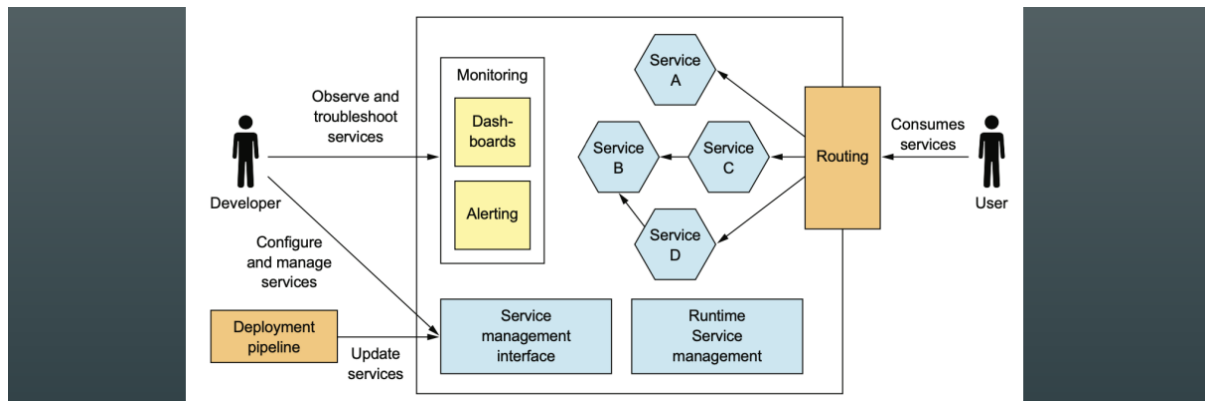


Chapter 10 Deployment

Overview of Microservices Deployment

- **Process** >> Steps ที่ทำโดยคน
- **Architecture** >> Structure ของ Environment
- Heavyweight >>
- Lightweight >>



- Production Environment
 1. **Service Management Interface** >> ทำให้ developer จัดการ service ได้
 2. **Runtime Service Management** >> ensure ว่า service กำลัง run อยู่
 3. **Monitoring**
 4. **Request Routing**

Language-Specific Packaging Format

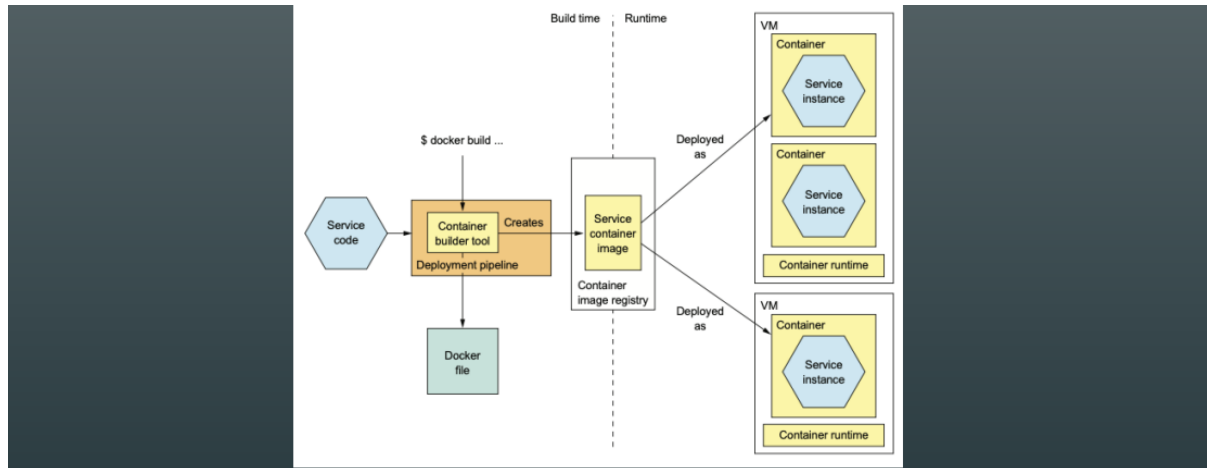
- **Language-Specific Packaging Format** >> เป็นการ deploy ลงใน **bare machine (local)**
- แต่ละภาษาใช้วิธีแตกต่างกัน
 - **Spring Boot-based Java**
 - **NodeJS**
 - **GoLang**
- ข้อดี
 - Fast Deployment
 - Efficient Resource Utilization >> เครื่องมีเท่าไร ใช้เต็มที่
- ข้อเสีย
 - Lack of **Encapsulation** of the Technology Stack
 - จำกัดให้ service หนึ่งใช้ **resource** เท่าใดไม่ได้
 - ต้องจัดการทุกอย่างเอง >> automatically determining **where to place service** instances is **challenging**

