Cloud Services: An Overview

Virtualization & Cloud Workshop

UNIVERSITY OF OREGON

Last updated 31st January 2024

NSRC
Network Startup Resource Center

What are "Cloud Services"?

- Ready-made components that you can assemble to build an execution environment for your software
- Note that "cloud workloads" can also run on your own hardware (where you build equivalent components yourself)
  - Packaging up your software in this way can give you flexibility for where to deploy it
- Let's briefly review the main options
  - Different clouds use different terms for broadly the same services: we'll show the AWS terms

Compute Services

- Somewhere to run your software and store your data

Software as a Service (SaaS)

- The whole application is deployed for you and fully managed
- You just need to login (and pay)
- Typically provide web interface and/or APIs
- Examples:
  - Office365 / Google Workspace / GMail
  - Salesforce
  - Telephony and call center platforms
  - Other CRM systems, ticketing systems, project management software…

Infrastructure as a Service (IaaS)

- Traditional server workloads: Bare Metal or Virtual Machines
  - Amazon name: EC2 (Elastic Compute Cloud), Lightsail
  - Each one is called an "instance"
- You need to install and manage the OS and the applications
  - Monitoring, patching and security are all your responsibility
  - Admin login like a regular server, e.g. SSH, RDP
- You need to choose appropriate dimensioning
  - CPUs, RAM, storage
- Different CPU types available (x86_64, ARM)

Containers

- Containers are a way of *running* processes, isolated from each other, on the same server or VM
  - Sharing the same kernel, but isolated via cgroups and namespaces
  - Much more lightweight than separate VMs, but less robust isolation
  - Can share CPU and RAM, but can also have quota limits applied
- OCI containers ("docker containers") are also a way of *packaging and distributing* software
  - Container image contains all the OS dependencies it needs (libraries etc)
  - Fully reproducible way of running applications in dev, test, production
  - Zero install or upgrade: just fire up a fresh container image

Containers (2)

- Container runtime
  - Define what containers you want running, and the platform manages them
  - ECS (Elastic Container Service), EKS (Elastic Kubernetes Service): provision a cluster of underlying EC2 VMs for the containers to run on
  - Fargate ("serverless"): you just define how much resource is needed for each container, and pay for it while it runs
- Container support infrastructure
  - ECR (Elastic Container Registry): where you store the OCI images that you've built, so they can be fetched and run
  - Tools for Continuous Integration / Continuous Deployment (CI/CD)

Block storage

- In a server or VM, the OS boots from a "hard drive", or something which acts like one
  - Linear set of fixed-sized blocks (typ. 512 bytes or 4096 bytes)
- Directly attached storage
  - very fast (e.g. NVMe), but when cloud instance is shutdown, data is lost
- Network-attached block storage
  - Amazon name: EBS (Elastic Block Storage)
  - Persists when instance is shutdown, and can be reattached to a different instance (but only one at a time)
  - Can choose size, storage class, and IOPS performance

Networked file storage

- Application data can also be stored on a remote "file share"
  - Traditional examples: NFS, SMB (Samba)
  - Amazon name: EFS (Elastic File System)
- Acts like a local filesystem with files and directories
  - Open, Read, Write, Seek, Delete etc.
- Multiple instances can have shared read-write access
- Limited access control, may be "all or nothing"
- Not all applications can safely store data on a network filesystem
  - e.g. some databases can only provide data integrity on a block store

Object storage

- Objects are stored and retrieved via HTTPS
  - Amazon name: S3 (Simple Storage Service)
- Organized into "buckets"
  - Access control at the level of buckets and individual objects
  - Authenticated access, including direct access from clients
  - Per-object metadata
- Objects cannot be partially updated in-place, only replaced
- Different storage classes for speed of access, replication
- Storage capacity unlimited (except by your wallet)

Networking

- Connect your components together
  - Amazon name: VPC (Virtual Private Cloud)
- Persistent public IP addresses
  - Amazon name: Elastic IP
- Distributing and filtering inbound traffic to servers
  - ELB (Elastic Load Balancing)
  - API Gateway
  - Cloudfront: content delivery network
- Supporting infrastructure: DNS (route 53), Certificates (ACM)

Network warning!

- Network traffic into the cloud (ingress) is free
- Network traffic out of the cloud (egress) is **chargeable** – and expensive!
  - Amazon: typically US 9¢ per GB
  - Lightsail has inclusive egress allowance, to compete with VPS offerings like Digital Ocean, Vultr, Linode
  - Wasabi (S3 alternative) has no egress charge, but monthly limit
- Designed to lock your data in
- Designed for you to use in-cloud services rather than out-of-cloud

Managed databases

- Saves you having to install and manage a database for your app
  - Fully managed with scheduled backups, multi-site replication etc
- Amazon has many of these!
  - RDS (Relational Database Service) is a real cluster of Postgres, Mysql, Oracle etc.
  - Aurora is an AWS-written SQL database with compatibility layers
  - DynamoDB is an AWS noSQL database
  - DocumentDB is MongoDB-compatible
  - RedShift is a data warehouse (analytics)
  - … and more

Drinking the (Kloud) Kool Aid

- If you're going to write your application *for a specific cloud* then you have more options available
- But it may not be able to run anywhere else

