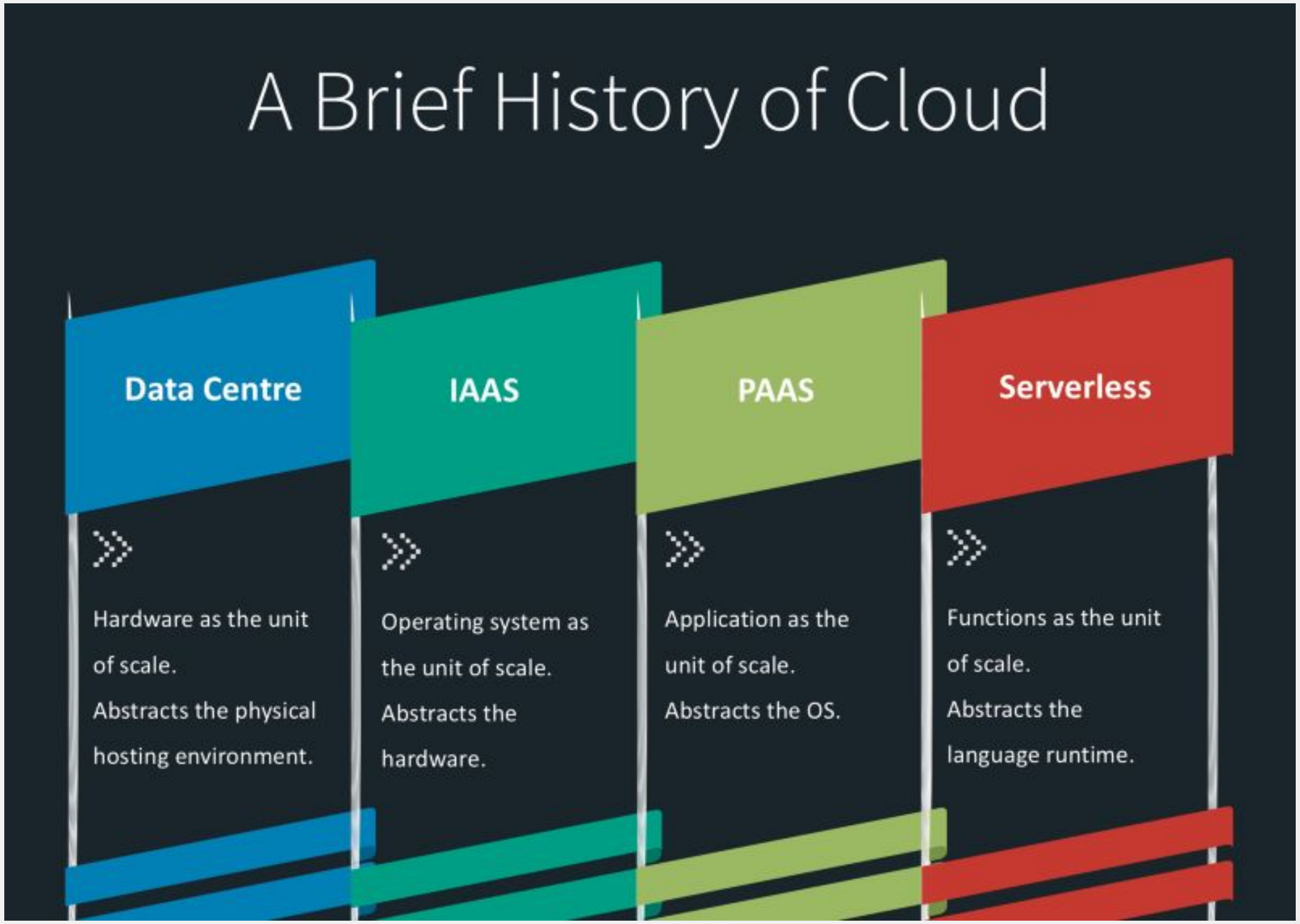
Building High Quality Openshift Applications

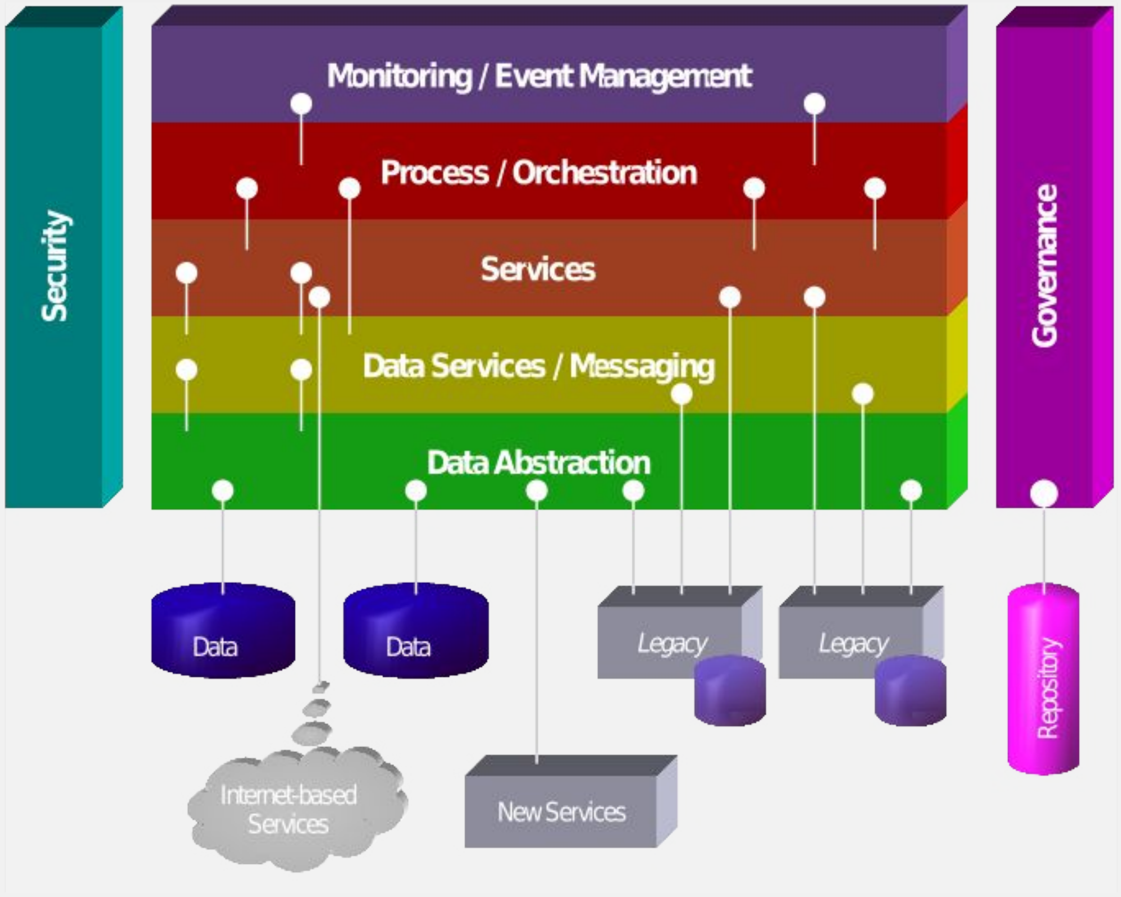
Ip Sam
Senior Consultant
Red Hat Tech Exchange 2019

