosting and for general S3 are different. Example: "bucketname.s3.amazonaws.com" is a standard S3 serving endpoint, but to have redirect and error page support, you need to use the website hosting endpoint listed for that bucket, e.g. "bucketname.s3-website-us-east-1.amazonaws.com" (or the appropriate region).
- By default, CloudFront will not forward HTTP Host: headers through to your origin servers. This can be problematic for your origin if you run multiple sites switched with host headers. You can enable host header forwarding in the default cache behavior settings.
- 4096-bit SSL certificates: CloudFront do not support 4096-bit SSL certificates as of late 2016. If you are using an externally issued SSL certificate, you'll need to make sure it's 2048 bits. See ongoing discussion.

DirectConnect

DirectConnect Basics

- Homepage   User guide   FAQ   Pricing
- **Direct Connect** is a private, dedicated connection from your network(s) to AWS.

DirectConnect Tips

- If your data center has a partnering relationship with AWS, setup is streamlined.
- Use for more consistent predictable network performance guarantees (**1 Gbps** or **10 Gbps** per link).
- Use to peer your colocation, corporate, or physical datacenter network with your VPC(s).
    - Example: Extend corporate LDAP and/or Kerberos to EC2 instances running in a VPC.
    - Example: Make services that are hosted outside of AWS for financial, regulatory, or legacy reasons callable from within a VPC.

Redshift

Redshift Basics

- Homepage Developer guide FAQ Pricing
- **Redshift** is AWS' managed data warehouse solution, which is massively parallel, scalable, and columnar. It is very widely used. It was built using ParAccel technology and exposes Postgres-compatible interfaces.

Redshift Alternatives and Lock-in

- Whatever data warehouse you select, your business will likely be locked in for a long time. Also (and not coincidentally) the data warehouse market is highly fragmented. Selecting a data warehouse is a choice to be made carefully, with research and awareness of the market landscape and what business intelligence tools you'll be using.

Redshift Tips

- Although Redshift is mostly Postgres-compatible, its SQL dialect and performance profile are different.
- Redshift supports only 12 primitive data types. (List of unsupported Postgres types)
- It has a leader node and computation nodes (the leader node distributes queries to the computation ones). Note that some functions can be executed only on the lead node.
- Make sure to create a new cluster parameter group and option group for your database since the default parameter group does not allow dynamic configuration changes.
- Major third-party BI tools support Redshift integration (see Quora).
- Top 10 Performance Tuning Techniques for Amazon Redshift provides an excellent list of performance tuning techniques.
- Amazon Redshift Utils contains useful utilities, scripts and views to simplify Redshift ops.

- VACUUM regularly following a significant number of deletes or updates to reclaim space and improve query performance.
- Avoid performing blanket VACUUM or ANALYZE operations at a cluster level. The checks on each table to determine whether VACUUM or ANALYZE action needs to be taken is wasteful. Only perform ANALYZE and VACUUM commands on the objects that require it. Utilize the Analyze & Vacuum Schema Utility to perform this work. The SQL to determine whether a table needs to be VACUUMed or ANALYZEd can be found in the Schema Utility README if you wish to create your own maintenance process.
- Redshift provides various column compression options to optimize the stored data size. AWS strongly encourages users to use automatic compression at the COPY stage, when Redshift uses a sample of the data being ingested to analyze the column compression options. However, automatic compression can only be applied to an empty table with no data. Therefore, make sure the initial load batch is big enough to provide Redshift with a representative sample of the data (the default sample size is 100000 rows).
- Redshift uses columnar storage, hence it does not have indexing capabilities. You can, however, use distribution key distkey and sort key sortkey to improve performance. Redshift has two type of sort keys: compounding sort key and interleaved sort key.
- A compound sort key is made up of all columns listed in the sort key definition. It is most useful when you have queries with operations using prefix of the sortkey.
- An interleaved sort key on the other hand gives equal weight to each column or a subset of columns in the sort key. So if you don't know ahead of time which column you want to choose for sorting and filtering, this is a much better choice than the compound key.Here is an example using interleaved sort key.
- **Distribution strategies:** Since data in Redshift is physically distributed among nodes, choosing the right data **distribution key** and distribution style is crucial for adequate query performance. There are three possible distribution style settings — **EVEN** (the default), **KEY**, or **ALL**. Use KEY to collocate join key columns for tables which are joined in queries. Use ALL to place the data in small-sized tables on all cluster nodes.

