Spring Cloud Configuration

Chapter Content

- 1. Configuration architecture
- 2. Configuration solution open source projects

- Cloud-Based microservice development emphasizes on these principles:
  - Completely separating the configuration of an application from the actual code being deployed
  - Building the server and the application and an immutable image that never changes as it's promoted through your environments
  - Injecting any application configuration information at startup time of the server through either environment variables or through a centralized repository the application's microservices read on startup

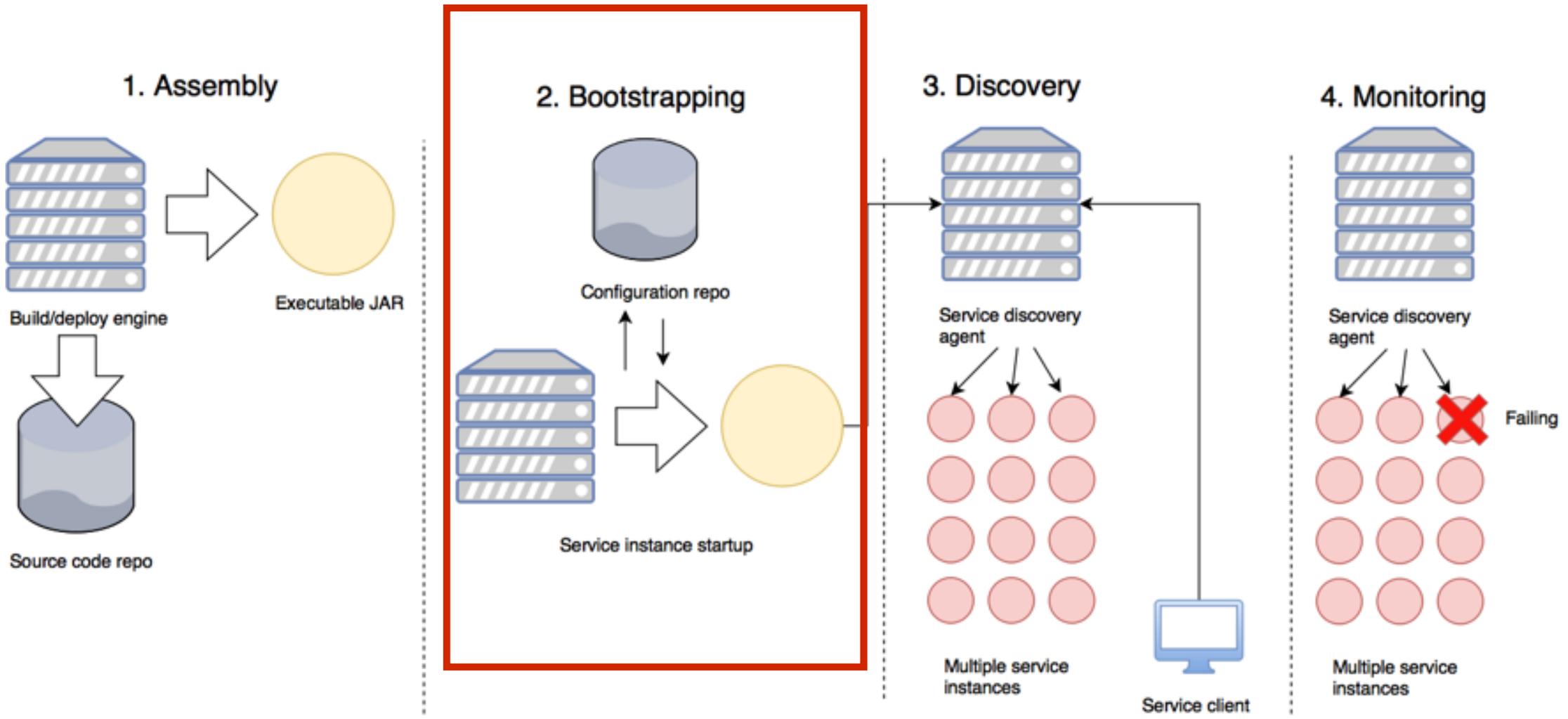
- **Segregate** - Application shouldn't be deployed with the service instance. Instead, configuration information should either be passed to the starting service as environment variables or read from a centralized repository when the service starts

- **Abstract** - Abstract the access of the configuration data behind a service interface. Rather than writing code that directly accesses the service repository, have the application use a REST-Based JSON service to retrieve the configuration data

