Microservices & Kubernetes

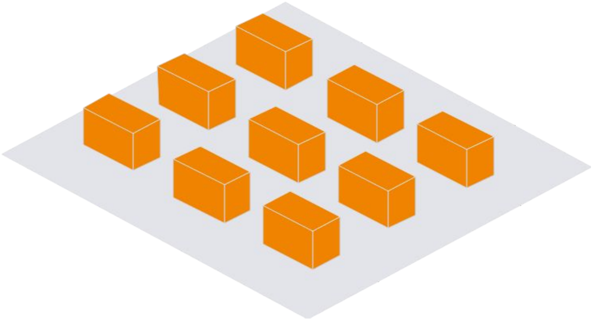
Veasna Sreng
sreng.veasna@gmail.com
@veasnaitc
Institute of Technology of Cambodia, 2010
Freelancer

Topics

- Microservices (MSA)
- Kubernetes & Docker
- Vault
- Example
- Go
- Overall Q/A
- Basic Demo

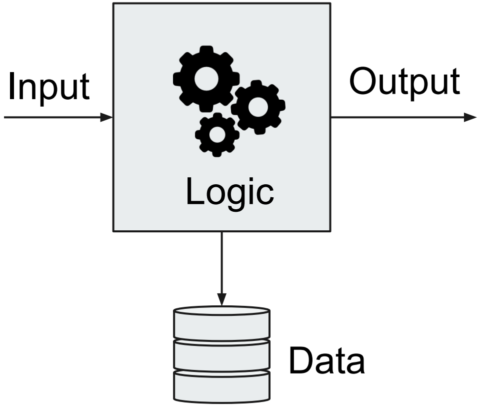
Microservices

- Modular
  - A small application that handle one or more business logic
- Lightweight
  - Simple to maintain
  - Simple to deploy, no effect to the other service
- Service oriented
  - Modular talk to each other over network



Anatomy of Microservice

- Interfaces
  - Known as Restful API
- Logic
  - Application Logic / Business logic
- Data
  - Dataset and modeling



Architectural & Infrastructure

Microservices make infrastructure and architecture more complicate than usual

- Service Discovery
- Image Registry
- API Gateway
- Service Load Balancer
- Global Load Balancer & Router

Service Discovery

Service discovery is a providing the ability to discover the availability of the other microservice.

- Registration microservice
- Availability of microservice

DNS is a part of service discovery.

Image Registry

Image is registry is a storage that responsible to store all deployed microservice images

- Pull images
- Push images
- Delete images

It’s optional, however it is needed for the sake of continuous integration and deployment

API Gateway (1)

A centralized traffic connection which provided more functionality than just a proxy

- Responsible for authorization & Authentication
- Data collection analytics
- Proxy
- Logging & Monitoring
- Caching
- Transforming
- Load Balancing
- More

It’s optional !

API Gateway (2)

Existing API Gateway

- Kong - https://getkong.org/ - Available as ***open source*** and ***enterprise***, Build on top of Nginx OSS
- Tyk.io - https://tyk.io/ - Available as ***open source on-premise*** and 3 different prices include ***enterprise***, Build from scratch with GO
- APIGee - https://apigee.com/about/cp/api-gateway - Paid only
- Zuul - https://github.com/Netflix/zuul/wiki - Available as ***open source***, Build from scratch with Java
- Gravitee.io - https://gravitee.io/ - Available as ***open source***, Build from scratch with Java
- Nginx Plus, paid only

Service Load Balancer

As each microservice will handle a logic or group functionality, a single microservice may not be enough to handle a hug request, that mean we required to run multiple microservice as a group of cluster.

