NYU TANDON SCHOOL OF ENGINEERING

Infrastructure Engineering

Intro to Computer Science (Spring 2021)

Prashant Patel
03/26/2020

Table of Content

*

About Me

- Computer Science Graduate from NYU Tandon School of Engineering (2016 – 2018)
- IBM
  - Advisory Software Engineer (2018 – 2020)
- Amazon Web Services
  - Software Development Engineer (2020 – Present)
- LinkedIn : https://www.linkedin.com/in/prashant182/

*

NYU TANDON SCHOOL OF ENGINEERING

Introduction to Infrastructure Engineering

What is Infrastructure Engineering?

Three main pilers of infrastructure engineering

1.Storage

2.Compute

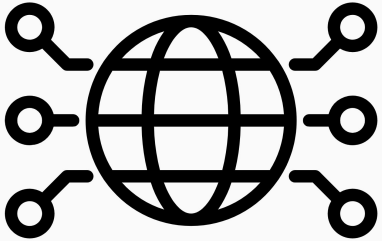
3.Network

Operations with Infrastructure

- Provisioning
- Deprovisioning
- Scaling
- Descaling
- Upgrading

Storage

Compute



Network

On Prem vs. Cloud Computing

On Prem
- Virtualization Stack (VMWare)
- Storage Devices
- Network Switches

Cloud Computing
- Elastic Computing Instances
- Functions as a Service (i.e Lambda)
- Storage as a Service (i.e. S3, Ceph, Rook.io)
- Network as a Service (i.e. VPC, Virtual Sockets)
- Containers as a Service (i.e. Kubernetes, Docker)

VS

aws

Google Cloud Platform

Azure

IBM Cloud

*

On Prem

Pros	Cons
• In some industries it provides more security of data than the counter part. • Complete control over all aspects of maintenance	• Single Point of Failure • Proving to be more costly • Slower Technology Adaptation • Limited choice of options for deployment / infrastructure.

vs

aws

Google Cloud Platform

Azure

IBM Cloud

*

Cloud Computing

Pros	Cons
• Much cheaper compared to counterpart • Wide variety of infrastructure components to choose from • Durability and on demand scaling is backed in. • For developers focus is more on code than the infrastructure.	• Security of data can be of challenge if cloud is compromised. • Vendor lock in makes migration harder • Steep learning curve for platform specific APIs for developers. • Cost / operation ratio may not work in favor with scale.

