Microservices Cheat Sheet

V2021.03.06

(Dr Yan Xu)

What is Microservices for?

➔ **Fast** response to business requirement changes

Microservices Architecture

➔ **Componentization via Services**
- via Services
  - Slower, due to network delay
  - Independent development and deployment
- via Library
  - Faster, since directly called within same process
  - Single development and deployment

➔ **Organized Around Business Capabilities**
- Business Entities
- Entity Grouping into Services
- Goal:
  - minimize cross-team communication
  - clear ownership and responsibility

➔ **Customer-Facing Development**
- "You build it, you run it" - Werner Vogels AWS CTO

➔ **Endpoint is the most important design**
- Use Simple and Standard Endpoints
- Version

➔ **Decentralized Governance**
- Each team makes its own decisions

➔ **Automate Everything**
- Infrastructure Automation
- CI/CD automation and speed
- Logging, monitoring and alerts

➔ **Decentralized Data Management (Be Careful)**
- Not always optimal or possible
- Single DB vs Isolated DB

➔ **Design for Failure**
- Exception Isolation, Re-try, Log, On-Call

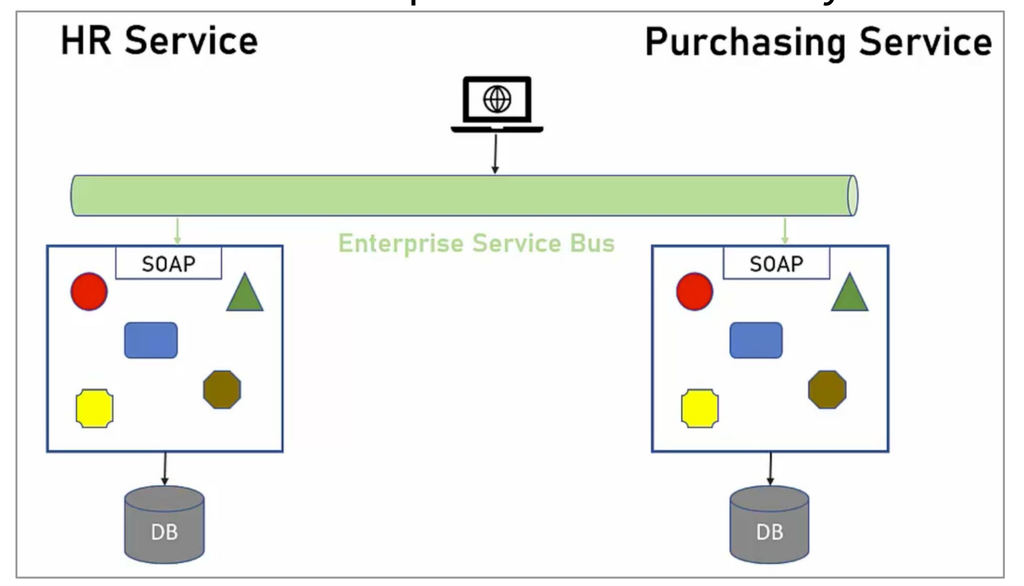
➔ **Iterative Design (working with Agile Method)**

Alternative Architectures

➔ **Monolith Architecture**
- Single design, development and deployment. Code is executed in the same process
- Use Cases:
  - Gaming
  - Desktop App (e.g. Adobe and Microsoft)
  - Embedded App
- Most Web App moved to Microservices

➔ **Monolith vs Microservices**
- Single Technology Problem
- Inflexible Deployment
- Inflexible Scalability
- Large & Complex Code Repository
- When applicable, Microservices indicates:
  - Lower cost (both development and maintenance)
  - Faster response to business requirement changes

➔ **Service Oriented Architecture**
- ESB controls everything between services
- SOA is too complex and not used any more



When not Microservices?

➔ **Small System**

➔ **Intermingled Business Logic**
- Independent services is not possible

➔ **Performance requirement <10ms**
- Military projects
- Gaming

➔ **POC System**

➔ **Systems without requirement change**
- Microservices superpower is fast change

Service Design

➔ Service design is the most critical step

➔ Method: Business Entity Analysis

➔ Criterion:
- minimize logic/data dependency
- support team-wise decision making

Service Communication Pattern

➔ 1-to-1 Sync
- synchronous service API (most common)
- solution: REST, GraphQL, gRPC
- notes:
  - recommend to use Gateway:
    - Load balance
    - Endpoint Server Isolation

➔ 1-to-1 Async
- asynchronous service API
- e.g. payment, heavy processing task
- solution: Queue (e.g. RabbitMQ, AWS SQS)