History of Cloud Computing


A Brief History of Cloud

Data Centre
Hardware as the unit of scale.
Abstracts the physical hosting environment.
IAAS
Operating system as the unit of scale.
Abstracts the hardware.
PAAS
Application as the unit of scale.
Abstracts the OS.
Serverless
Functions as the unit of scale.
Abstracts the language runtime.

Microservices



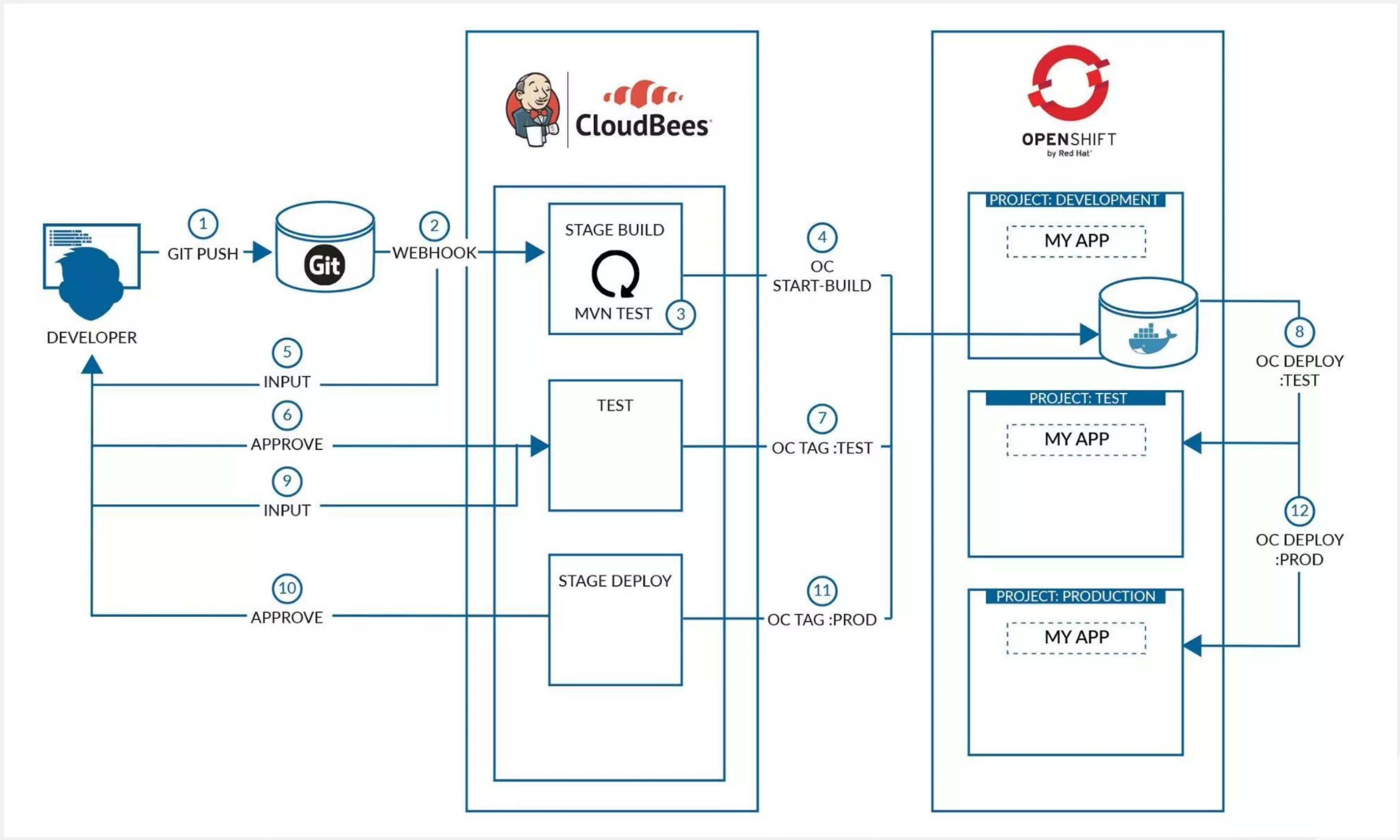
What makes a good Openshift Application?

Your Answers:

Openshift Characteristics

Applicatoins adopting the principles of Microsoftservices packaged as Containers orchestrated by Platforms running on top of Cloud infrastructure.

Openshift Ecosystems



SOLID Principles in App Development

- **Single Responsibility Principle**
- **Open Closed Principle**
- **Liskov Substitution Principle**
- **Interface Segregation Principle**
- **Dependency Inversion Principle**

