**Final DevOps Study Guide**

**Continuous Integration**

Example of principle of if it is difficult or error prone do it more often

Source control is the only thing essential for this

You can fail a build for reasons other than bad compile or failed unit tests

Continuous Delivery > Continuous Integration

Continuous Integration = automated compilation + automated unit tests

Best practices of CI:

1. Every developer is responsible for fixing issues in their changes
2. Wait for tests to run locally
3. Never go home on a broken build
4. Never comment out failing tests
5. Don’t commit on a broken build

**Infrastructure**

Environment – all resources that the app needs to work and their config

Infrastructure – all environments + services that support them

1. All networking, computing and storage hardware
2. Middleware (web servers, etc.)
3. External integration points
4. Operating system and its configuration

NOT the application itself and its configuration

Infrastructure principles

1. Version controlled configuration
2. Autonomic, automatic correction to desired state
3. Observable thought instrumentation and monitoring

Changes logged, audited, single ticketing system, metrics, production-like environment, version control, automated tests

**Virtualization**

Virtualization – simulate an entire computer, in order to run multiple instances on a single physical machine

More efficient use of computing resources and divide physical hardware into multiple VM

Type 1 Virtualization = VM OS ->VM -> Hypervisor -> Hardware

Type 2 Virtualization = VM OS ->VM -> Hypervisor -> OS -> Hardware

Type 1 is always faster used for cloud, type 2 is desktop

Benefits: fast changes in equipment, standardization, easy to maintain images, easy to simulate abnormal system conditions

Image = virtual machine template

**Cloud Computing**

Cloud computing – On demand network access to a shared pool of configurable computing resources

Pay as you go = Public Cloud, service is called Utility Computing

Enterprise IT(nothing) -> Infrastructure as a service(everything but servers) -> Platform as a service (everything but app) -> Software as a service (everything)

3 Types of clouds = Public (provider for public), Private (single organization), Hybrid (both)

Elasticity = Ability for an application to use more of the resources of each virtual machine on which it is installed

When to use cloud?

1. Demand varies
2. Demand unknown
3. Demand can leverage parallelism

12-factor applications is a set of guidelines for building software for cloud

(scalable, pay-as-go, no bare metal, billing models, designed for cloud, cost of hardware as important as performance)

**Infrastructure Automation**

Infrastructure configuration as code

No one should have access to any boxes except operations team

Any change should be automated and stored in source control

**Vagrant** – provisions VMs from images (boxes) (from external repository box catalog)

Providers (runtime environment in which VM is executed) vs Provisioners (methods we use to make custom changes to base image)

Vagrant Up start the VM called environment

Vagrant can synch files between the host OS and the guest VM

You can map ports between host to guest (forwarding)

Bootstrap.sh – execute scripts to configure the VM once it has been started (provisioning)

(by default only run once when building the box, but they can be configured to run “always”)

**Chef** – Configuration management tool, resource-based

Cookbook- manage recipes, single responsibility principle (all recipes manage a single service or app)

Repices – collections of related resources that describe a particular configuration

**Terraform –** software for automating creation and provisioning of cloud based resources (avoids directly interacting with vendor specific CLIs and SDKs)

**Containerization**

Container = Lightweight virtual machine

\*VM are effectively machines but Docker containers manifest as processes

Just like real shipping containers – allows user to encapsulate payload and run it in any hardware

Benefits

1. Build once run anywhere
2. Effectively sandboxed
3. Container promotions across environments
4. Much more lightweight than VMs

They run like VM but: Container-> Docker Engine -> OS -> Infrastructure

There is no guest OS, so much smaller, start faster and simpler, just a process in the main computer’s OS

Docker Registry – similar to artifact repos, hold Docker images, can be Private or Piblic. (Docker Hub, like Github but for Docker images)

Base image + layers of actions defined in the Dockerfile

Dockerfile Commands

FROM – Identifies Docker image on which new image will be based

COPY – Copies a single file to the image

ADD – Copies an archive file to the container image and unpacks it

ENTRYPOINT – Executes the specified command when the container is executed using “docker run”

RUN – Executes the specified command when the container is being built using “docker build”

Docker image tags

1. They are used by “docker push” to upload an image and associated layers to a docker registry
2. Tags are used to identify versions of the image
3. Docker tags are needed because they contain the URL’s for the target registry and shows version

Docker compose: Allows you to define a docker-compose.yml file and manage multiple containers, but you can do practically the same with normal docker, just more tedious

**Testing**

Test doubles allow us to control the behavior and perform better testing:

• Dummy objects just satisfy argument lists.

• Fake objects have actual implementations, but are not suited to production.

• Stubs return pre-fabricated answers to requests.

• Spies are stubs that record information about the calls that were made to it.

• Mocks are configured with expectations that define the expected interaction with the object.

(specially useful to not have to rely on external systems for testing)

**Calculating Server Hours**

Needed hours = average

Actually provisioned = peak

Needed Hours \* hourly cost \* 365 = yearly cost for cloud

Actual Hours \* hourly cost \* 365 = yearly cost data center