Platform as a Service (PaaS)

COSC349—Cloud Computing Architecture

David Eyers

Learning objectives

- Define **Platform as a Service** (PaaS)
- Contrast **PaaS** with **IaaS** (and SaaS)
- Indicate good and bad points about PaaS
- Sketch how an application might be **deployed** using a given PaaS platform
- Explain how Docker and other **container technology** has affected PaaS offerings

PaaS—Platform as a Service

- PaaS is the middle ground between IaaS and SaaS
  - You do not manage the VM infrastructure directly (that's IaaS)
  - However you can't just use application software (that's SaaS)
  - Aimed at **use by software developers**
- 'Platform' is a fairly broad and imprecise term
  - One view is that you build your apps sharing tools that your cloud provider would use to manage their cloud platform
- Cloud provider will see **your software components**
  - In IaaS just sees VMs, and not what role they are performing

Benefits and disadvantages of PaaS

- Focus on **your application logic**, not managing VMs
  - Just get the cloud environment, such as APIs to work with
  - Cloud provider can **leverage economies of scale**

- Disadvantages: potentially **get lock-in**
  - More likely API is tied to specific software
    - Although mature interchange languages like SQL mitigate this
  - **Lack of flexibility**: public PaaS isn't necessarily very extensible
    - Also don't have complete control over the cloud's services

PaaS examples emerged soon after IaaS

- **Heroku** (since 2007) provided cloud hosting of Ruby
  - Was PaaS, since you deployed your Ruby source code
  - Like many popular PaaS offerings, it is hosted on Amazon EC2
- **Google App Engine** (2008)
  - Google already had scalable APIs for their own software
  - App Engine was a way to turn that into a service for sale
- **RedHat OpenShift** (2011)—closed then open source…
  - Sought to effect paradigm shift: scalable components (v2)
- **VMware Cloud Foundry** (2011)—always open source

Heroku

- Ruby on Rails (2004) promoted Ruby for web coding
  - popularised model-view-controller; usually web+database
  - Ruby's portability is quite good, *e.g.*, it's a high-level language
- Deploying code to Heroku typically **done using git**
  - Pushing commits to Heroku causes deployment of your code
- Language-focused clouds **don't have to be Ruby**
  - Now also Node.js, Clojure, Scala, PHP, Python, Go, Java, …
- Other deployment methods added: **Dropbox**; an **API**
- HTTP-focused web accessibility (*e.g.*, web and **REST**)

Google App Engine (GAE)

- Lots of development language options:
  - *e.g.*, Java, Python, Go, PHP, Node.js, …
- Limitations in terms of software behaviour
  - Code can only **react to HTTP requests** (including self-requests)
- Database: originally Google Cloud Datastore
  - Now also Google's **Cloud SQL**: direct legacy SQL support
- Overall makes coding easy, but limited in form
- Lock-in concerns mitigated (?) by FOSS AppScale, *etc.*

RedHat OpenShift v1 and v2

- Applications used ‘gears’ to do their computing
  - Gears used namespaces, cgroups and SELinux for isolation
  - Free-tier allowed three non-scalable gears (until platform EOL)
    - I hosted a test Drupal website and an Etherpad server…

- Notion of ‘cartridges’ that can be combined in a gear
  - Language cartridges such as Ruby on Rails
  - Database cartridges such as MySQL or MongoDB
- Cartridges auto-interconnected, e.g., Rails + MySQL

RedHat OpenShift version 3

- Gears turned into **Docker containers**
- Orchestration of containers uses **Kubernetes**
  - (We'll discuss orchestration later…)
  - OpenShift 2 had a custom broker to manage multi-gear apps
- Images are mapped 1:1 to containers
  - OpenShift 2 cartridges could be loaded N:1 into a gear
- OpenShift 3 uses images like any other Docker client
  - OpenShift 2 required a code repository within OpenShift itself

Cloud Foundry

- Started within VMware—**open source** throughout
  - Targets multiple execution platforms
    - e.g., private clusters running VMware vSphere, OpenStack
    - All the IaaS cloud providers we've discussed

- Cloud Foundry supports software '**lifecycles**'
  - Buildpack lifecycle: Java; JavaScript; Ruby; Python; PHP; Go; notably adds .NET and .NET Core
  - Docker containers can be run in a different type of lifecycle

PaaS and containers?

- Containers rose to prominence after PaaS began
  - RedHat OpenShift redesigned itself for Docker + Kubernetes
- Amazon ECS provides two container solutions
  - **EC2 launch type** can help manage a cluster of VMs
    - Essentially is assisted IaaS: you specify container server EC2 types
  - **Amazon Fargate type** accepts container images directly
    - No management of VMs, so much more PaaS-like
- Google Kubernetes Engine
  - Uses Google Compute Engine nodes as workers

Typical services provided by PaaS

- **Language** runtime
  - Possibly as a framework, *e.g.*, rake rather than just Ruby
- **Database**—your PL usually doesn't include a DB
- **Load balancing** and **autoscaling** layer

- While AWS is IaaS-focused, it provides many PaaS tools
  - Elastic Load Balancer works with HTTP and other protocols
  - Amazon Relational Database Service
  - Offerings like Elastic MapReduce (EMR)—managed Hadoop

AWS database offerings

- We'll focus on relational databases—in common use
  - Amazon provides many non-relational databases too
- DIY: allocate an **EC2 instance** and install a database
  - You can install whatever you want …
    - but patching, scaling and backup/restore are your problem
- **Amazon Relational Database Service** (RDS)
  - Choose: PostgreSQL, MySQL, MariaDB, Oracle, SQL Server…
    - Patching, scaling and backup/restore are Amazon's problem
- **Amazon Aurora**: choose PostgreSQL or MySQL variant

Amazon Aurora

- Amazon realised MySQL on EC2 had **too many layers**
  - MySQL optimising file access on disk—opaque to Amazon
- But MySQL has pluggable datable storage engines
  - In Aurora, Amazon switches in their **own database engine**
  - All data has 6 replicas across 3 availability zones
  - Database is backed up continuously to S3
  - Performance+reliability boost is Amazon-specific: is this lock in?

- Amazon later reengineered PostgreSQL in a similar way