Virtual Machine

- **Virtual Machine** >> นำ service ไป deploy ไปไว้ใน VM instance
- ข้อดี
 - VM image **encapsulate technology stack** >> ไม่จำเป็นต้องรู้ว่าเครื่องจริง ๆ นั้นเป็นยังไง
 - Service **instance are isolated** >> ถึงพังก็ไม่เป็นไร สร้างใหม่ได้

- **mature cloud infrastructure** >> Cloud VM ไม่ใช่ของใหม่ stable แล้ว
- ข้อเสีย
 - less-efficient **resource utilization** >> ต้องแบ่ง resource ส่วนนี้มาจัดการ Cloud VM
 - Relatively **slow deployments** >> เสียเวลามา config ตอนสร้างเล็กน้อย
 - System **administration overhead** >> อาจมีเรื่อง security เข้ามาเกี่ยว ทำให้จัดการได้ช้าลง

Container



- **Container** >> Isolated sandbox ที่ run อยู่ใน **single machine (shared OS)**
- ข้อดี
 - เหมือน VM แต่ **lightweight** และเร็วกว่า
 - ถ้าใช้ Docker compose จัดการ Service Discovery ให้
- ข้อเสีย
 - Responsible for the **heavy lifting** >> หมายถึงมีความรู้และจัดการ Docker images
 - และก็ต้องจัดการ **VM** รวมถึง **infrastructure** ของมันด้วย

Kubernetes

- **Kubernetes** >> Docker orchestration framework
- ข้อดี
 - Pods มี Dynamic IP แต่ **Kubernetes** จัดการ **Service Discovery** ให้




รายละเอียดทุกอย่างตามที่เรียนใน SDS

Using a service mesh to separate deployment from release

- อยาก rollout (release new version) ทำได้ตามขั้นตอนนี้
 1. Deploy new version into production **without traffic routing**
 2. **Test** in production
 3. ค่อย ๆ **ปล่อย** new version ให้ user ทีละนิด ๆ
 4. ถ้าโดยรวมโอเคแล้ว ใช้ version ใหม่ทั้งหมด

- **Service Mesh** >> networking infrastructure ที่จัดการ **rule-based load balancing** และ **traffic routing**
 - ก่อนหน้าที่จะมี service mesh ต้องทำเองกับมือ แต่พอมีแล้วสามารถเลือกได้ว่าจะใช้ **new version** ที่ % จากทั้งหมด
- **Istio** >> Open Platform (Service Mesh) ตัวนี้
 - Traffic Management (Serv)



- **Traffic management**—Includes service discovery, load balancing, routing rules, and circuit breakers
- **Security**—Secures interservice communication using Transport Layer Security (TLS)
- **Telemetry**—Captures metrics about network traffic and implements distributed tracing
- **Policy enforcement**—Enforces quotas and rate limits

- ประกอบไปด้วย
 - **Control Plane**
 - **Plot** >> จัดการเกี่ยวกับการเชื่อมต่อระหว่าง Kubernetes กับ Pods
 - **Mixer** >> จัดการ Metric และ Monitoring ต่าง ๆ
 - **Data Plane**

Serverless (AWS Lambda)

- **Heavy lifting** >> พุ่งถึง**การจัดการ computer resource** (ยังต้องจัดการเยอะ ยิ่งหนัก)
- **Severless** >> ช่วยให้เราสามารถ Deploy โดยไม่จำเป็นต้องรู้ infrastructure ด้านล่างได้
- **AWS Lambda** >> ถูกสร้างมาเพื่อใช้ deploy **event-driven services**
- ข้อดี
 - **Integrated** with many **AWS Services**
 - **Eliminates** many **system admin tasks**
 - ไม่ต้องคิดเรื่อง VM หรือ Container
 - **Usage-based pricing**
- ข้อเสีย
 - **Long-tailed latency** >> บางที scale ไม่ทัน request ที่ต้องการความเร็ว
 - **Limited** event/request-based programming model >> ไม่ได้สร้างมาให้ใช้งานกับ **long-running services** (เช่น message broker หรือ run model)