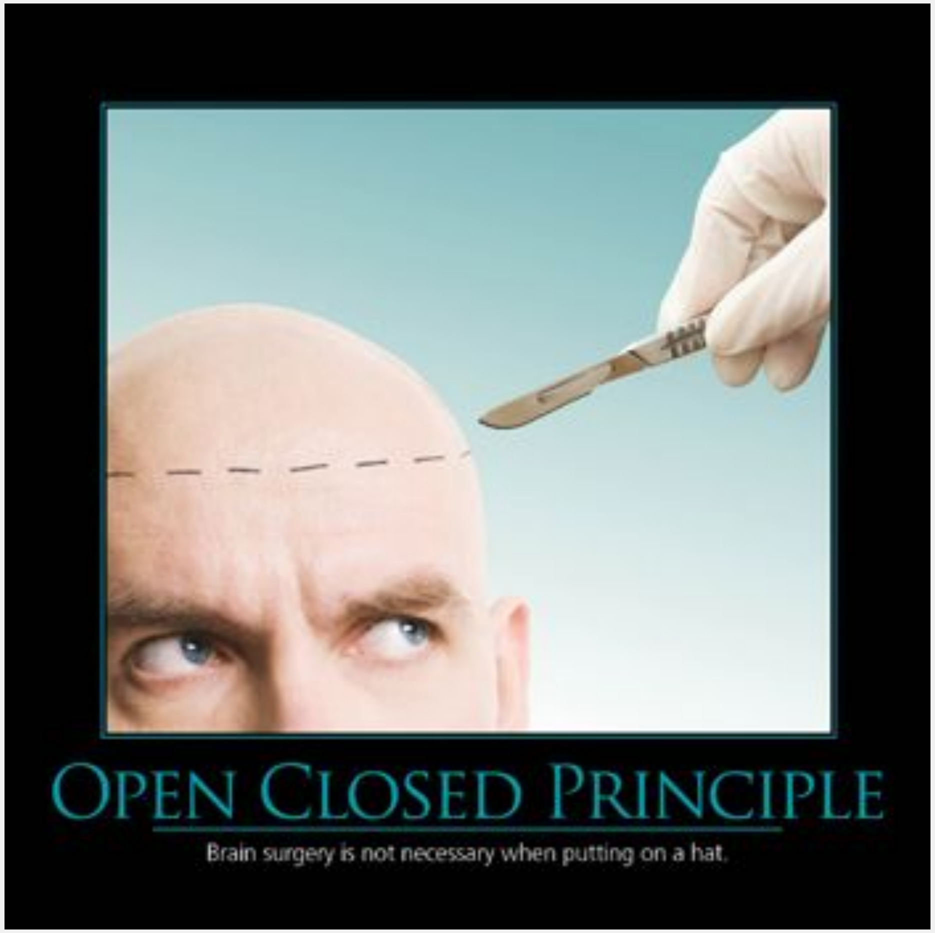
Single Responsibility Principle

SINGLE RESPONSIBILITY PRINCIPLE
Just Because You Can, Doesn't Mean You Should

Single Responsibility Principle

A class should only have a single responsibility, that is, only changes to one part of the software's specification should be able to affect the specification of the class.

Open Closed Principle



Open Closed Principle

Software entities should be open for extension, but closed for modification.

Examples include using interfaces and abstract classes.

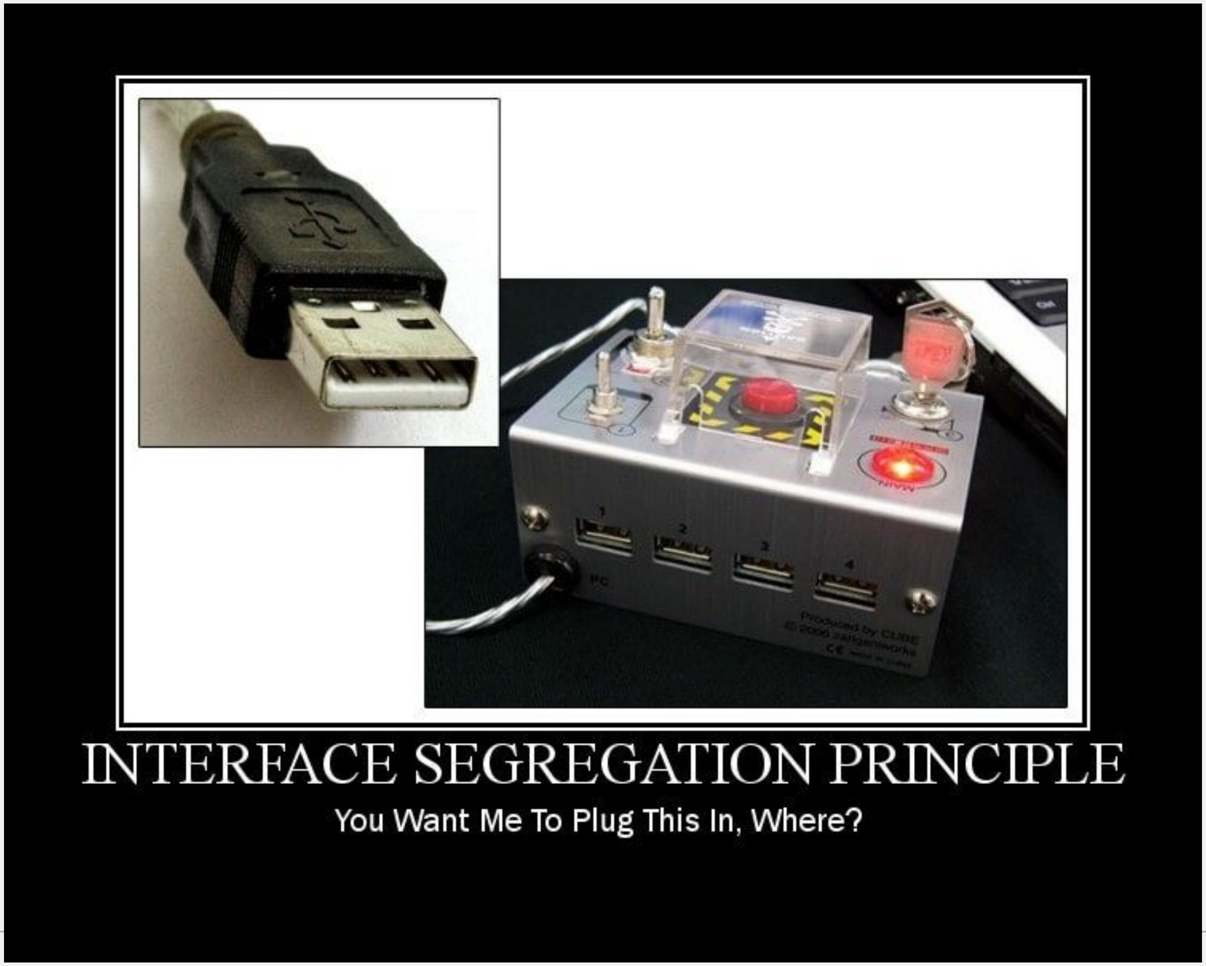
Liskov Substitution Principle

LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

Liskov Substitution Principle

Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.

Interface Segregation Principle


INTERFACE SEGREGATION PRINCIPLE
You Want Me To Plug This In, Where?