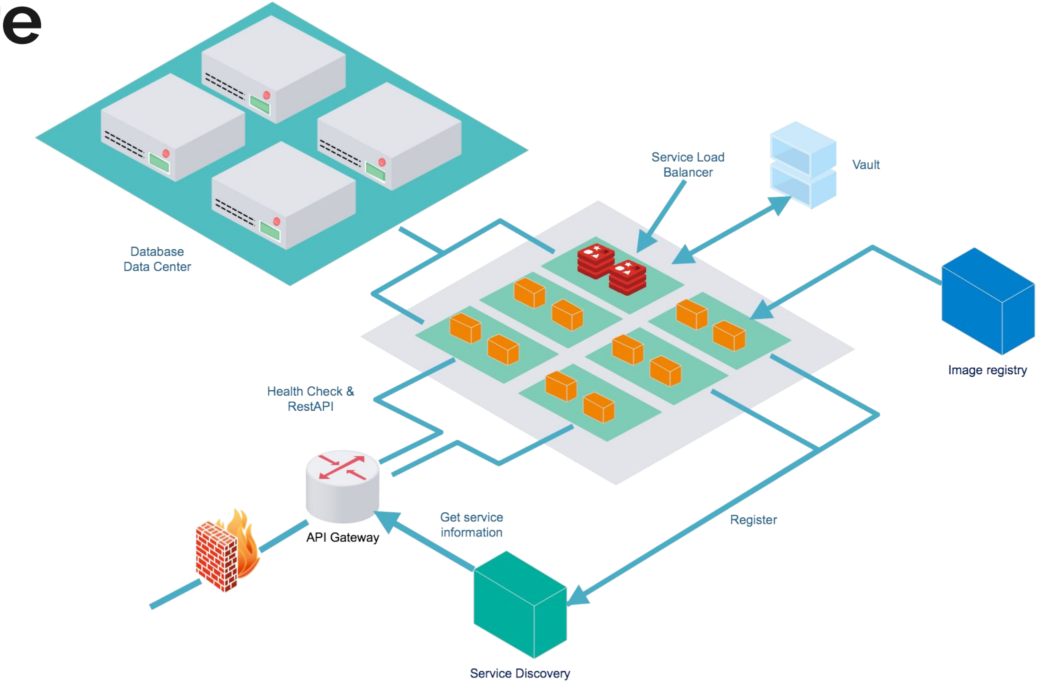
- Similar to service discovery but instead of all microservice, it only responsible for a microservice
- Register themself to Service Discovery

Global Load Balancer & Router

Similar to nginx, Global Load Balancer & Router is responsible to manage the load balancing of the traffic and routing the traffic to each microservice.

- Determine the availability of microservice or a cluster of microservice
- Manage load balance
- Routing the traffic

Overall Architecture & Infrastructure



Advantages Microservices

- Each microservice is independent which and enable deployment much simpler
- Applicable for Agile development
- Each service is simple to maintain, small code base, simpler for the new staff
- Fast update and deployment allow company to release or fix issue faster than monolithic architecture

Downside of Microservices

- Because each microservice has it owns business logic, so document on each microservice is almost a must
- Communication of team is overhead
- Network Security and Vulnerabilities
- Increase network traffic and latency as each microservice need to talk to each other over network
- Testing might be a troublesome
- Monitoring the application is much more expansive than monolithic architecture
- Infrastructure and Architecture is complex

Best practice for the downside (1)

Documentation, it’s troublesome when come to documentation as each RestAPI must documented and even include version control, this leading to developer must take time to write down or update the document to reflect to the new update in the code. To solve this problem, we use

- OpenAPI (https://www.openapis.org/)
- Swagger (https://swagger.io/)

Common Pattern

- Write documentation first then generate the code and finally implement logic (**Recommended**)
- Write the code first, add comment to your code and then generate documentation

Best practice for the downside (2)

Other than using OpenAPI and Swagger, we can use GRPC a common and a better for communication between microservice, https://grpc.io/ as it has more advantages than just swagger or openAPI alone. It also improve network latency as well.

- Using protobuf make encoding and decoding much more faster than raw json
- Code generate almost perfect for both server and client
- Enable multi-language microservice
- Support Flatbuffer as data format (https://google.github.io/flatbuffers/)
- There are a lot of tools out there to convert protobuf schema to swagger and openAPI, official request to support GRPC https://github.com/OAI/OpenAPI-Specification/issues/801

Alternative to GRPC: Message Pack, Cap’n Pro

Best practice for the downside (3)

- As testing is a lot harder for microservices, a unit test or tdd with bdd is a must.
- Monitoring application might be expensive but as long as we use API Gateway, it will solve most of the fundamental issue.