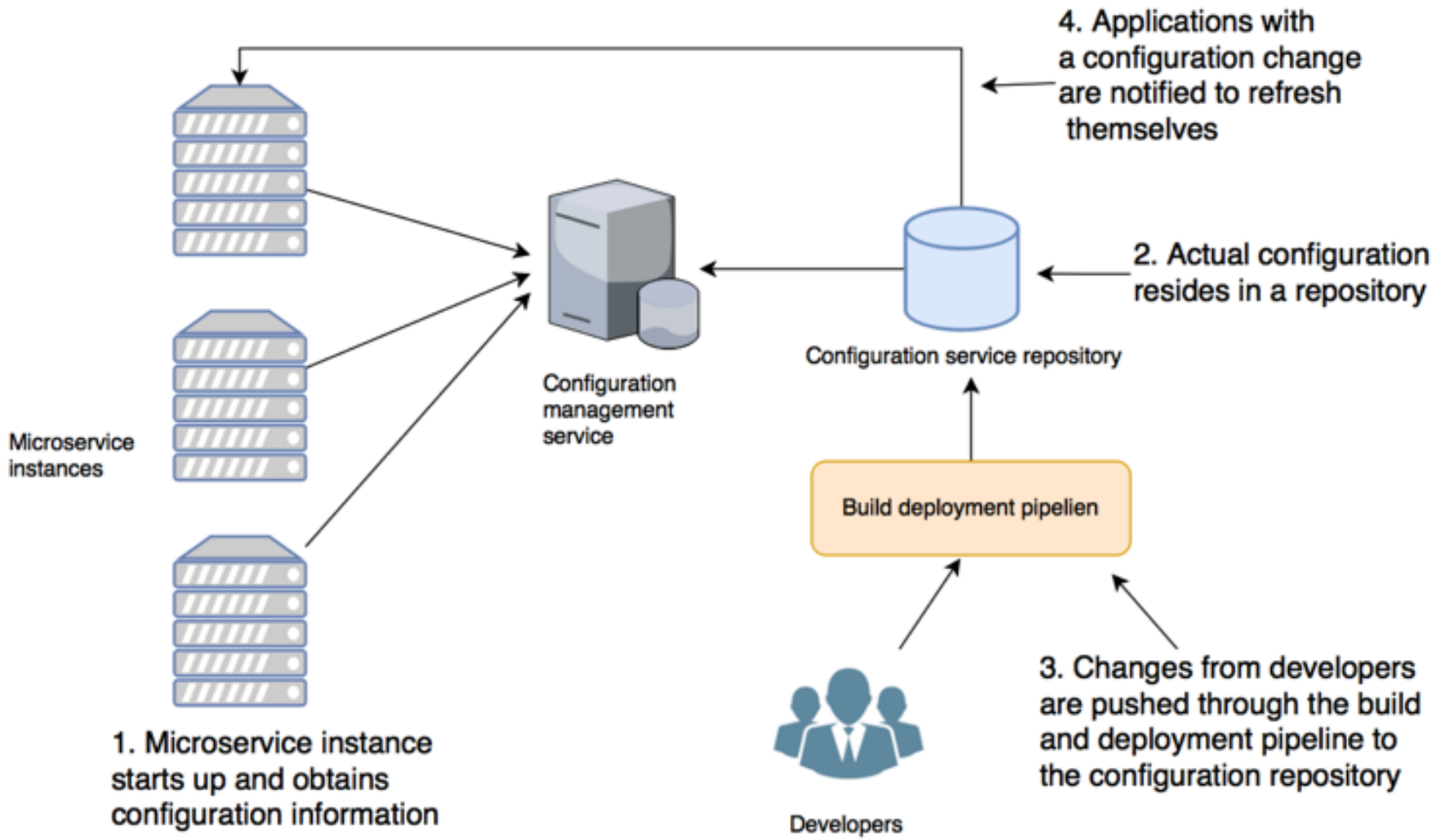
- **Centralize** - A centralized configuration hub makes it efficient to manage hundreds of instances on the cloud. The number of centralized repo should be minimum

- **Harden** - The application configuration should be highly available and redundant

- **Versioning** - The application configuration data should be managed and version controlled


1. Assembly
Build/deploy engine
Executable JAR
Source code repo
2. Bootstrapping
Configuration repo
Service instance startup
3. Discovery
Service discovery agent
Multiple service instances
Service client
4. Monitoring
Service discovery agent
Failing
Multiple service instances

Configuration Architecture



2. Configuration Solution Open Source Projects

Project	Description	Characteristics
Etcd	Open source project written in Go. Used for service discovery and key-value management.	Very fast and scalable Distributable Command-line driven Easy to use and setup
Eureka	Written by Netflix. Extremely battle-tested. Used for both service discovery and key-value management.	Distribute key-value store. Flexible; takes effort to set up Offers dynamic client refresh out of the box
ZooKeeper	An Apache project that offers distributed locking capabilities. Often used as a con-figuration management solution for accessing key-value data.	Oldest, most battle-tested of the solutions The most complex to use Can be used for configuration manage- ment, but should be considered only if you're already using ZooKeeper in other pieces of your architecture
Spring Cloud Configuration Server	An open source project that offers a general configuration management solution with different back ends. It can integrate with Git, Eureka, and Consul as a back end.	on-distributed key/value store Offers tight integration for Spring and non-Spring services Can use multiple back ends for storying configuration data including a shared filesystem, Eureka, Consul, and Git

- Reasons to choose Spring Cloud Configuration:
  - Spring Cloud Configuration server is easy to set up and use
  - Spring Cloud Configuration integrates tightly with Spring Boot. A few simple-to-use annotations will do the job.
  - Spring Cloud Configuration Server offers multiple back ends for storing configuration data.If you are already using tools such as Eureka and Consul, you can plug them into Spring Cloud conjuration server
  - Of all the solutions in table, Spring Cloud Configuration server can integrate directly with the Git source control platform, which make it easy to manage and do version controlling