Interface Segregation Principle

Many client-specific interfaces are better than one general-purpose interface.

Dependency Inversion Principle

DEPENDENCY INVERSION PRINCIPLE
Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?
Radio Shack
Rosin Core Solder

Dependency Inversion Principle

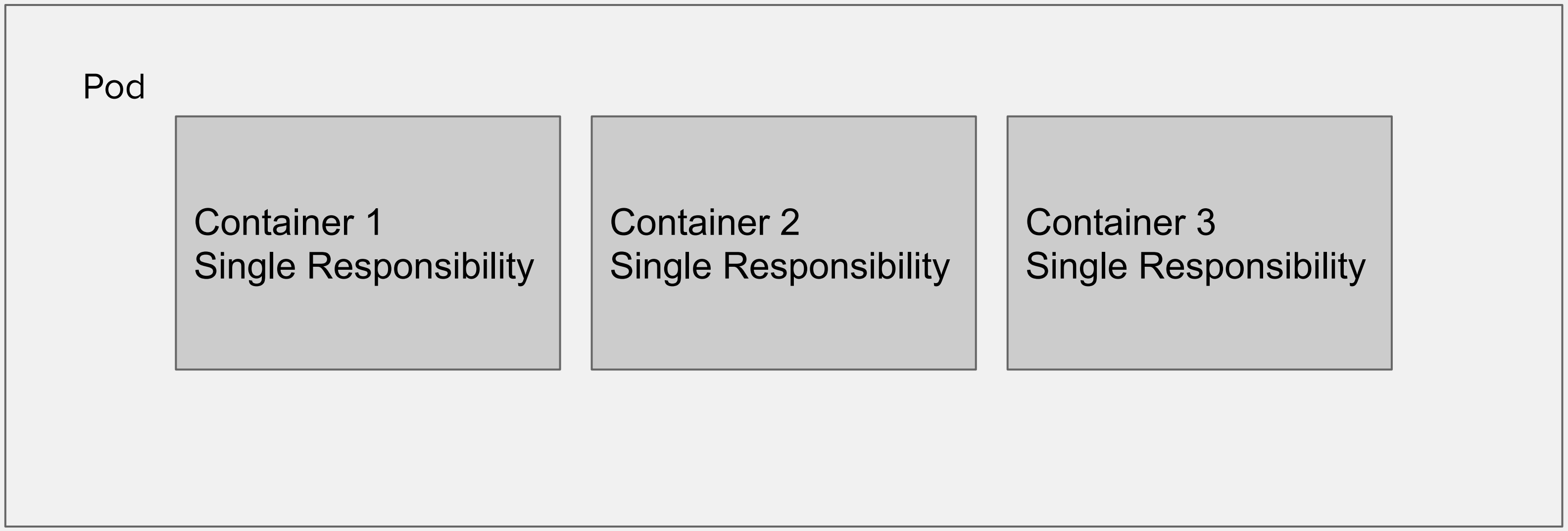
One should depend upon abstractions, not concretions.

Examples include autowired dependency injection, bean creation.

SOLID Principles in Openshift

- **Single Responsibility Principle**
- **Self Containment Principle**
- **Image Immutability Principle**
- **High Observability Principle**
- **Lifecycle Conformance Principle**
- **Process Disposability Principle**
- **Runtime Confinement Principle**

Single Responsibility Principle



Self Containment Principle

Container should not rely on anything else except the linux kernel that it runs on.