Complexity of Microservices

- Even though each microservice is small and independent but create cluster from multiple of those microservice instance is somewhat painful
- Running a cluster of microservices it a lot harder as we need to consider available of the resources (CPU, Memory, ...etc)
- Difficult when it come to horizontal scaling, include a server will require to re-adjust microservices cluster to balance resource usage across multiple server instances

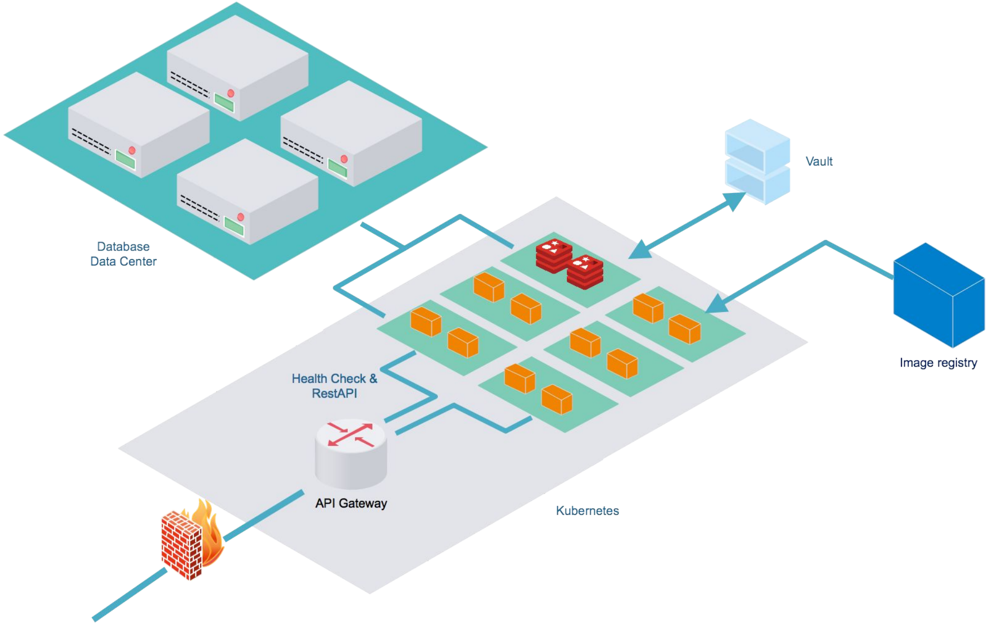
So what is the solution to this problem ?

Kubernetes

Kubernetes is an open-source backed by Google and the large community. Kubernetes provide a solution to make microservices as simple as possible. Kubernetes include almost all component from that need by microservices architecture

- Load Balancer (Global and Internal)
- Service Discovery
- Horizontal scaling and auto scaling, adding new server and spawn a new microservice in a cluster is very simple
- Self healing
- Container base platform
- Secret data