Redshift Gotchas and Limitations

- While Redshift can handle heavy queries well, it does not scale horizontally, i.e. does not handle multiple queries in parallel. Therefore, if you expect a high parallel load, consider replicating or (if possible) sharding your data across multiple clusters.
- The leader node, which manages communications with client programs and all communication with compute nodes, is the single point of failure.

- Although most Redshift queries parallelize well at the compute node level, certain stages are executed on the leader node, which can become the bottleneck.
- Redshift data commit transactions are very expensive and serialized at the cluster level. Therefore, consider grouping multiple mutation commands (COPY/INSERT/UPDATE) commands into a single transaction whenever possible.
- Redshift does not support multi-AZ deployments. Building multi-AZ clusters is not trivial. Here is an example using Kinesis.
- Beware of storing multiple small tables in Redshift. The way Redshift tables are laid out on disk makes it impractical. The minimum space required to store a table (in MB) is nodes * slices/node * columns. For example, on a 16 node cluster an empty table with 20 columns will occupy 640MB on disk.
- Query performance degrades significantly during data ingestion. WLM (Workload Management) tweaks help to some extent. However, if you need consistent read performance, consider having replica clusters (at the extra cost) and swap them during update.
- Never resize a live cluster. The resize operation takes hours depending on the dataset size. In rare cases, the operation may also get stuck and you'll end up having a non-functional cluster. The safer approach is to create a new cluster from a snapshot, resize the new cluster and shut down the old one.
- Redshift has **reserved keywords** that are not present in Postgres (see full list here). Watch out for DELTA (Delta Encodings).
- Redshift does not support many Postgres functions, most notably several date/time-related and aggregation functions. See the full list here.
- Compression on sort key can result in significant performance impact. So if your Redshift queries involving sort key(s) are slow, you might want to consider removing compression on a sort key.
- Choosing a sort key is very important since you can not change a table's sort key after it is created. If you need to change the sort or distribution key of a table, you need to create a new table with the new key and move your data into it with a query like "insert into new_table select * from old_table".
- When moving data with a query that looks like "insert into x select from y", you need to have twice as much disk space available as table "y" takes up on the cluster's disks. Redshift first copies the data to disk and then to the new table. Here is a good article on how to this for big tables.

EMR

EMR Basics

- Homepage Release guide FAQ Pricing
- **EMR** (which used to stand for Elastic Map Reduce, but not anymore,

since it now extends beyond map-reduce) is a service that offers managed deployment of Hadoop, HBase and Spark. It reduces the management burden of setting up and maintaining these services yourself.

EMR Alternatives and Lock-in

- Most of EMR is based on open source technology that you can in principle deploy yourself. However, the job workflows and much other tooling is AWS-specific. Migrating from EMR to your own clusters is possible but not always trivial.

EMR Tips

- EMR relies on many versions of Hadoop and other supporting software. Be sure to check which versions are in use.
- Off-the-shelf EMR and Hadoop can have significant overhead when compared with efficient processing on a single machine. If your data is small and performance matters, you may wish to consider alternatives, as this post illustrates.
- Python programmers may want to take a look at Yelp's mrjob.
- **Hourly pricing roundoff:** Since EMR jobs are billed at one-hour granularity, considering changing the number and/or type of instances that your job runs in order to best make use of that time slice (fewer / smaller instances to make more efficient use of an undersubscribed hour, more / larger instances to reduce your job's runtime).
- It takes time to tune performance of EMR jobs, which is why third-party services such as Qubole's data service are gaining popularity as ways to improve performance or reduce costs.

EMR Gotchas and Limitations

- **EMR costs** can pile up quickly since it involves lots of instances, efficiency can be poor depending on cluster configuration and choice of workload, and accidents like hung jobs are costly. See the section on EC2 cost management, especially the tips there about Spot instances and avoiding hourly billing. This blog post has additional tips.
- Beware of "double-dipping". With EMR, you pay for the EC2 capacity and the service fees. In addition, EMR syncs task logs to S3, which means you pay for the storage and **PUT requests** at S3 standard rates. While the log files tend to be relatively small, every Hadoop job, depending on the size, generates thousands of log files that can quickly add up to thousands of dollars on the AWS bill. YARN's log aggregation is not available on EMR.