VS

aws

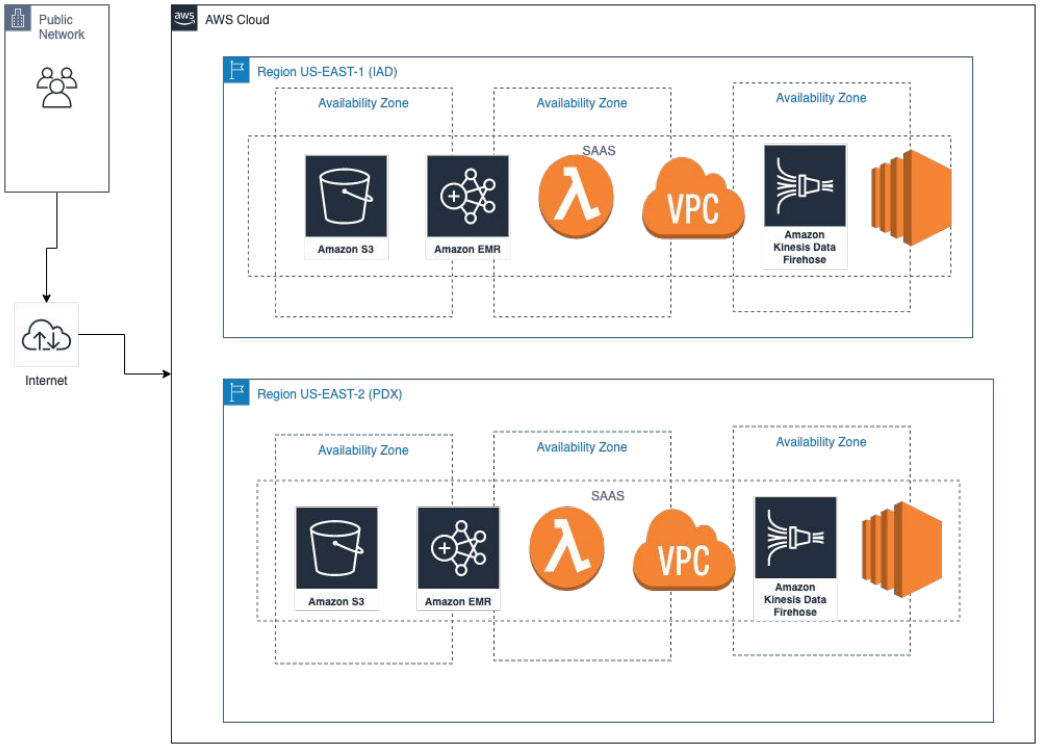
Google Cloud Platform

Azure

IBM Cloud

*

Cloud Computing



aws

Google Cloud Platform

Azure

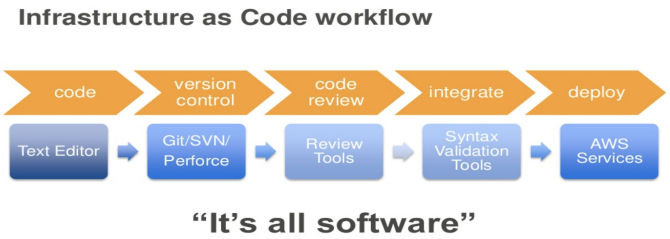
IBM Cloud

*

Infrastructure-as-Code (IaC)

- Infrastructure as Code is a practice by where traditional infrastructure management techniques are supplemented and often replaced by using coding based tools and software development practices.
- Moving to cloud based model adds new possibility on how infrastructure is seen.
- Infrastructure component can be provisioned / deprovisioned in seconds.
- Scaling can be achieved without capacity planning.
- Most of the cloud providers vend APIs for infrastructure lifecycle. APIs open doors for lot of tooling and automation.
- All the coding practices and standers can be applied. Linting, Version Control, Code Reviews, Performance optimization etc.
- Example snippets.

```
//Sudo Code for a demo cloud like env.
Database DB = new Database(us-east-1);
Lambda lb = new Lambda(us-east-2);
lb.setTrigger(apiGateway)
```



*

Software-As-Service (SaaS)

A software distribution model in which applications are hosted by a vendor or a service provider and made available over a private or public network.

In SaaS model software is deployed as a hosted service and accessed over the specified network interface, as opposed to ”On Premise”

Packaged Software	Software As a Service
• Upto the customer to maintain the lifecycle (install, upgrade, remove) • Typically a one time fee association with this sort of software. Not designed to scale with the proportion of users that simultaneously access it. • Ideal for onprem deployment	• Designed from the outset up for the delivery as network based service. • Designed to run multitude of different users in a single deployment. • Lifecycle is maintained by the software developer, abstracted from consumers.

*

Who gets to do this?

- SRE (Site Reliability Engineer)
- DevOps Engineer
- SDE (Software Development Engineer)
- Systems Engineer

*

DevOps

Key principals for an “Infrastructure Engineer” to live by

- Reduce Organization Silos
- Accept Failure as normal
- Implement Gradual Change (Deploy with confidence)
- Leverage Tooling and Automation
- Measure Everything

*

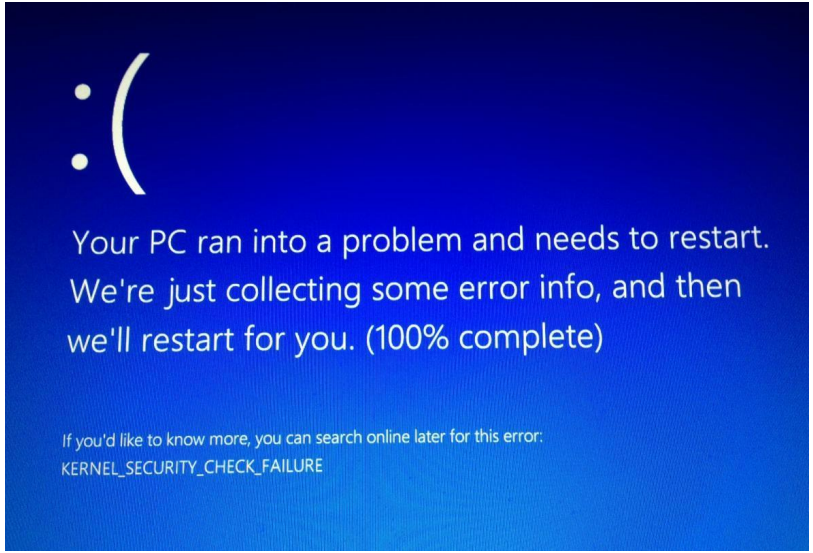
Reduce Organization Silos

- Promote the reusability of the components
- Create common infrastructure patterns
- Share the Failures / Success stories, learn from the mistakes of others
- Common patterns can help with the common understanding of problems and allow larger audience to contribute
- SLO (Service Level Objectives) – Objective that your team must meet.
- SLI (Service Level Indicators) - The real numbers on the performance of Service
- SLA (Service Level Agreement) –The agreement you make to your client.