Overall Architecture & Infrastructure with Kubernetes

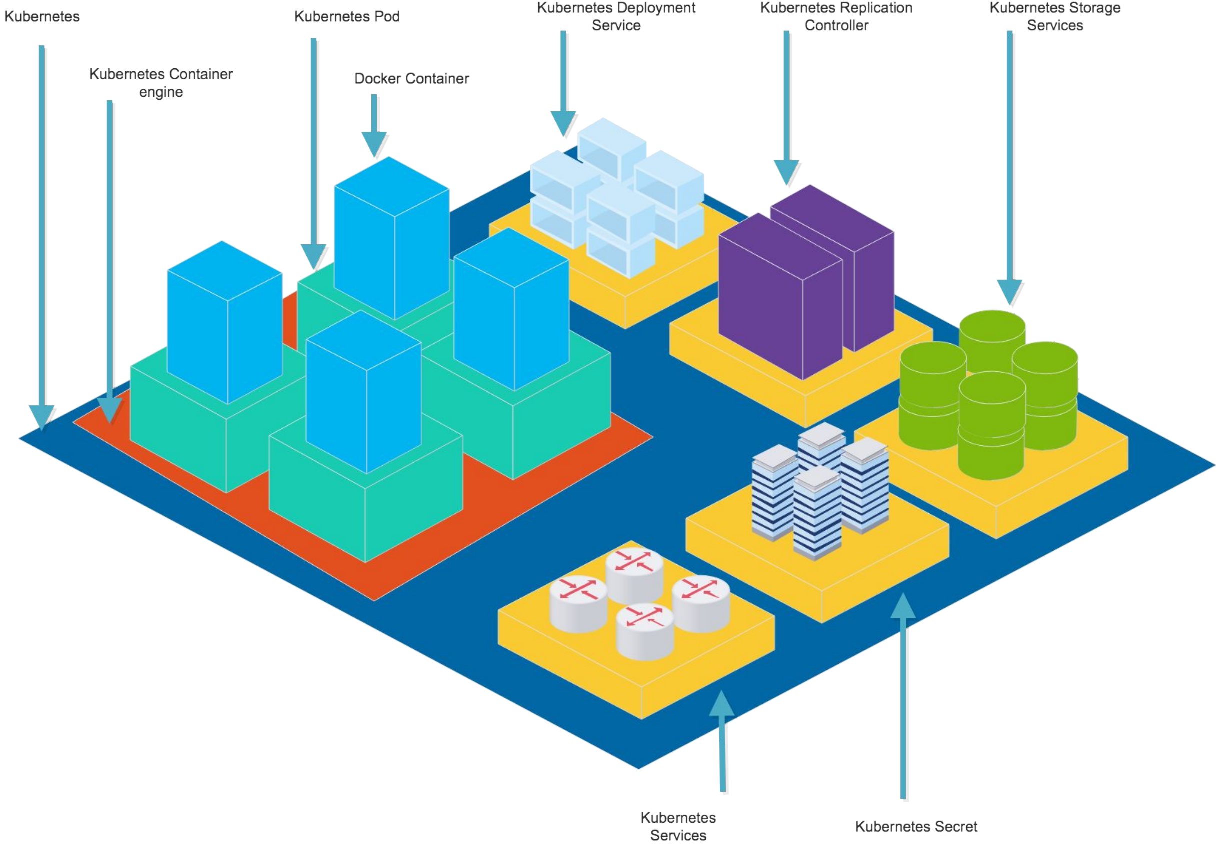


Kubernetes Key Concept (1)

- **Master** is refer to a group of VM or physical machine that host Kubernetes master
- **Node** is refer to a group of VM that host a Pod
- **Pod** is refer to a VM that running the microservice or application inside the docker images
- **Label** is refer to a tag of each configuration such as deployment
- **Deployment** is refer to a configuration template of a cluster of the microservice
- **Replication Controller** is refer to a service that responsible to manage the cluster of a microservice based on Deployment configuration
- **Services** is refer to a network service that responsible for creating cluster virtual ip address, manage load balancer, routing and mapping

Kubernetes Key Concept (2)

- **Storage** is refer to a network storage service which responsible to create a session storage or a persistent storage that can be shared across all microservice
- **Scheduler** is refer to a service that responsible to execute a job or task at the given time. Scheduler also support syntax like Cron Job on Linux
- **Container** is refer a service engine that responsible execute Pod in a sandbox


Kubernetes
Kubernetes Pod
Kubernetes Deployment Service
Kubernetes Replication Controller
Kubernetes Storage Services
Kubernetes Container engine
Docker Container
Kubernetes Services
Kubernetes Secret

Kubernetes Installation

Cloud

- Google Cloud
- Amazon Web Service
- Microsoft Azure
- Digital Ocean
- Openshift

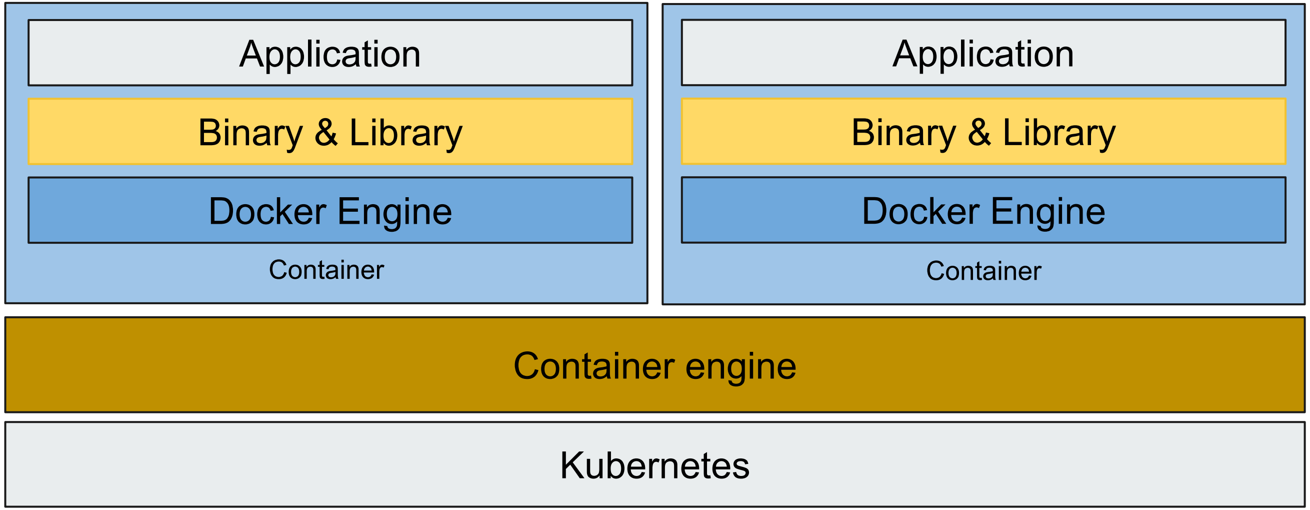
None Cloud

- Ubuntu
- Tectonic by Kubernetes Core Team (Free upto 10 nodes)
- Custom Solution