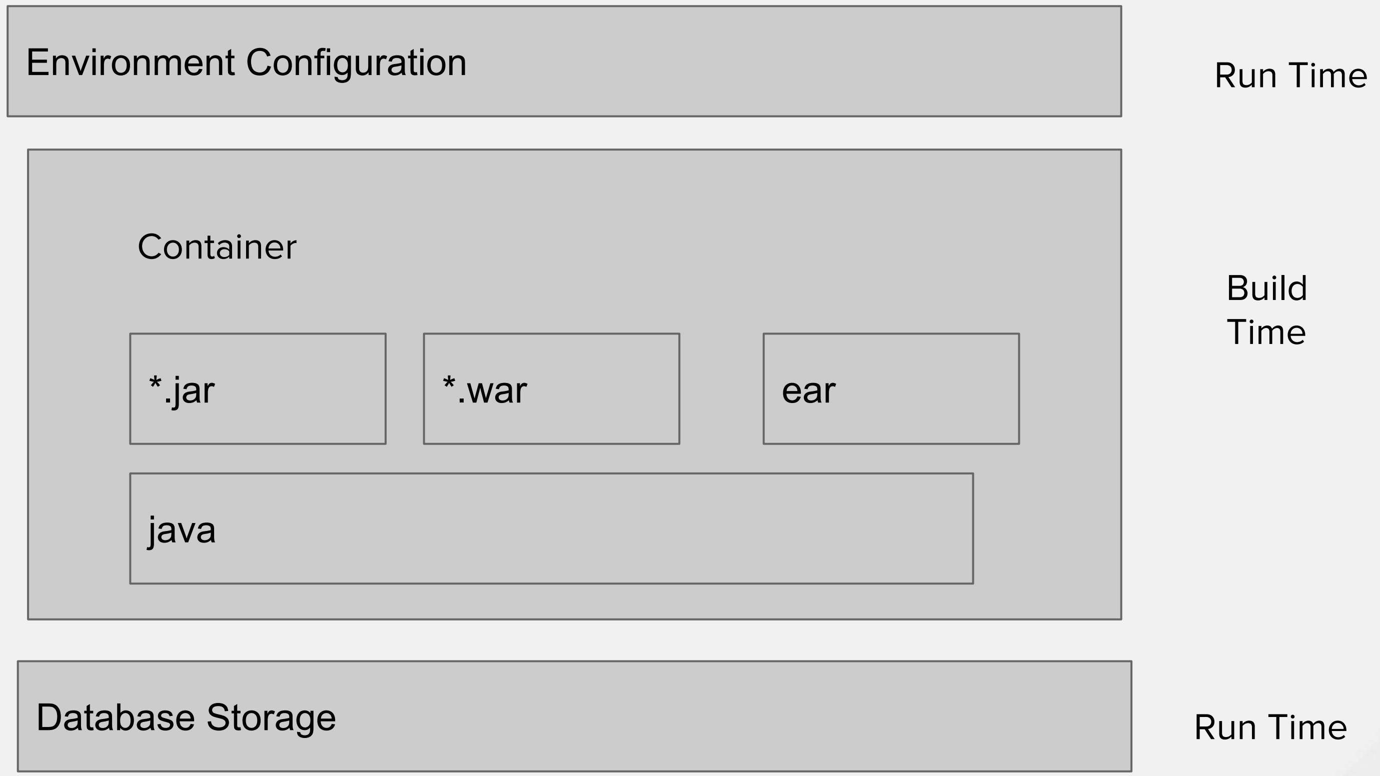
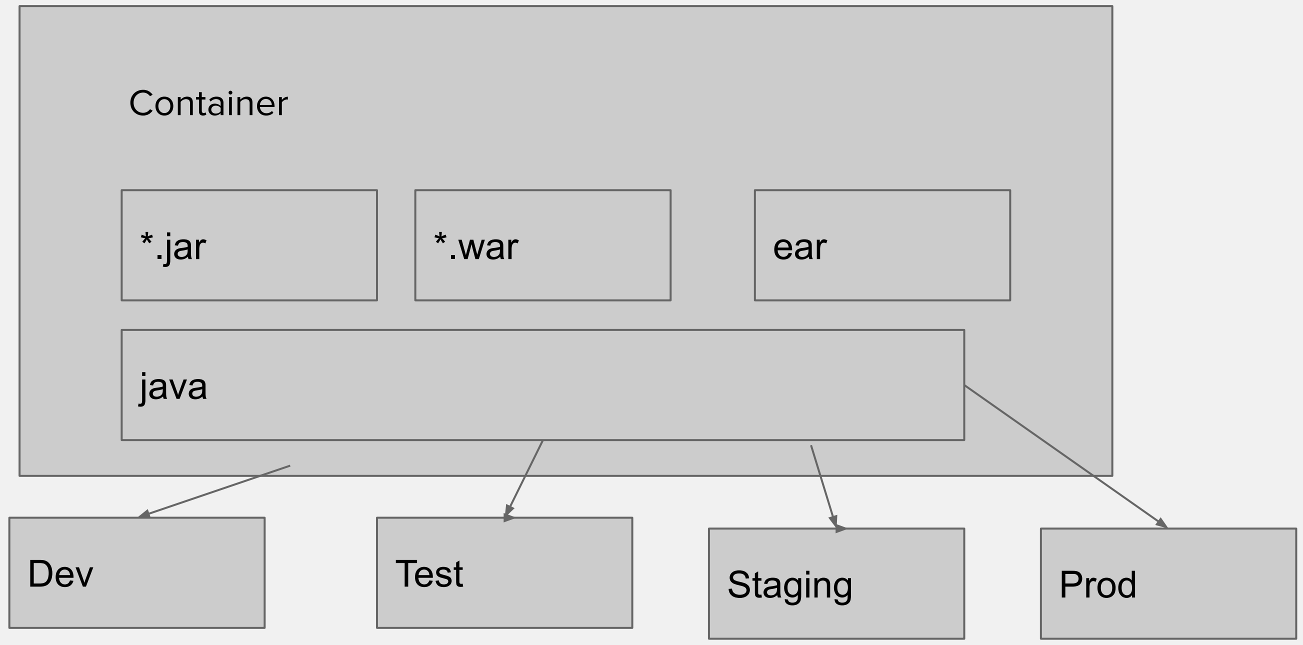


Image Immutability Principle

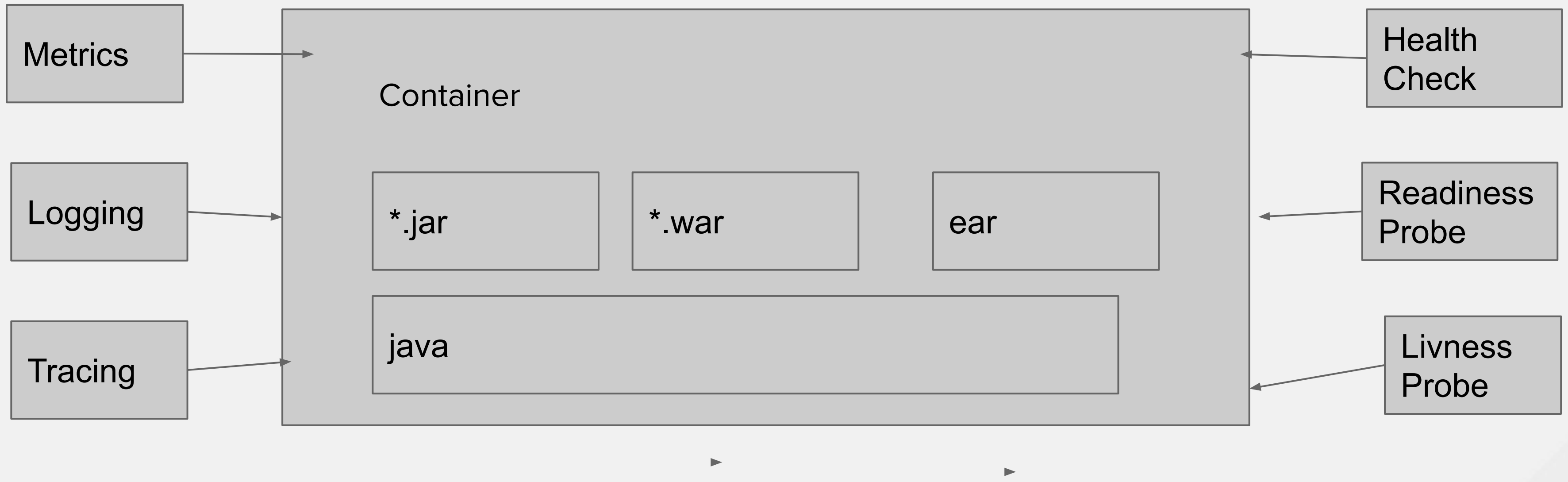
Container should target for all environments.
Dev and Prod Parity