Kinesis Streams

Kinesis Streams Basics

- Homepage   Developer guide   FAQ   Pricing
- **Kinesis Streams** (which used to be only called Kinesis, before Kinesis Firehose and Kinesis Analytics were launched) is a service that allows you to ingest high-throughput data streams for immediate or delayed processing by other AWS services.
- Kinesis Streams' subcomponents are called **shards**. Each shard provides 1MB/s of write capacity and 2MB/s of read capacity at a maximum of 5 reads per second. A stream can have its shards programmatically increased or decreased based on a variety of metrics.
- All records entered into a Kinesis Stream are assigned a unique sequence number as they are captured. The records in a Stream are ordered by this number, so any time-ordering is preserved.
- This page summarizes key terms and concepts for Kinesis Streams.

Kinesis Streams Alternatives and Lock-in

- Kinesis is most closely compared to Apache Kafka, an open-source data ingestion solution. It is possible to set up a Kafka cluster hosted on EC2 instances (or any other VPS), however you are responsible for managing and maintaining both Zookeeper and the Kafka brokers in a highly available configuration. Confluent has a good blog post with their recommendations on how to do this here, which has links on the bottom to several other blogs they have written on the subject.
- Kinesis uses very AWS-specific APIs, so you should be aware of the potential future costs of migrating away from it, should you choose to use it.
- An application that efficiently uses Kinesis Streams will scale the number of shards up and down based on the required streaming capacity. (Note there is no direct equivalent to this with Apache Kafka.)

Kinesis Streams Tips

- The KCL (Kinesis Client Library) provides a skeleton interface for Java, Node, Python, Ruby and .NET programs to easily consume data from a Kinesis Stream. In order to start consuming data from a Stream, you only need to provide a config file to point at the correct Kinesis Stream, and functions for initialising the consumer, processing the records, and shutting down the consumer within the skeletons provided.
    - The KCL uses a DynamoDB table to keep track of which records have been processed by the KCL. This ensures that all records are processed "at least once". It is up to the developer to ensure that the program can handle doubly-processed records.
    - The KCL also uses DynamoDB to keep track of other KCL "workers".

70

It automatically shares the available Kinesis Shards across all the workers as equally as possible.

Kinesis Streams Gotchas and Limitations

- Kinesis Streams' shards each only permit 5 reads per second. If you are evenly distributing data across many shards, your read limit for the Stream will remain at 5 reads per second on aggregate, as each consuming application will need to check every single shard for new records. This puts a hard limit on the number of different consuming applications possible per Stream for a given maximum read latency.
  - For example, if you have 5 consuming applications reading data from one Stream with any number of shards, they cannot read with a latency of less than one second, as each of the 5 consumers will need to poll *each shard* every second, reaching the cap of 5 reads per second per shard.
  - This blog post further discusses the performance and limitations of Kinesis in production.
- **Kinesis Streams are not included in the free tier.** Make sure if you do any experimentation with it on a personal account, you shut down the stream or it may run up unexpected costs (~$11 per shard-month.)

Device Farm

Device Farm Basics

- Homepage Developer guide FAQ Pricing
- **Device Farm** is an AWS service that enables mobile app testing on real devices.
- Supports iOS and Android (including Kindle Fire) devices, as well as the mobile web.
- Supports remote device access in order to allow for interactive testing/debugging.

Device Farm Tips

- AWS Mobile blog contains several examples of Device Farm usage for testing.
- Device Farm offers a free trial for users who want to evaluate their service.
- Device Farm offers two pricing models: Paying **per device minute** is useful for small usage levels or for situations where it's hard to predict usage amount. **Unmetered plans** are useful in situations where active usage is expected from the beginning.

Device Farm Gotchas and Limitations

- Devices don't have a SIM card and therefore can't be used for testing SIM card-related features.
- Device Farm supports testing for most popular languages/frameworks, but not for all. An actual list of supported frameworks and languages is presented on this page.
- The API and CLI for Device Farm is quite a low level and may require developing additional tools or scripts on top of it.
- AWS provide several tools and plugins for Device Farm, however, it doesn't cover all cases or platforms. It may require developing specific tools or plugins to support specific requirements.
- In general, Device Farm doesn't have Android devices from Chinese companies like Huawei, Meizu, Lenovo, etc. An actual list of supported devices located here.