Kubernetes Alternative

- Nomad by Hashicorp
- Docker Swarm

Use Docker as Kubernetes Container



Building Docker

Docker is a very simple tools, what we need is install docker via docker installation, create Docker file and we build docker images with a single command line.

```
FROM alpine
ADD service /
ENTRYPOINT /service
EXPOSE 8080
```

Docker file reference

Kitematic Docker UI management

Microservices & Kubernetes

Best Practice and Resources:

- Use message distribution (RabbitMQ, NSQ ...etc) or Distribution Streaming (Apache Kafka ...etc) to write log for all microservices. ***Notes*** Kafka is not just for logging but can be use for more like realtime distribution message.
- Java Spark, minimal library for java microservice, language like Go, Rust does not require external library
- Learn more about TDD, BDD with awesome automate test. For Go, build-in library provide enough way to implement TDD and BDD

Microservices & Kubernetes

Q/A

Vault

Vault is an open-source solution for secure key management system (KMS) and other sensitive data.

- Secret Key such database username & password
- RestAPI Key and Secret Key
- Private & Public Certificate such as key use to generate use authentication token
- User’s authentication token
- More

Interacting with Vault is done through RestAPI

Vault

Q/A

Example (1)

Application is shopping online !

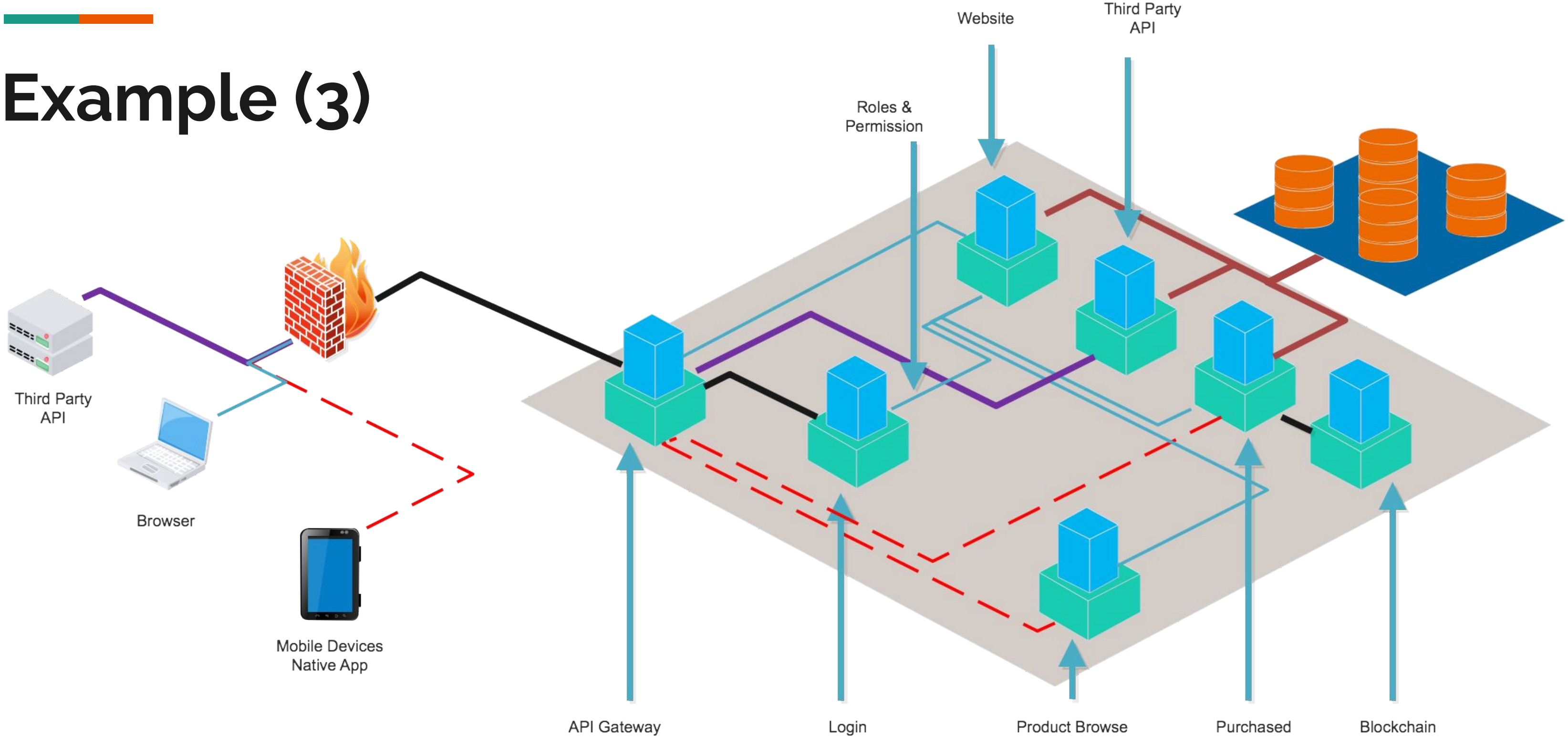
- Website (Desktop, Mobile)
- Mobile App
- Third party