*

Accept Failure as normal

- Failures happen and there is no real way to avoid them. Learn to live with failures.
- Failures could be due to Bug in the code, Hardware failure, human mistakes, dependency degradation etc.
- Failure mitigation strategies can help.
  - Caching
  - Redundancies
  - Graceful termination
- Make sure failures are properly detected.
- Always prioritize mitigation over the resolution.
- Learn from the mistakes share it with other orgs



*

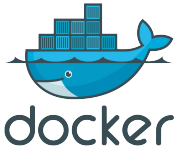
Implement Gradual Change

- This is generic to all the software development practices. Not specific to SRE.
- Small changes are easy to detect / understand. They are easy to rollback to.
- Small changes help in reducing the integration cost.
- Scenario : DataBase migration and scale to keep up with the incoming traffic.
  - You might want to migrate the database first
  - You might want to verify the migration
  - Once verified database can be scaled.
- Small changes tell a greater story about the product journey and it’s lifecycle.
- IaC is a code too. It applies everywhere.

*

Leverage tooling and automation

- Developer tooling is probably one of the most important and fundamental principals
- There are so many tools, Docker, Python, Ansible, Jenkins, Terraform, Cloud Specific SDK.
- Automation is always better than manual human interaction. It's auditable, testable, scalable, reliable.
- Allows developers to spend more time on business problem than to do manual tedious tasks.
- Have automated pipelines and processes for
  - Continuous delivery (CD)
  - Continuous Integration (CI)
  - System Monitoring
  - Alerting
  - Orchestration
  - Change Management
  - Release Management. Etc.



ANSIBLE

*

Measure Everything.

- Metrics are literally the key to success.
- Data driven process allows for better decision making.
- Every change in the system can be evaluated using various metrics.
- Metrics allows you to be ready for the peak. Helps with Auto scale in many instances
- Metrics followed by Alarms is the way to go.
- i.e. For a typical distributes system you can measure
  - Average Latency
  - Number of calls on a given minutes
  - Number of Faults
  - Number of new consumers
  - Peak traffic time