High Observability Principle

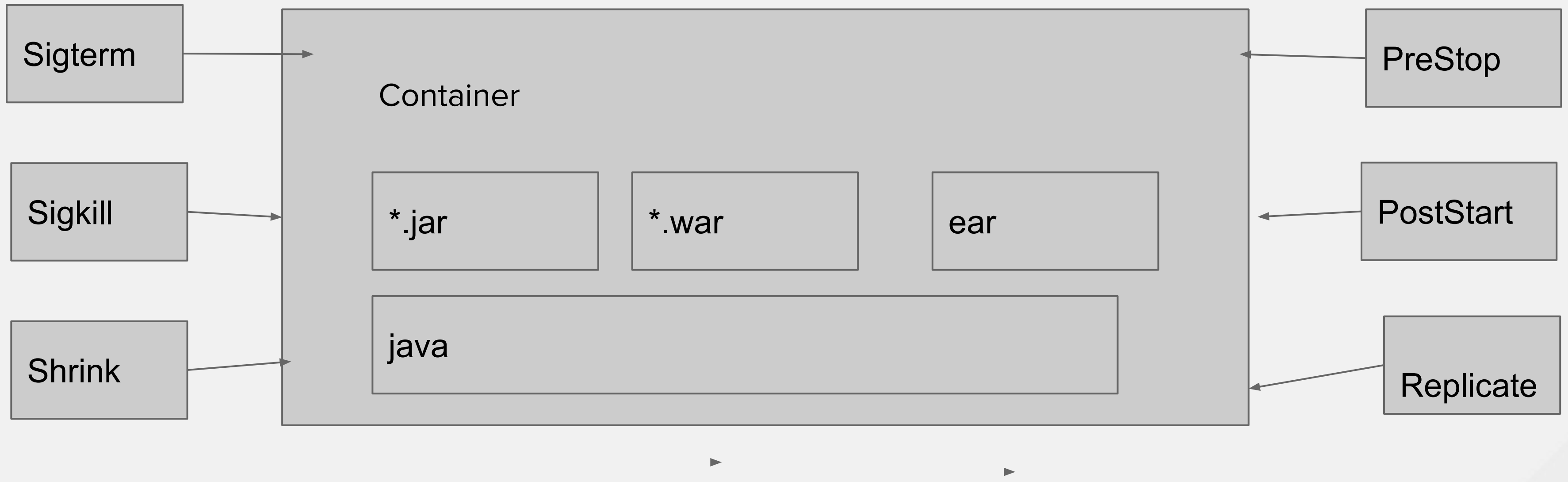
Health Checks

- Liveness probe, Readiness probe for microservices, Nexus, SonarCube, Databases.
- Spring Boot Actuator for Java
- Splunk, CloudWatch, Application Insights for monitoring and alert



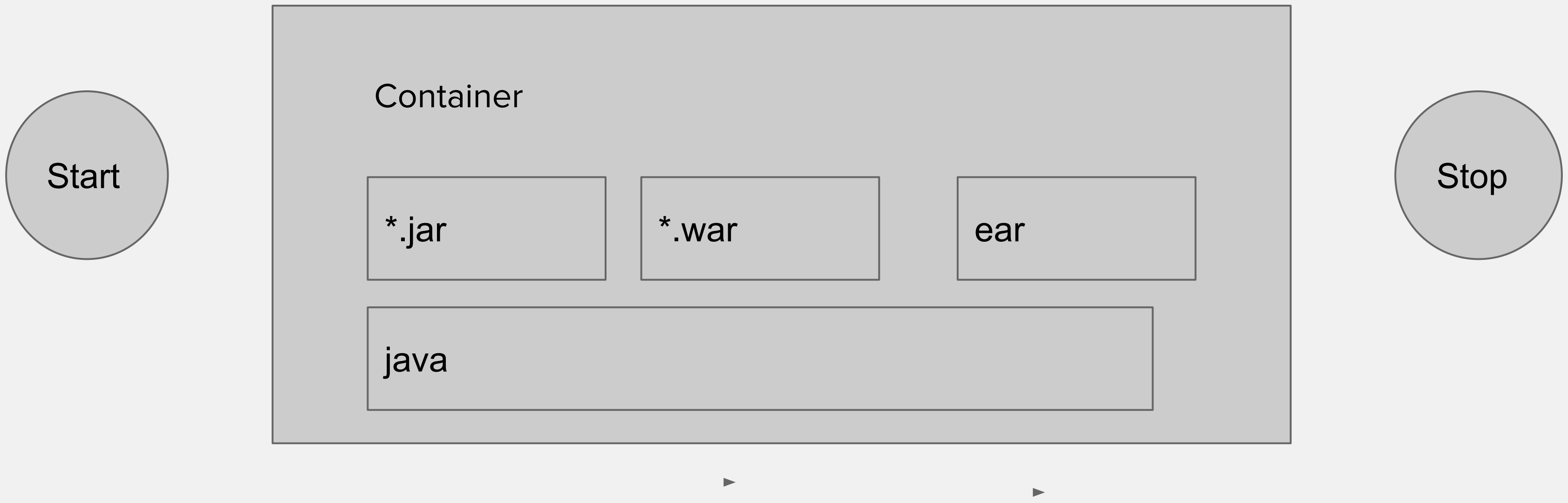
Lifecycle Conformance Principle

- Container should conform to signals coming from the platform.
- Signals include Sigterm, Sigkill, PreStop, PostStart