IoT

IoT Basics

- Homepage   User guide   FAQ   Pricing
- **IoT** is a platform for allowing clients such as IoT devices or software applications (examples) to communicate with the AWS cloud.
- Clients are also called **devices** (or **things**) and include a wide variety of device types. Roughly there are three categories of device types that interact with IoT services by sending message over an IoT protocol to the IoT Pub/Sub-style message broker, which is called the IoT **Device Gateway**:
    - Send messages only: For example, the AWS IoT Button on an eddystone beacon.
    - Send and receive messages: For example, the Phillips Home Safe Medical Alert device
    - Send, receive, and process messages: For example, a simple processing board, such as a **Raspberry Pi** (quick start guide), or an AWS device, such as Echo or Echo Dot, which are designed to work with the AWS Alexa skills kit (a programmable voice-enabled service from AWS).
- AWS has a useful quick-start (using the Console) and a slide presentation on core topics.
- **IoT terms:**
    - AWS **IoT Things** (metadata for devices in a registry) and can store device state in a JSON document, which is called a **device shadow**. Device metadata can also be stored in **IoT Thing Types**. This aids in device metadata management by allowing for reuse of device description and configuration for more than one device. Note that IoT Thing Types can be deprecated, but not changed — they are immutable.

72

- AWS **IoT Certificates** (device authentication) are the logical association of a unique certificate to the logical representation of a device. This association can be done in the Console. In addition, the public key of the certificate must be copied to the physical device. This covers the authentication of devices to a particular AWS Device Gateway (or message broker).
- AWS **IoT Policies** (device/topic authorization) are JSON files that are associated to one or more AWS IoT certificates. This authorizes associated devices to publish and/or subscribe to messages from one or more MQTT topics.
- AWS **IoT Rules** are SQL-like queries which allows for reuse of some or all device message data, as described in this presentation, which summarizes design patterns with for IoT Rules.
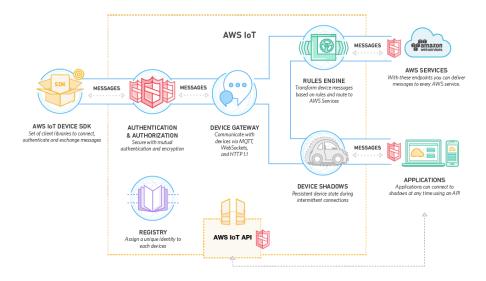- Shown below is a diagram which summarizes the flow of messages between the AWS IoT services:

Figure 1: How AWS IoT Works

IoT Alternatives and Lock-in

- AWS, Microsoft and Google have all introduced IoT-specific sets of cloud services since late 2015. AWS was first, moving their IoT services to general availability in Dec 2015. Microsoft released their set of IoT services for Azure in Feb 2016. Google has only previewed, but not released their IoT services Brillo and Weave.
- Issues of lock-in center around your devices — protocols (for example MQTT, AMQP), message formats (such as, JSON vs. Hex…) and security (certificates).

IoT Tips

- **Getting started with Buttons:** One way to start is to use an **AWS IoT Button**. AWS provides a number of code samples for use with their IoT Button, you can use the AWS IoT console, click the "connect AWS IoT button" link and you'll be taken to the AWS Lambda console. There you fill out your button's serial number to associate it with a Lambda. (As of this writing, AWS IoT buttons are only available for sale in the US.)
- **Connections and protocols:** It is important to understand the details of about the devices you wish to connect to the AWS IoT service, including how you will secure the device connections, the device protocols, and more. Cloud vendors differ significantly in their support for common IoT protocols, such as MQTT, AMQP, XMPP. AWS IoT supports **secure MQTT**, **WebSockets** and **HTTPS**.
- Support for **device security** via certificate processing is a key differentiator in this space. In August 2016, AWS added just-in-time registrations for IoT devices to their services.
- **Combining with other services:** It's common to use other AWS services, such as AWS Lambda, Kinesis and DynamoDB, although this is by no means required. Sample IoT application reference architectures are in this screencast.
- **Testing tools:**
  - To get started, AWS includes a lightweight MQTT client in the AWS IoT console. Here you can create and test sending and receiving messages to and from various MQTT topics.
  - When testing locally, if using MQTT, it may be helpful to download and use the open source Mosquitto broker tool for local testing with devices and/or device simulators
  - Use this MQTT load simulator to test device message load throughout your IoT solution.

