Pivotal

Load Balancing Patterns

Traditional Load Balancing

- Traditionally, load balancing is performed by:
  - A dedicated appliance such as F5 or ServerIron
  - A software Reverse Proxy, such as Nginx, Apache Httpd or Cloud Foundry GoRouter
  - DNS Round Robin
- Configured manually
- Entry point for HTTP requests from end users. i.e. Public-facing
- Fronts monolithic server instances

Benefits of Traditional Load Balancing

- Simplifies load balancing configuration
- Abstracts load balancing behavior separate from applications
- Consistency in load balancing algorithms

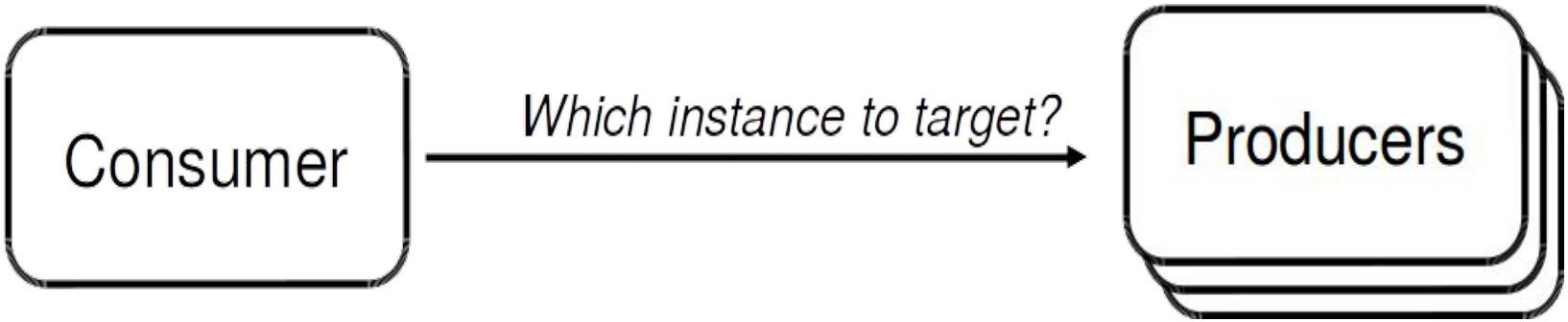
Drawbacks of Traditional Load Balancing

- Manual configuration does not scale, works against elastic scaling requirements
- Extra network hops and associated latency between consumers and providers
- Limited ability to tune timeout/retry characteristics specific to app

Client Load Balancing in a Cloud Native Application

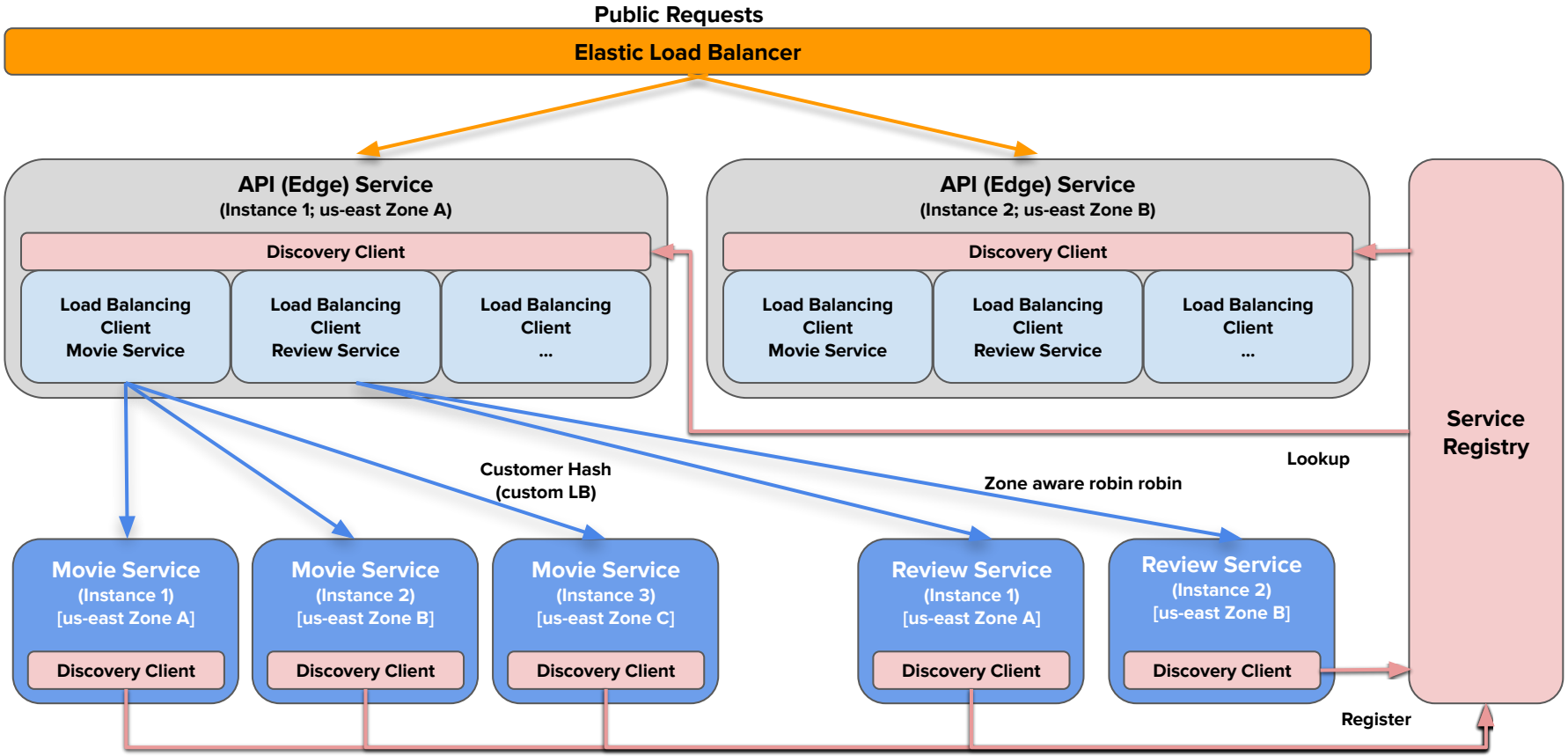
- Embed load balancing logic in consumer (caller)
- Configuration is dynamic and automatic
- Not public-facing
- Load balancing is between services (inter-service)
- Server Pool endpoints are IP and Port address combinations

Service Instances are scaled out



A service registry lookup yields multiple service instances for a given service name

Inter-service Load Balancing



Benefits of Client Load Balancing

- Tailor fit load balancing algorithm to application needs
- Developer can control algorithm
- Reduced hops when using point-to-point communications between *Consumers* and *Producers*
- A side effect of using it can provide better security when consumers and providers are running on a container orchestrator

Drawbacks of Client Load Balancing

- Addition burden to developers when wiring or configuring client load balancing behavior
- Potentially runtime overhead if using software client load balancing components inside of *Consumers*.