➔ Pub-Sub/Event Driven:
- M-N communication pattern
- solution: Queue (e.g. RabbitMQ, AWS SQS)

Web Development Stacks

➔ Each team chooses its own tech stack

	App Types	Type System	Cross Platform	Community	Performance	Learning Curve
.NET	All	Static	No	Large	OK	Long
.NET Core	Web Apps, Web API, Console, Service	Static	Yes	Medium and growing rapidly	Great	Long
Java	All	Static	Yes	Huge	OK	Long
node.js	Web Apps, Web API	Dynamic	Yes	Large	Great	Medium
PHP	Web Apps, Web API	Dynamic	Yes	Large	OK -	Medium
Python	All	Dynamic	Yes	Huge	OK -	Short

(source:https://www.udemy.com/course/microservices-architecture-the-complete-guide/learn/lecture/20963698#overview)

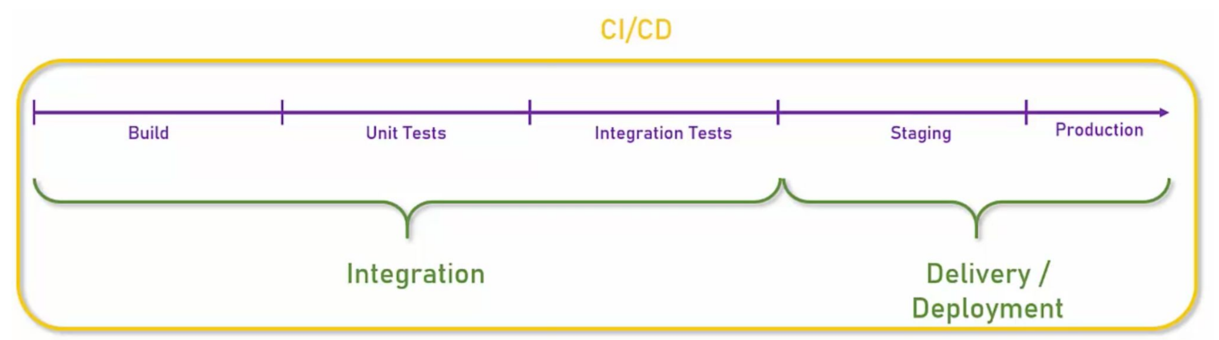
➔ Team existing skills & market skill availability should be considered

➔ Database
- Relational DB: MySQL, PostgreSQL
- NoSQL DB: DynamoDB, MongoDB
- In-memory DB: Redis

➔ Deep Learning Stack: Pytorch, TensorFlow

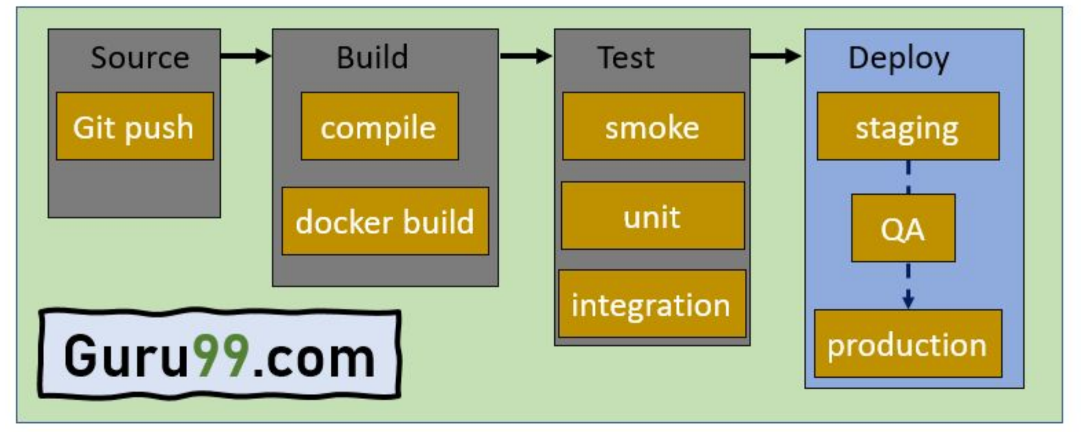
Fast Deployment

➔ CI/CD (daily deployment)



(source: https://www.udemy.com/course/microservices-architecture-the-complete-guide/learn/lecture/21000674#overview)