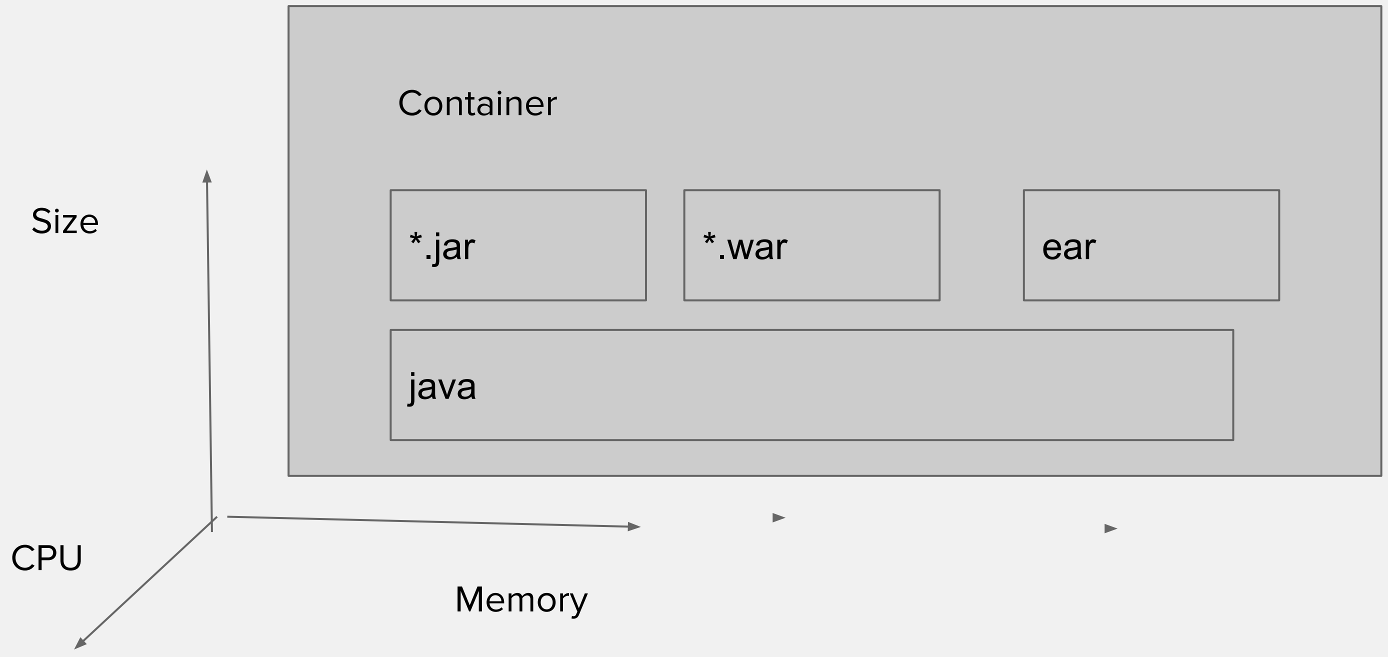
Process Disposability Principle

- A Container can be killed at runtime
- Your application should be depend on a specific instance of your container
- Store application states to databases or Persistence Volume
- Rapid startup and shutdown



Runtime Confinement Principle

- A Container should be viewed with the run time dimensions including Size, Memory, and CPU usages
- Specify these dimensions in the configuration including auto scaling, max and min number of instances. Warm-up and cool-down period of scaling, scaling threshold, scheduling.



Design Patterns

Creation

- Factory Pattern
- Singleton Pattern
- Builder Pattern
- Prototype Pattern

Structure

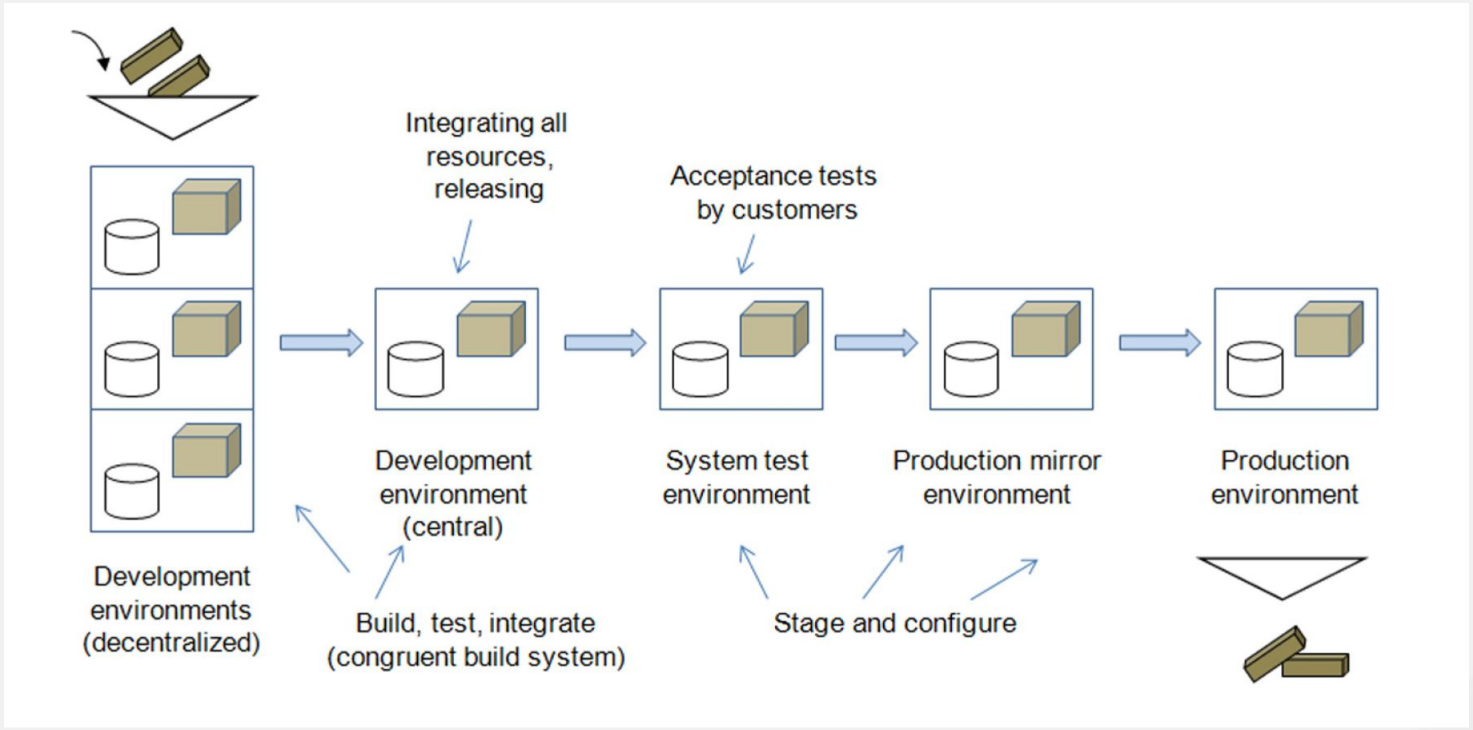
- Adapter Pattern
- Bridge Pattern
- Composite Pattern
- Decorator Pattern
- Facade Pattern
- Proxy Pattern

Behavior

- Chain of Responsibility Pattern
- Command Pattern
- Iterator Pattern
- Observer Pattern
- Strategy Pattern
- Template Pattern
- Visitor Pattern