IoT Gotchas and Limitations

- **IoT protocols:** It is important to verify the exact type of support for your particular IoT device message protocol. For example, one commonly used IoT protocol is MQTT. Within MQTT there are three possible levels of QoS in MQTT. AWS IoT supports MQTT QoS 0 (fire and forget, or at most once) and QoS 1(at least once, or includes confirmation), but *not* QoS 2 (exactly once, requires 4-step confirmation). This is important in understanding how much code you'll need to write for your particular application message resolution needs. Here is a presentation about the nuances of connecting.
- The ecosystems to match **IAM users or roles** to **IoT policies** and their associated authorized AWS IoT devices are immature. Custom coding to enforce your security requirements is common.
- A common mistake is to misunderstand the importance of IoT **device**

security. It is imperative to associate *each* device with a unique certificate (public key). You can generate your own certificates and upload them to AWS, or you can use AWS generated IoT device certificates. It's best to read and understand AWS's own guidance on this topic.

- There is only one **AWS IoT Gateway** (endpoint) per AWS account. For production scenarios, you'll probably need to set up multiple AWS accounts in order to separate device traffic for development, test and production. It's interesting to note that the Azure IoT Gateway supports configuration of multiple endpoints, so that a single Azure account can be used with separate pub/sub endpoints for development, testing and production

- **Limits:** Be aware of limits, including device message size, type, frequency, and number of AWS IoT rules.

IoT Code Samples

- Simple Beer Service is a surprisingly useful code example using AWS IoT, Lambda, etc.
- IoT-elf offers clean Python sample using the AWS IoT SDK.
- IoT Button projects on Hackster include many different code samples for projects.
- 5 IoT code examples: a device simulator, MQTT sample, just in time registration, truck simulator, prediction data simulator.
- AWS Alexa trivia voice example is a quick-start using Alexa voice capability and Lambda.
- Some Raspberry Pi examples include the Beacon project, Danbo, and GoPiGo.

SES

SES Basics

- [Homepage](https://aws.amazon.com/ses/)   [Documentation](https://aws.amazon.com/doc

- **SES** (or Simple Email Service) is a service that exposes SMTP endpoints for your a

SES Tips

- **Bounce Handling:** Make sure you handle this early enough. Your ability to send em

- **Credentials:** Many developers get confused between [SES credentials](https://docs

SES Gotchas and Limitations

- **Internet Access:** SES SMTP endpoints are on the Internet and will not be accessib

Certificate Manager

Certificate Manager Basics

- Homepage User guide FAQ Pricing
- Use the **Certificate Manager** to manage SSL/TLS certificates in other AWS services.
- Supports importing existing certificates as well as issuing new ones.

Certificate Manager Alternatives and Lock-in

- Certificates issued by the Certificate Manager can't be used outside of the services

Certificate Manager Tips

- **Supported services:** Managed [Load Balancers](#load-balancers) and [CloudFront](#

- During the domain validation process, Certificate Manager will send an email to ever

Certificate Manager Gotchas and Limitations

- In order to use **Certificate Manager** for CloudFront distributions certificate mus

- **IoT** has its [own way](http://docs.aws.amazon.com/iot/latest/developerguide/creat

High Availability

This section covers tips and information on achieving high availability.

High Availability Tips

- AWS offers two levels of redundancy, regions and availability zones (AZs).
- When used correctly, regions and zones do allow for high availability. You may want to use non-AWS providers for larger business risk mitigation (i.e. not tying your company to one vendor), but reliability of AWS across regions is very high.
- **Multiple regions:** Using multiple regions is complex, since it's essentially like managing completely separate infrastructures. It is necessary for business-critical services with the highest levels of redundancy. However, for many applications (like your average consumer startup), deploying extensive redundancy across regions may be overkill.
- The High Scalability Blog has a good guide to help you understand when you need to scale an application to multiple regions.
- **Multiple AZs:** Using AZs wisely is the primary tool for high availability!
  - A typical single-region high availability architecture would be to deploy in two or more availability zones, with load balancing in front, as in this AWS diagram.