➔ CI/CD Pipeline

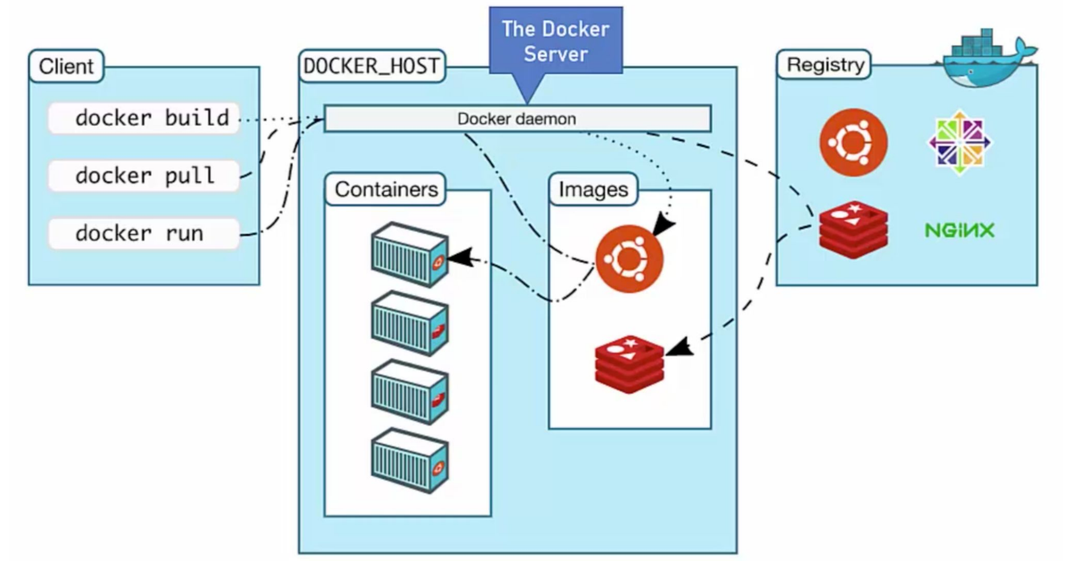


(source: https://www.guru99.com/ci-cd-pipeline.html)

➔ CI/CD Products (2 examples)
- Jenkins
- Bamboo

➔ Container
- Designed for predictable behavior
- Container vs VM
  - Container reuses Host OS, while VM has its own OS
  - Container takes seconds to launch, while VM takes minutes

➔ Docker



https://docs.docker.com/get-started/overview/

➔ Example docker file

```
WORKDIR /opt/node_app
COPY package.json package-lock.json* ./
RUN npm install --no-optional && npm cache clean --force
ENV PATH /opt/node_app/node_modules/.bin:$PATH
WORKDIR /opt/node_app/app
COPY . .
```

Fast Deployment (Continued)

➔ Container Management
- Deployment
- Scalability
- Routing (load balance)
- Monitoring

➔ Container Management Standard
- Kubernetes (from Google 2014)

➔ Code Example: github

Testing

➔ Unit Test
- Cover each method

➔ Integration Test
- Cover most data/Logic paths in the service
- Dependency service or 3rd-party system (DB)
  - ***Stub***
    - Simple functions to replace dependency service/system
  - ***Mock***
    - Only verifies access was made
    - Hold no data or adaptor functions

➔ End-to-end Test
- Cover only key user scenarios
- Should touch all services
- Maybe not part of CI/CD

Logging

➔ Logging should a centralized service and logging format should be designed (e.g. json)

➔ Context ID given crossing all services

➔ Logging as much as possible, with
- Context ID (critical)
- Timestamp
- User
- Severity
- Service
- Message
- Error Stack Trace (critical)

➔ Severity
- DEBUG, INFOR, WARNING, ERROR

➔ Logging Service:
- Splunk, ELK Stack

Observability

➔ Health Check API & Health Dashboard

➔ Team-wise Exception Dashboard

➔ Metrics
- Infrastructure Metrics
  - CPU, memory, Disc
- Application Metrics
  - Error, Warning, Request/min, Log

➔ Metrics Dashboard
- Splunk, ELK Stack

➔ Alerts
- Email, Slack, Phone Call

➔ Escalation Policy

➔ Incident Review Meeting (when required)

Anti-Pattern

➔ Services have no clear boundary

➔ Poor API Design
- consistent
- versioned
- typically should be designed, not programmed

➔ Ignoring Services Dependency

➔ Significantly Change Service Boundary

Monolith -> Microservices

➔ Three Candidate Strategies:
- Strategy 1: Only New Modules as Services
- Strategy 2: Module as Service One by One
- Strategy 3: Complete Rewrite

➔ Rule: the more complex the existing system is, the more biased towards Complete Rewrite

Some notes

➔ "Supporting faster business iteration" is the ONLY priority for microservices, we should never overlook it

➔ If the team structure does not fit services, the team structure should be changed

➔ Microservices should be applied together with Agile methodology to enforce high-quality and fast-paced business iteration