Example (2)

Application is shopping online !

- Login
- Browser Product
- Purchase a product

Example (3)

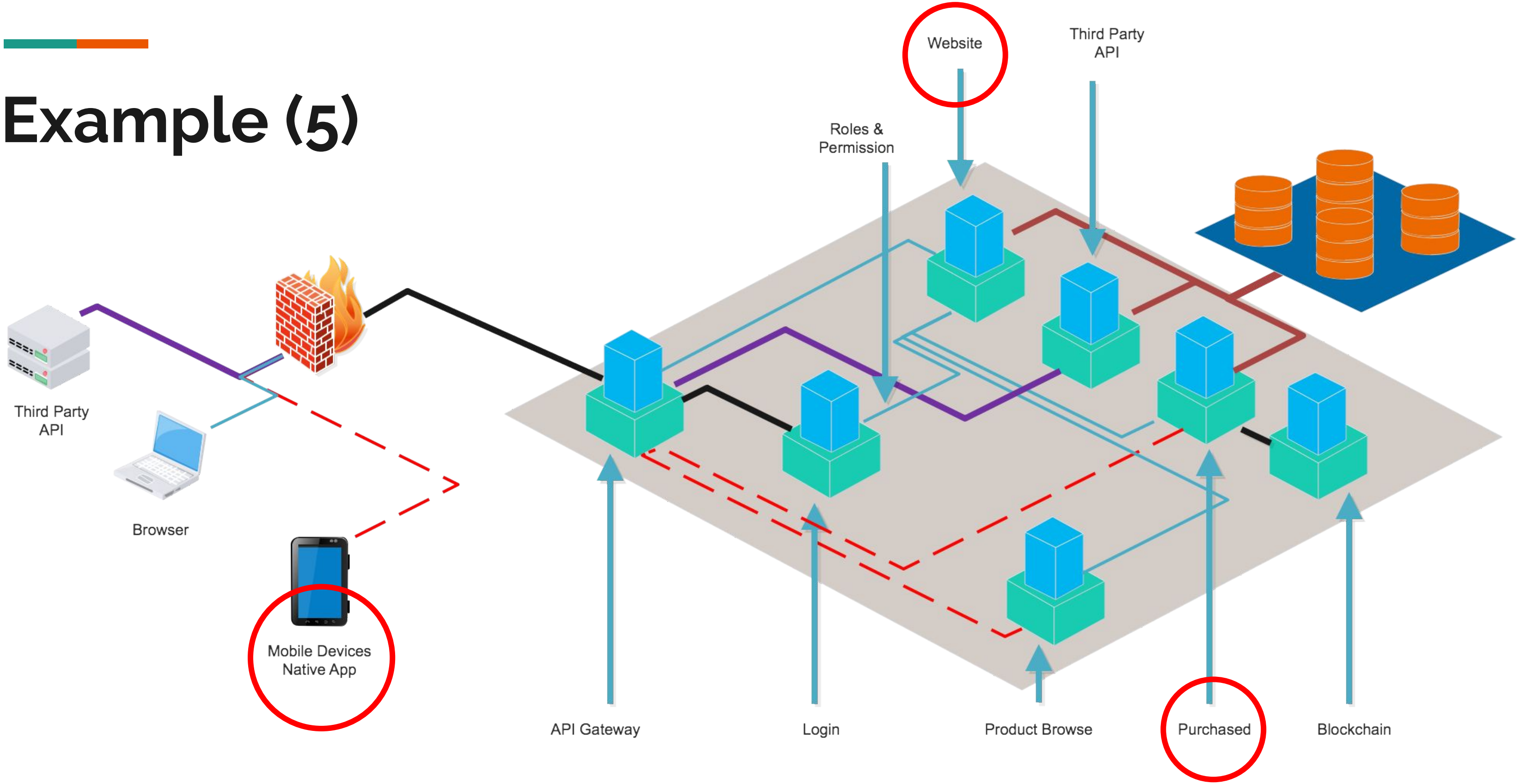

Website
Third Party API
Roles & Permission
Third Party API
Browser
Mobile Devices Native App
API Gateway
Login
Product Browse
Purchased
Blockchain

Example (4)

Application is shopping online !