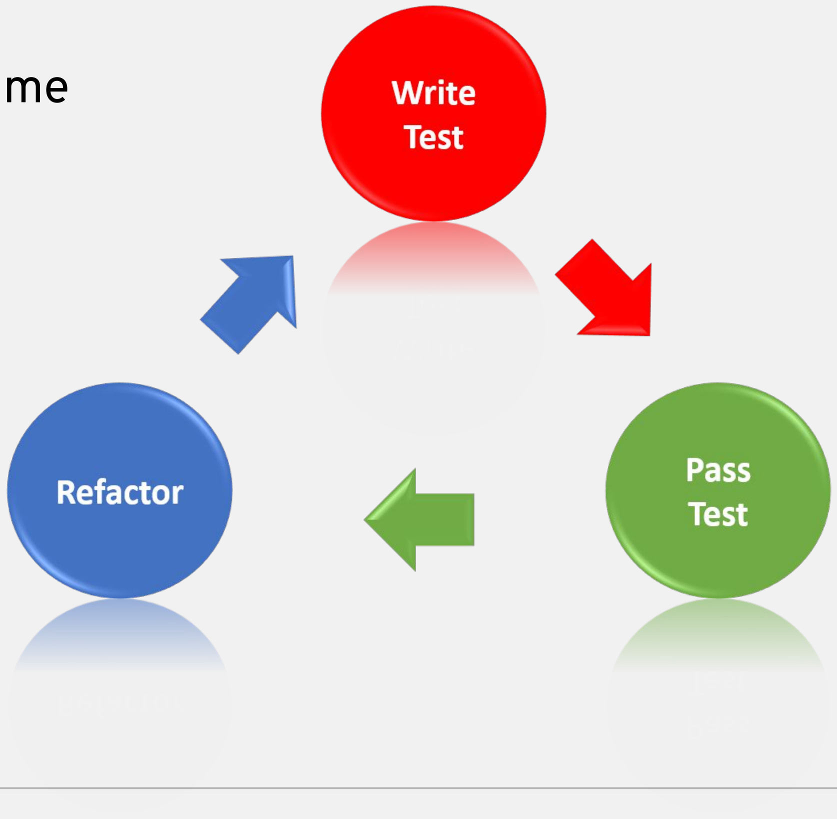
Gated Jenkins Check-In

Jenkins CI / CD pipeline can safeguard your codes with unit tests, integration tests, UI Automation tests, Static Code Analysis, Lint Style Check, Code Coverage,etc

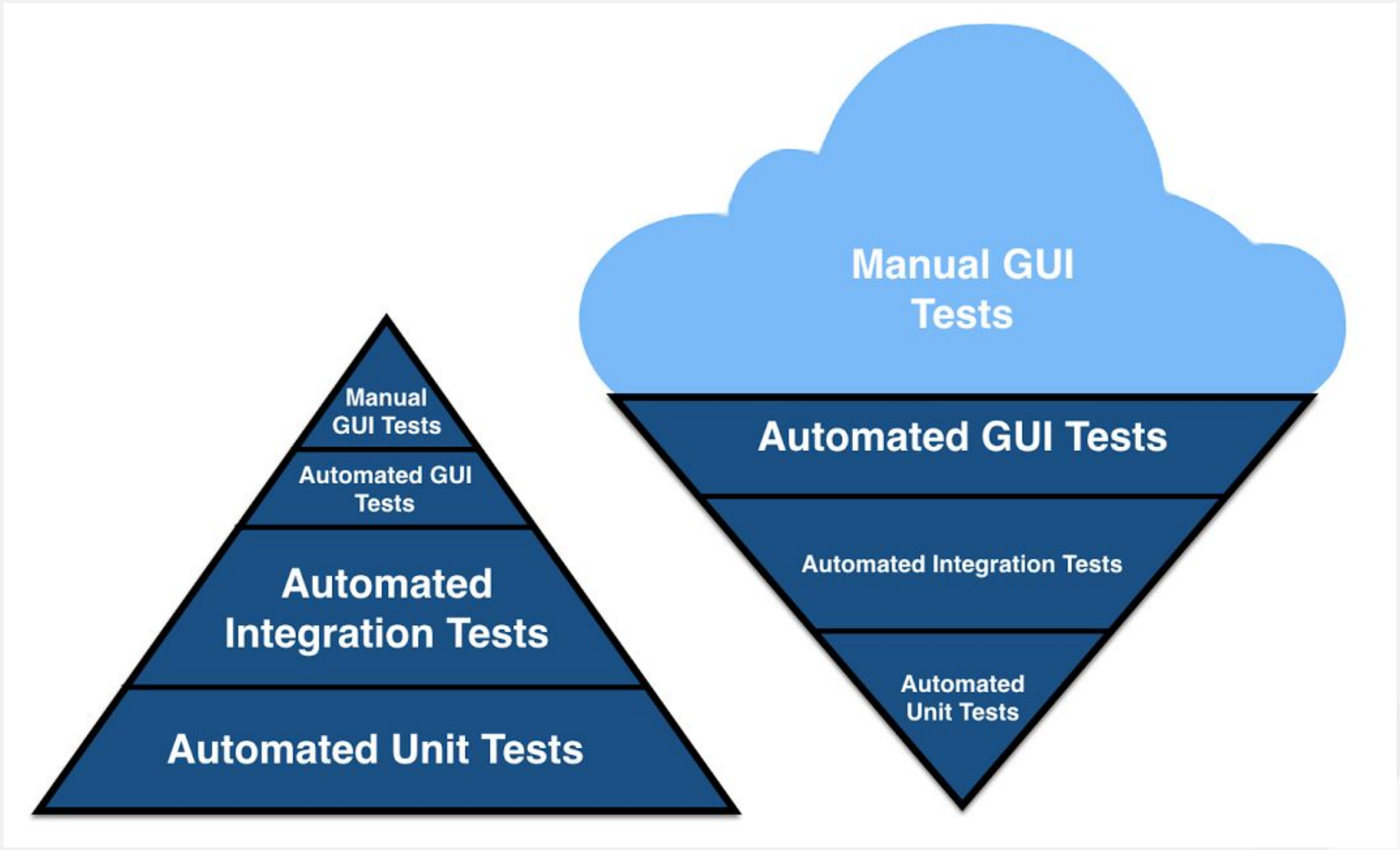

Integrating all resources, releasing
Acceptance tests by customers
Development environments (decentralized)
Development environment (central)
System test environment
Production mirror environment
Production environment
Build, test, integrate (congruent build system)
Stage and configure

Test Driven Development

Benefits:
High code coverage
Codes are intentional (only required codes will be implemented)
Understand business use cases before development.
Reduce bugs
Shorten development time



Test Pyramid



Q & A

Open Discussion