- The bulk of outages in AWS services affect one zone only. There have been rare outages affecting multiple zones simultaneously (for example, the great EBS failure of 2011) but in general most customers' outages are due to using only a single AZ for some infrastructure.
- Consequently, design your architecture to minimize the impact of AZ outages, especially single-zone outages.
- Deploy key infrastructure across at least two or three AZs. Replicating a single resource across more than three zones often won't make sense if you have other backup mechanisms in place, like S3 snapshots.
- A second or third AZ should significantly improve availability, but additional reliability of 4 or more AZs may not justify the costs or complexity (unless you have other reasons like capacity or Spot market prices).
- Watch out for **cross-AZ traffic costs**. This can be an unpleasant surprise in architectures with large volume of traffic crossing AZ boundaries.
- Deploy instances evenly across all available AZs, so that only a minimal fraction of your capacity is lost in case of an AZ outage.
- If your architecture has single points of failure, put all of them into a single AZ. This may seem counter-intuitive, but it minimizes the likelihood of any one SPOF to go down on an outage of a single AZ.
- **EBS vs instance storage:** For a number of years, EBSs had a poorer track record for availability than instance storage. For systems where individual instances can be killed and restarted easily, instance storage with sufficient redundancy could give higher availability overall. EBS has improved, and modern instance types (since 2015) are now EBS-only, so this approach, while helpful at one time, may be increasingly archaic.
- Be sure to use and understand CLBs/ALBs appropriately. Many outages are due to not using load balancers, or misunderstanding or misconfiguring them.

High Availability Gotchas and Limitations

- **AZ naming** differs from one customer account to the next. Your "us-west-1a" is not the same as another customer's "us-west-1a" — the letters are assigned to physical AZs randomly per account. This can also be a gotcha if you have multiple AWS accounts.
- **Cross-AZ traffic** is not free. At large scale, the costs add up to a significant amount of money. If possible, optimize your traffic to stay within the same AZ as much as possible.

Billing and Cost Management

Billing and Cost Visibility

- AWS offers a **free tier** of service, that allows very limited usage of resources at no cost. For example, a micro instance and small amount of storage is available for no charge. (If you have an old account but starting fresh, sign up for a new one to qualify for the free tier.) AWS Activate extends this to tens of thousands of dollars of free credits to startups in certain funds or accelerators.
- You can set **billing alerts** to be notified of unexpected costs, such as costs exceeding the free tier. You can set these in a granular way.
- AWS offers Cost Explorer, a tool to get better visibility into costs.
- Unfortunately, the AWS console and billing tools are rarely enough to give good visibility into costs. For large accounts, the AWS billing console can time out or be too slow to use.
- **Tools:**
  - Enable billing reports and install an open source tool to help manage or monitor AWS resource utilization. **Netflix Ice** is probably the first one you should try. Check out docker-ice for a Dockerized version that eases installation.
  - One challenge with Ice is that it doesn't cover amortized cost of reserved instances.
  - Other tools include Security Monkey and Cloud Custodian.
- **Third-party services:** Several companies offer services designed to help you gain insights into expenses or lower your AWS bill, such as OpsClarity, Cloudability, CloudHealth Technologies, and ParkMyCloud. Some of these charge a percentage of your bill, which may be expensive. See the market landscape.
- AWS's Trusted Advisor is another service that can help with cost concerns.
- Don't be shy about asking your account manager for guidance in reducing your bill. It's their job to keep you happily using AWS.
- **Tagging for cost visibility:** As the infrastructure grows, a key part of managing costs is understanding where they lie. It's strongly advisable to tag resources, and as complexity grows, group them effectively. If you set up billing allocation appropriately, you can then get visibility into expenses according to organization, product, individual engineer, or any other way that is helpful.
- If you need to do custom analysis of raw billing data or want to feed it to a third party cost analysis service, enable the detailed billing report feature.
- Multiple Amazon accounts can be linked for billing purposes using the Consolidated Billing feature. Large enterprises may need complex billing structures depending on ownership and approval processes.

AWS Data Transfer Costs

- For deployments that involve significant network traffic, a large fraction of AWS expenses are around data transfer. Furthermore, costs of data transfer, within AZs, within regions, between regions, and into and out of AWS and the internet vary significantly depending on deployment choices.
- Some of the most common gotchas:
    - *AZ-to-AZ traffic:* Note EC2 traffic between AZs is effectively the same as between regions. For example, deploying a Cassandra cluster across AZs is helpful for high availability, but can hurt on network costs.
    - *Using public IPs when not necessary:* If you use an Elastic IP or public IP address of an EC2 instance, you will incur network costs, even if it is accessed locally within the AZ.
- This figure gives an overview:

AWS Data Transfer Costs

EC2 Cost Management

- With EC2, there is a trade-off between engineering effort (more analysis, more tools, more complex architectures) and spend rate on AWS. If your EC2 costs are small, many of the efforts here are not worth the engineering time required to make them work. But once you know your costs will be growing in excess of an engineer's salary, serious investment is often worthwhile.
- Larger instances aren't necessarily priced higher in the spot market – therefore, you should look at the available options and determine which instances will be most cost effective for your jobs. See Bid Advisor.
- **Spot instances:**
    - EC2 Spot instances are a way to get EC2 resources at significant discount — often many times cheaper than standard on-demand prices — if you're willing to accept the possibility that they be terminated with little to no warning.
    - Use Spot instances for potentially very significant discounts whenever you can use resources that may be restarted and don't maintain long-term state.
    - The huge savings that you can get with Spot come at the cost of a significant increase in complexity when provisioning and reasoning about the availability of compute capacity.
    - Amazon maintains Spot prices at a market-driven fluctuating level, based on their inventory of unused capacity. Prices are typically low but can spike very high. See the price history to get a sense for this.
    - You set a bid price high to indicate how high you're willing to pay, but you only pay the going rate, not the bid rate. If the market rate exceeds the bid, your instance may be terminated.
    - Prices are per instance type and per availability zone. The same

instance type may have wildly different price in different zones at the same time. Different instance types can have very different prices, even for similarly powered instance types in the same zone.

– Compare prices across instance types for better deals.
– Use Spot instances whenever possible. Setting a high bid price will assure your machines stay up the vast majority of the time, at a fraction of the price of normal instances.
– Get notified up to two minutes before price-triggered shutdown by polling your Spot instances' metadata.
– Make sure your usage profile works well for Spot before investing heavily in tools to manage a particular configuration.

• **Spot fleet:**
  – You can realize even bigger cost reductions at the same time as improvements to fleet stability relative to regular Spot usage by using Spot fleet to bid on instances across instance types, availability zones, and (through multiple Spot Fleet Requests) regions.
  – Spot fleet targets maintaining a specified (and weighted-by-instance-type) total capacity across a cluster of servers. If the Spot price of one instance type and availability zone combination rises above the weighted bid, it will rotate running instances out and bring up new ones of another type and location up in order to maintain the target capacity without going over target cluster cost.

• **Spot usage best practices:**
  – **Application profiling:**
    * Profile your application to figure out its runtime characteristics. That would help give an understanding of the minimum cpu, memory, disk required. Having this information is critical before you try to optimize spot costs.
    * Once you know the minimum application requirements, instead of resorting to fixed instance types, you can bid across a variety of instance types (that gives you higher chances of getting a spot instance to run your application).E.g., If you know that 4 cpu cores are enough for your job, you can choose any instance type that is equal or above 4 cores and that has the least Spot price based on history. This helps you bid for instances with greater discount (less demand at that point).
  – **Spot price monitoring and intelligence:**
    * Spot Instance prices fluctuate depending on instance types, time of day, region and availability zone. The AWS CLI tools and API allow you to describe Spot price metadata given time, instance type, and region/AZ.
    * Based on history of Spot instance prices, you could potentially build a myriad of algorithms that would help you to pick an instance type in a way that **optimizes cost**, **maximizes availability**, or **offers predictable performance**.
    * You can also track the number of times an instance of certain

type got taken away (out bid) and plot that in graphite to improve your algorithm based on time of day.

– **Spot machine resource utilization:**
  * For running spiky workloads (spark, map reduce jobs) that are schedule based and where failure is non critical, Spot instances become the perfect candidates.
  * The time it takes to satisfy a Spot instance could vary between 2-10 mins depending on the type of instance and availability of machines in that AZ.
  * If you are running an infrastructure with hundreds of jobs of spiky nature, it is advisable to start pooling instances to optimize for cost, performance and most importantly time to acquire an instance.
  * Pooling implies creating and maintaining Spot instances so that they do not get terminated after use. This promotes re-use of Spot instances across jobs. This of course comes with the overhead of lifecycle management.
  * Pooling has its own set of metrics that can be tracked to optimize resource utilization, efficiency and cost.
  * Typical pooling implementations give anywhere between 45-60% cost optimizations and 40% reduction in spot instance creation time.
  * An excellent example of Pooling implementation described by Netflix (part1, part2)