Modular components

- Queuing services
  - SQS (Simple Queuing Service), Kinesis, Firehose
  - MSK (Managed Streaming for Kafka)
- Notifications: SMS, Email, mobile push
  - SNS (Simple Notifications Service)
- Machine learning, media transcoding, …
- These are components that you can integrate into your applications
  - In most cases you have to code to proprietary APIs (except MSK)

Functions as a Service, a.k.a. "Serverless"

- Upload pieces of code which run on demand
  - Amazon name: Lambda, Fargate
  - Can be triggered directly by HTTP (API Gateway), S3 bucket upload, queued messages in SQS, …
- You do not pay while they are not executing: *"scale to zero"*
- They auto-scale up to meet incoming load: *"scale to infinity"*
  - No need to provision or size EC2 instances
- Build connected sequences of tasks
  - Amazon name: Step functions, SWF (Simple Workflow - older)

Benefits and costs

- Benefit: all scaling and OS management overhead is gone
- Benefit: using battle-tested, managed infrastructure
  - Less stuff for you to manage
  - You can focus on your own business requirements
- Cost: you are strongly tied to the cloud – or one cloud
  - There are tools like "serverless framework" which can deploy to multiple clouds, but you'll likely be using cloud-specific APIs in your code
- Total cost for a given amount of compute resource is higher

Application Lifecycle Management

Deployment tools, or "Infrastructure as code"

- You want to reproduce what you have in development, test and production – the full stack of connected components
  - More or less identical, differing in scale and access controls
- Deploy automatically from a well-defined template
  - Amazon specific: Cloud Formation
  - Generic/multi-cloud: Terraform (OpenTofu)
  - Containers: Kubernetes (+ kustomize, helm, jsonnet, cue…)
- Template can be tested and stored in version control, just like code

CI/CD

- Mainly applicable when you are developing your own software
- Continuous Integration (CI): automatically run tests whenever changes are checked in to version control system
- Continuous Deployment (CD): automatically deploy to production when tests pass
  - There are risk reduction strategies, e.g. "canaries"
- Many tools and platforms available
  - eg. Github Actions, Gitlab, Travis, CircleCI, Jenkins, TeamCity, ArgoCD…
  - Some are cloud services, some are self hosted, and some can do either

Logging and monitoring

- You need to know both your resource utilization and how well your application is performing
- Clouds typically provide some sort of dashboard for monitoring CPU usage, storage usage etc
- Collect logs from your application
  - Amazon: CloudWatch, X-Ray, CloudTrail
- Collect metrics (measurements) from your application
  - Amazon: Cloudwatch Metrics, Managed Grafana
- Bills and detailed usage records

Dealing with failures

- Clouds are built of real physical servers, and they DO fail
- Entire data centers can fail (power, storms, fires, fat fingers…)
- AWS is divided into geographic *Regions*, and each region is formed of locally connected data centers called *Availability Zones*
  - Regions are independent of each other (mostly)
  - Availability Zones are closely connected (e.g. EBS volumes are replicated between AZs, and an instance in any AZ can access any EBS volume)
  - Object storage and database replicas can be spread over AZs
  - Inter-AZ traffic is faster and cheaper than Inter-Region traffic

Dealing with failures (2)

- It is up to YOU to architect a system which can cope with failures in a way that works for your use case, e.g.
  - Run multiple instances behind an ELB
  - Run instances in different AZs
  - Repeatedly launch instances from templates (no long-lived instances)
  - Have a complete replica in a different region, and be prepared to swing traffic to it (e.g. via DNS)
- You should continuously test your ability to cope with failures
  - Netflix have a tool called "Chaos Monkey"
  - AWS have a tool called "Fault Injection Service"

Security

- Cloud does not make your OS or your application any more secure than running it locally
  - You still have to do OS and application patching
- Cloud resources are accessible over the Internet, by design
- You must be *very* careful to ensure correct access policies are applied to every resource – and frequently audit them
  - S3 buckets with "read all" permissions have caused many data leaks
  - Many more subtle errors are possible
  - "Doing it right" is hard, and programmers/admins are tempted to do something simple and insecure rather than learning the right way

Security (2)

- Equally scary, the *management APIs* are also accessible over the Internet
- Any compromised admin account could not only read and write any resource, they could destroy the whole account
- Protection of admin credentials is critical (2FA)
- Staff need special training, e.g.
  - Don't generate long-lived API keys on admin accounts
  - Don't check in credentials to source version control
  - Consider completely separate cloud accounts for dev, test and production

Security (3)

- Not surprisingly, security is most easily managed when working within the cloud, rather than outside it
  - For example, EC2 instances or Lambda functions can assume "roles" which in turn give them rights to access other AWS resources, which you define in "policies"
  - This means you don't need to hard-code credentials within the applications themselves, as long as you use these mechanisms correctly
- There are features for encrypting data at rest and in transit, and managing the keys (Amazon: KMS)
- Learning all this is a big investment, but essential

Cost control

- With your own servers, you pay a fixed cost for hosting, until you run out of capacity and need to install more
- With public cloud, you pay for everything you've turned on
- If you turn something on and forget to turn it off, you keep paying indefinitely
- Very easy to end up with huge bills
- Tag every single resource with its responsible business owner, and make the costs come out of that owner's budget
- Frequent audits, and switch off anything unknown

Summary

- There are many pieces to learn, and choices to make
- Which you use depends on whether you are deploying already-written applications, or writing your own software
- We will look at some of them in more detail later in this workshop