- Login
- Browser Product
- Purchase a product, ***promotion? Via promotion code or coupon***

Example (5)

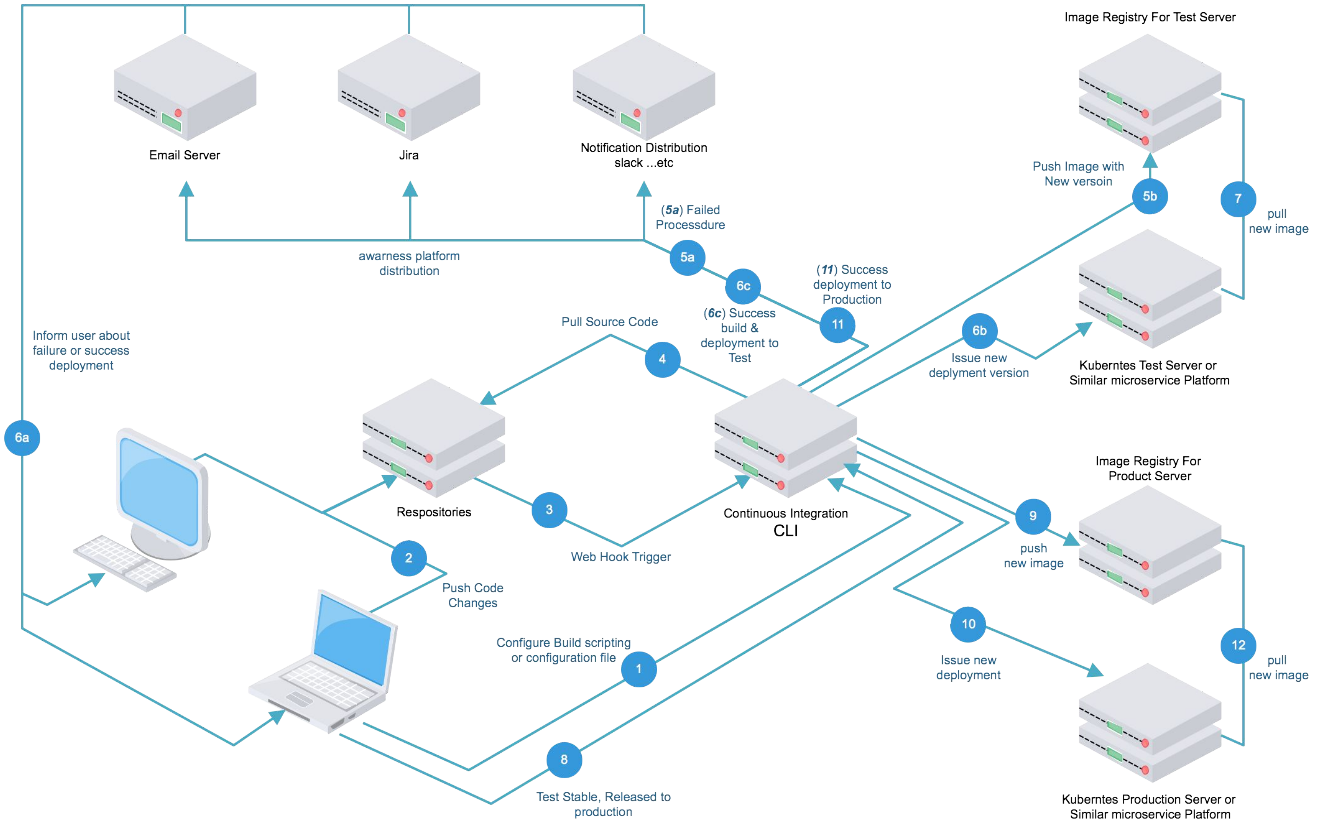

Website
Third Party API
Roles & Permission
Third Party API
Browser
Mobile Devices Native App
API Gateway
Login
Product Browse
Purchased
Blockchain

Example (6)

Caution !

