Orchestration and Infrastructure as Code

COSC349—Cloud Computing Architecture

David Eyers

Learning objectives

- Define **orchestration** as relevant to cloud computing
  - including **provisioning** of virtual servers and infrastructure
  - **software configuration** of running cloud virtual servers
- Define '**infrastructure as code**' (IAC)
- Contrast **declarative/imperative** software configuration
- Describe benefits of **immutable** VM deployment

Defining orchestration

- **Automated coordination of computer systems**
  - Deployment and configuration
  - Interconnection and management
  - Monitoring often included, to inform management actions

- Vast, growing set of good solutions, many open source
  - **Machine focused**: *e.g.*, Puppet, Terraform, Ansible, Salt, Chef…
  - **Cloud-based**: *e.g.*, AWS CloudFormation, Terraform
  - **Container clusters**: *e.g.*, Kubernetes

FYI: Choreography similar to orchestration

- **Orchestration** typically involves central control
  - *i.e.*, a coordinator has code to instruct components
- **Choreography** often involves distributed operation
  - … but also about management of computer systems
  - *e.g.*, the protocols and rules between specific services
- **W3C Web Services** technologies define standards
  - … but cloud evolved services independently from W3C WS

Machine-focused configuration tools

- Common goal: target machine reaches target state
  - different paradigms: **declarative** versus **imperative**
- Different extents of coverage
  - **Config. management**: install & manage software; OS assumed
  - **Provisioning**: may set up machine from (virtual) bare-metal
- Different types of presence on target machine
  - *e.g.*, run **persistent agent** vs. gather **information on demand**

Orchestration in the cloud

- Usually need to both **provision** and **configure** VMs
  - Also need to configure services via API: *e.g.*, storage, and …
  - Set up virtual networking: load balancing, firewall security, …

- Different **possible life-cycles** for orchestration's reach
  - *e.g.*, making app-specific AMI vs. use Linux AMI and configure
    - Choose based on frequency of software change on the VM image
  - **Need to bootstrap** cloud access for tools from somewhere
    - But also IAM? Also networks? What is needed depends on context

Container orchestration

- **Containers are convenient units** for cloud deployment
  - As seen previously: combines OS with particular app. function
  - Storage and network configuration specified explicitly
  - Contrast VMs, where configuration has to be done on 'inside'
- **Orchestration of containers** involves:
  - Keeping quotas of active containers of different types
    - (*e.g.*, recover from failure of containers; scale as needed)
  - Managing inter-networking of containers
  - Ensuring that disk layers are available where needed

Multi-cloud orchestration

- Most large organisations use many cloud providers: *e.g.*,
  - explicit strategy to **protect against vendor lock-in**;
  - **resilience to failures** within one cloud provider;
  - and/or consequence of non-coordinated decisions in organisation
- Services emerging that **manage multiple+hybrid clouds**
  - *e.g.*, Scalr applies policy controls across all cloud resources
- VM / container support common across cloud providers
  - but specifics of security and network configuration will be different

Declarative configuration management

- Declarative tools specify the **desired target state**
  - The means to reach target state is up to the config. engine
    - (Embodies declarative paradigm: *e.g.*, SQL in DBs, Prolog in PLs, …)
  - Can take **corrective action** to react to drift in machine's state
- State specification will be a **domain specific language**
  - (There is no general-purpose declarative mechanism)
- Common FOSS tools with large user communities:
  - Puppet, Terraform, CFEngine (was used in CS Labs), SaltStack

Imperative configuration management

- Also termed 'procedural', *i.e.*, **specifying steps to run**:
  - usually chunks of code in conf. systems' authors' favourite PL
- Common tools: Ansible (Py), Chef (Ruby), SaltStack

- Can write imperative code to have declarative effect
  - **Idempotency**—repeated run of code has no additional effect
  - **Dependencies**—code ensures good order of operations
  - Although just because it's possible doesn't make it a good idea:
    - *e.g.*, declarative engines will do scheduling of dependencies for you

Declarative vs. imperative paradigms

- Imperative suits migrating from **custom scripting**
- Declarative can well suit **evolution**:
  - Want 10 EC2 instances? Ansible and Terraform syntax similar
  - Now say you want to increase to 15 EC2 VM instances
    - Declarative—update spec to 15; **engine sees need to add 5 VMs**
    - Imperative… 15 new instances? Or create 5… but have to ensure that scripting to setup from clean creates 15? May be messy…

- Different approaches may best suit different job roles:
  - *e.g.*, declarative for sysadmin; imperative for devop coder?

Immutable—no configuration management

- Configuration management **patches running servers**
  - **Configuration drift** in large systems: slowly get OS differences…
- Instead, deploy & upgrade **whole OS+app atomically**
  - Possible now that redeployment is cheap, *e.g.*, containers
  - Application mutability through **blue/green evolution**:
    - try blue version on set of new servers, if OK, retire green servers
- Netflix: technician touched server? Reimage it soon!

Infrastructure as code (IAC)

- IAC covers configuration management & provisioning
  - also involves avoiding hardware configuration (*e.g.*, switches)
  - goal is for **complete automation**, from **machine readable files**
  - works both for cloud, cluster and single server operation
- **Cost reduction** for organisations in terms of staff
  - Focus on business needs rather than device management
- DevOp-style: if Git repository defines app, you have IAC
  - Continuous integration pipelines often used to deploy