- **Spot management gotchas**
  – **Lifetime:** There is no guarantee for the lifetime of a Spot instance. It is purely based on bidding. If anyone outbids your price, the instance is taken away. Spot is not suitable for time sensitive jobs that have strong SLA. Instances will fail based on demand for Spot at that time. AWS provides a two-minute warning before Amazon EC2 must terminate your Spot instance.
  – **API return data:** - The Spot price API returns Spot prices of varying granularity depending on the time range specified in the api call.E.g If the last 10 min worth of history is requested, the data is more fine grained. If the last 2 day worth of history is requested, the data is more coarser. Do not assume you will get all the data points. There **will** be skipped intervals.
  – **Lifecycle management:** Do not attempt any fancy Spot management unless absolutely necessary. If your entire usage is only a few machines and your cost is acceptable and your failure rate is lower, do not attempt to optimize. The pain for building/maintaining it is not worth just a few hundred dollar savings.

- **Reserved Instances:** allow you to get significant discounts on EC2 compute hours in return for a commitment to pay for instance hours of a specific instance type in a specific AWS region and availability zone for a pre-established time frame (1 or 3 years). Further discounts can be

realized through "partial" or "all upfront" payment options.

- Consider using Reserved Instances when you can predict your longer-term compute needs and need a stronger guarantee of compute availability and continuity than the (typically cheaper) Spot market can provide. However be aware that if your architecture changes your computing needs may change as well so long term contracts can seem attractive but may turn out to be cumbersome.
- There are two types of Reserved Instances - [Standard and Convertible](http://doc
- Instance reservations are not tied to specific EC2 instances - they are applied at the billing level to eligible compute hours as they are consumed across all of the instances in an account.
- **Hourly billing waste:** EC2 instances are billed in instance-hours — rounded up to the nearest full hour! For long-lived instances, this is not a big worry, but for large transient deployments, like EMR jobs or test deployments, this can be a significant expense. Never deploy many instances and terminate them after only a few minutes. In fact, if transient instances are part of your regular processing workflow, you should put in protections or alerts to check for this kind of waste.
- If you have multiple AWS accounts and have configured them to roll charges up to one account using the "Consolidated Billing" feature, you can expect *unused* Reserved Instance hours from one account to be applied to matching (region, availability zone, instance type) compute hours from another account.
- If you have multiple AWS accounts that are linked with Consolidated Billing, plan on using reservations, and want unused reservation capacity to be able to apply to compute hours from other accounts, you'll need to create your instances in the availability zone with the same *name* across accounts. Keep in mind that when you have done this, your instances may not end up in the same *physical* data center across accounts - Amazon shuffles availability zones names across accounts in order to equalize resource utilization.
- Make use of dynamic Auto Scaling, where possible, in order to better match your cluster size (and cost) to the current resource requirements of your service.

Further Reading

This section covers a few unusually useful or "must know about" resources or lists.

- AWS
  - AWS In Plain English: A readable overview of all the AWS services
  - Awesome AWS: A curated list of AWS tools and software
  - AWS Tips I Wish I'd Known Before I Started: A list of tips from Rich Adams
  - AWS Whitepapers: A list of technical AWS whitepapers, covering

topics such as architecture, security and economics.

- Books
  - Amazon Web Services in Action
  - AWS Lambda in Action
  - Serverless Architectures on AWS
  - Serverless Single Page Apps
  - The Terraform Book
  - AWS Scripted 2 book series
  - Amazon Web Services For Dummies
  - AWS System Administration
  - Python and AWS Cookbook
  - Resilience and Reliability on AWS
  - AWS documentation as Kindle ebooks
- General references
  - AWS Well Architected Framework Guide: Amazon's own 56 page guide to operational excellence - guidelines and checklists to validate baseline security, reliability, performance (including high availability) and cost optimization practices.
  - Awesome Microservices: A curated list of tools and technologies for microservice architectures. Worth browsing to learn about popular open source projects.
  - Is it fast yet?: Ilya Grigorik's TLS performance overview
  - High Performance Browser Networking: A full, modern book on web network performance; a presentation on the HTTP/2 portion is here.

Disclaimer

The authors and contributors to this content cannot guarantee the validity of the information found here. Please make sure that you understand that the information provided here is being provided freely, and that no kind of agreement or contract is created between you and any persons associated with this content or project. The authors and contributors do not assume and hereby disclaim any liability to any party for any loss, damage, or disruption caused by errors or omissions in the information contained in, associated with, or linked from this content, whether such errors or omissions result from negligence, accident, or any other cause.

License