- “Purchase” microservices must maintain Rest API backward compatibility

Continuous Integration CLI (1)



Continuous Integration CLI (2)

CLI is tools which provide ability to execute certain task based on script or configuration and even more. So it’s required a few dependencies before CLI update and running.

- System Required such as, Platform, CPU, Ram, Java Runtime (If CLI that build on top of Java)
- Platform Compiler or Interpreter (Java need both compiler and interpreter while Native app required only compiler
- Docker Engine or similar Container, Kubernetes required Docker at least
- Scripting (Shell script mostly)
- Test Engine (depend on framework), JMeter one of most popular tools for loading test framework.

Continuous Integration CLI (3)

Common CLI Task for most of the project

- Web Hook activated (Trigger by repositories server), could similar way, however web hook is the most popular out there, especially with GIT
- Pull Source code
- Follow defined script or execute by configuration (Compiler, Run Test, ...etc)
- Failed, inform user (see diagram in the previous slide)
- Success, deploy to test

Continuous Integration CLI (4)

Popular CLI

- Jenkins
- Hudson - Haven’t use it

Most of the new comer is cloud based such as CircleCI is cloud based does not have standalone or on-premises.

Continuous Integration CLI

Q/A

Go Programming Language (1)

Go is programming language created by Google at 2007 and open source at 2009.

- Simplicity
- Better performance
- Lightweight
- Provided concurrency out of the box, painless thread
- Garbage collection
- Build-in library is almost what we need
- Build-in Unit Test and Benchmark
- Work perfectly with microservices architecture and container

Go Programming Language (2)

Downside of Go

- Error Handling is the most annoying ever
- No Polymorphism

Go Programming Language (3)

Best places to learn go is try out it tour. https://tour.golang.org/welcome/1

Before trying on your own computer make sure to download and install Go https://golang.org/dl/.

```go
package main

import (
   "net/http"
   "encoding/json"
)

func main() {
   mux := http.NewServeMux()
   mux.HandleFunc("/", func(writer http.ResponseWriter, request *http.Request) {
      writer.WriteHeader(http.StatusOK)
      json.NewEncoder(writer).Encode(struct {
         Code int `json:"code"`
      }{
         Code: 200,
      })
   })
   http.ListenAndServe("localhost:8080", mux)
}
```

Thread in GO

```go
go func(number int) {
  for {
    number++
    if number > 1000 {
      break
    }
  }
}(0)
```

Testing & Benchmarking in GO

Go has a predefined pattern unit test pattern and the community has build up a strong framework for tdd and bdd.

- A filename must end with "_test.go"
- A unit test function name must start with "Test" and take a single argument "testing.T"
- A benchmarking function name must start with "Benchmark" and take a single argument "testing.B"

Cross-Compiling in GO

```
# env  GOOS=linux  GOARCH=amd64 go build -o service main.go
```

- “service” is output compiler name
- “main.go” is go file contain main function, it could a directory

Alternative outside of Go

- Swift by Apple
- Rust by Mozilla

All above alternative choice is a compiler language and are all type safe.

Q/A

Demo (1)

Goal

- Start local Minikube
- Deploy a microservice
- Test

Demo (2)

To Install minikube follow the instruction at https://github.com/kubernetes/minikube, also you install kubernetes command line as well, please follow the instruction at kubernetes website.

- Start docker
- Start minikube with command line "*minikube start*"
- Start access kubernetes dashboard by running command "*minikube dashboard*"
- Tell kubernetes command line to use minikube context "*kubectl config use-context minikube*"
- Tell minikube to use local docker "*eval $(minikube docker-env)*"
- Create sample and docker image
- Deploy microservice
- Access to microservice "*minikube service sample1 --url*"

Thanks You !

- Microservices
- Kubernetes, @kubernetesio
- Docker
- Vault
- Go