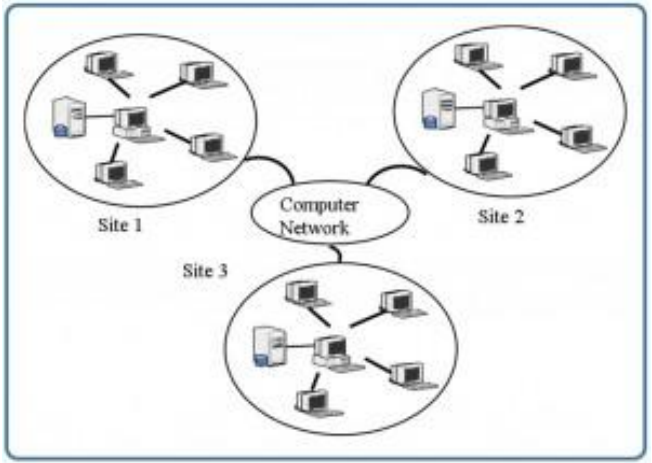
*

NYU TANDON SCHOOL OF ENGINEERING

The Case of Distributed Systems

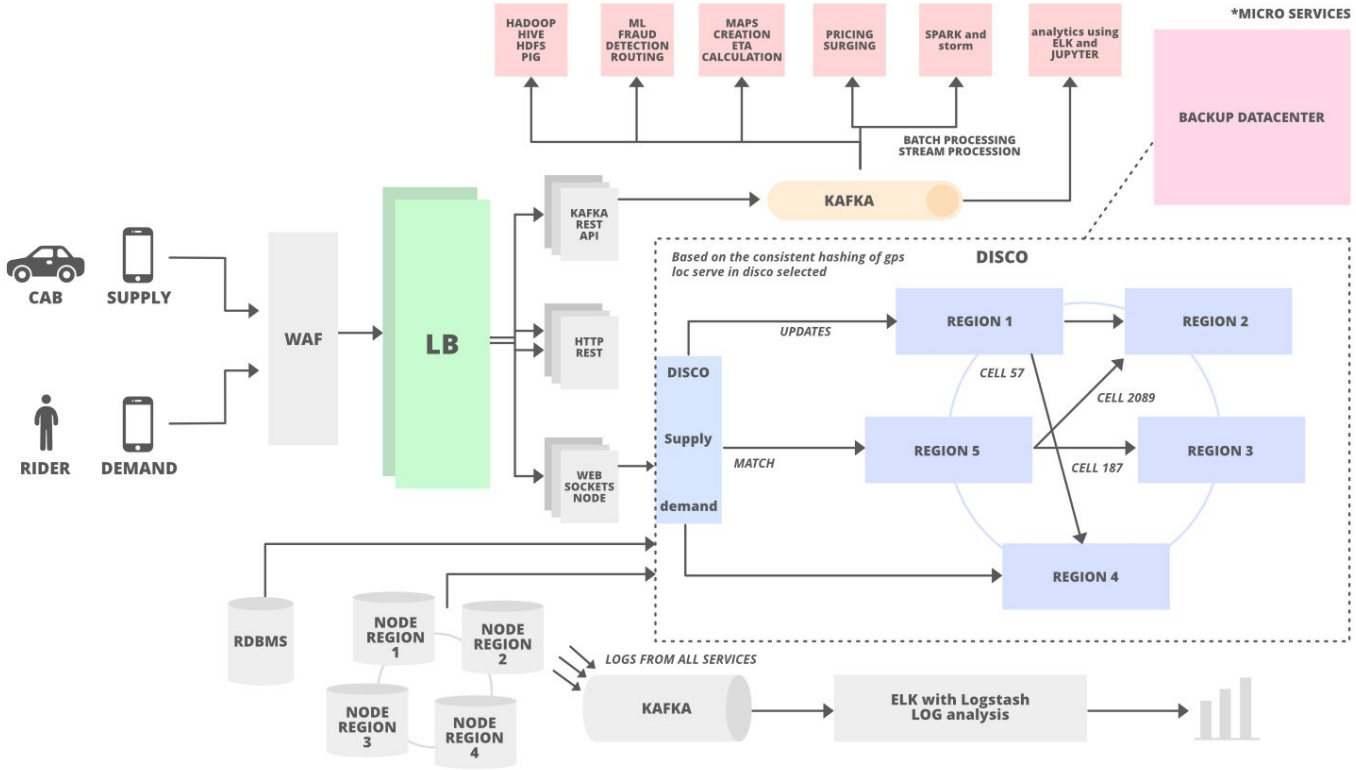
Distributed Systems

- Textbook definition “A distributed system is a collection of independent computers, interconnected via network and capable on collaborating on a task”



*

Case Study : “Uber”

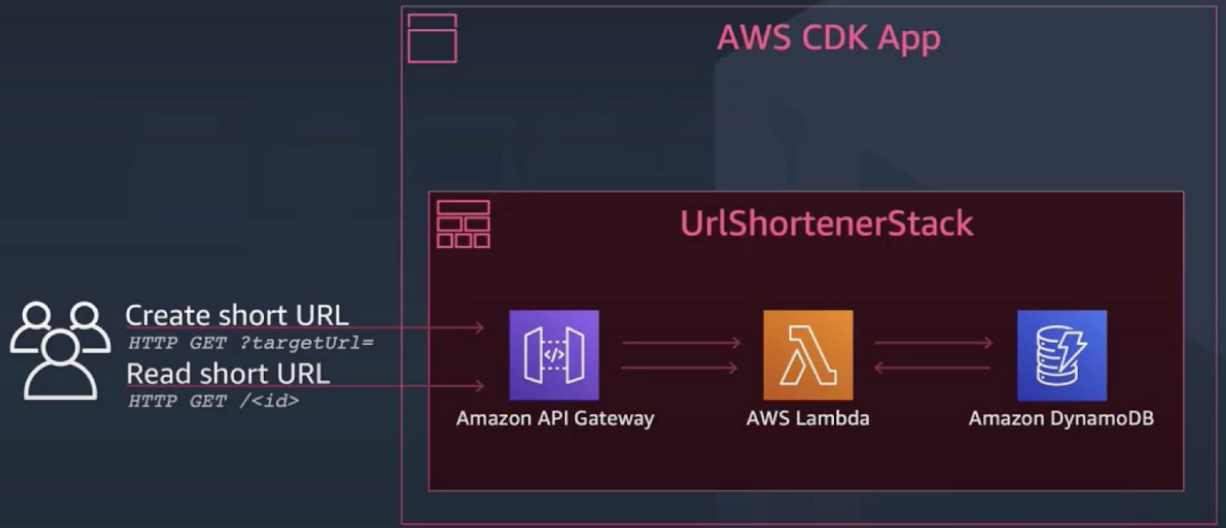


NYU TANDON SCHOOL OF ENGINEERING

AWS Lab

URL Shortener

URL Shortener Architecture



aws