Microservices Architecture

The Complete Guide

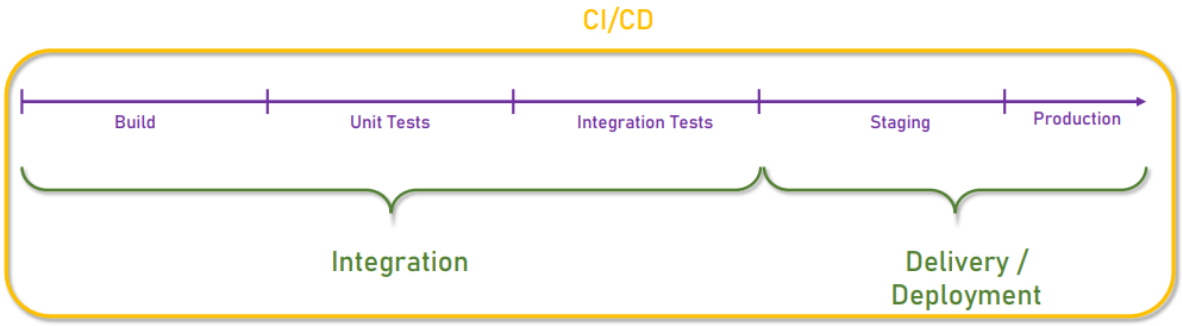
Notes

- Repository – https://github.com/sameerbhilare/Microservices
- Refer Course Slides as per sequence
- Using multiple development platforms in one system is called **Polyglot**
- https://martinfowler.com/articles/microservices.html
- When you have many services, direct communication between services can cause **Spiderweb** problem.
- Libraries – called directly within the process
- Services – called by out-of-process mechanism (Web API, RPC, REST)
- "You build it, you run it!" meaning your job is not done by building the service you are responsible for successfully running it.
- Remember: Components = Services
    - When talking about components in microservices architecture, we actually talk about services as opposed to libraries in a monolithic system.
- Cross-cutting services are services that provide a system wide utilities, meaning utilities that are not tied to a specific business scenario, but that almost every service can benefit from. Common examples: Logging, Caching, User management
- Backend Development Platform

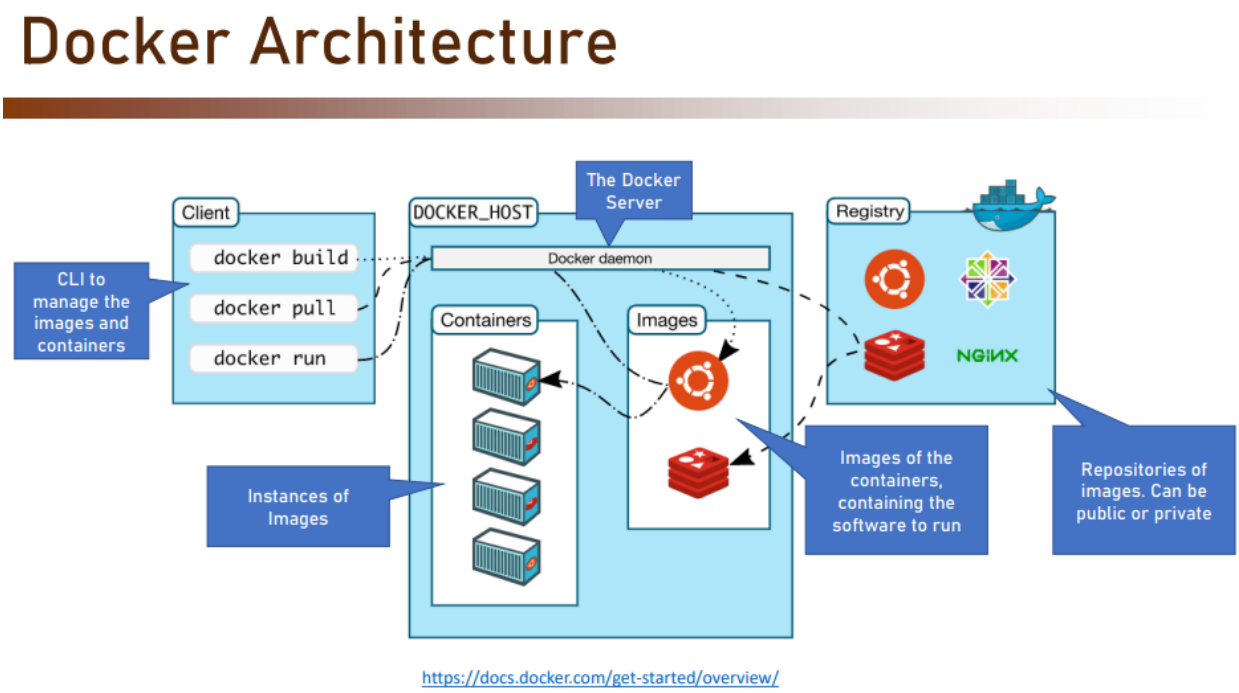
Development Platform

	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

- CI / CD



- Difference between VM and Container. VMs have their own guest OS on top of Host OS.
- Docker Architecture



- Pod is the basic building block in Kubernetes.
- Pod is the atomic unit in Kubernetes.
- You can think of pod as a container of containers.
- In other words, the docker container leaves inside a pod and the pod functions as a wrapper around it, provides connectivity and monitoring
- Service Mesh Manages all service-to-service communication. And also provides handling for problems like timeouts, security, retries, monitoring.
- Service Mesh provides Services like Protocol conversion, Communication security, Authentication, Reliability (timeouts, retries, health checks, circuit breaking), Monitoring, and Service Discovery.

- Circuit breaker monitors the service and checks how many time outs it experiences. If this number goes above some threshold, the circuit breaker goes into action and actually gets that service off. And from now on, when a call is sent to that service, the circuit breaker will immediately respond with an error and will not wait for the service to time out. This way we prevent a cascading failure of the service.
- You should use Service Mesh ONLY IF –
    - You have a lot of services…
    - Which communicate a lot with each other
    - Or you have a complex communication requirement with various protocols or brittle network
- We must have CorrelationID for each log, so that we can trace the end to end flow for user actions.
- When Not to Use – Small systems, Intermingled Functionality or Data, Performance Sensitive Systems, Quick-and-Dirty (POC) Systems, No Planned Updates systems.
    - If almost all requests for data span more than one service – there's a problem. Don't use microservices in such cases.
- "Every internal team should be small enough that it can be fed with two pizzas" - Jeff Bezos
- Some of the Anti-Patterns and Common Mistakes which we must avoid – No Well-Defined Services, No Well-Defined API, Implementing Cross-Cutting Last, Expanding Service Boundaries.
- Does a **service** depend on other services? Here the word 'depends' means – will the service continue working if it will be the only operational service in the system? Note that we can say a service is independent if it depends on data managed outside its scope.