Where is my cache?

Architectural patterns for caching microservices by example

•••

Rafał Leszko
@RafalLeszko
Hazelcast

About me

- Cloud Software Engineer at Hazelcast
- Worked at Google and CERN
- Author of the book "Continuous Delivery with Docker and Jenkins"
- Trainer and conference speaker
- Live in Kraków, Poland

About Hazelcast

- Distributed Company
- Open Source Software
- 140+ Employees
- Hiring (Remote)!
- Recently Raised $21M
- Products:
  - Hazelcast IMDG
  - Hazelcast Jet
  - Hazelcast Cloud

hazelcast

@Hazelcast

www.hazelcast.com

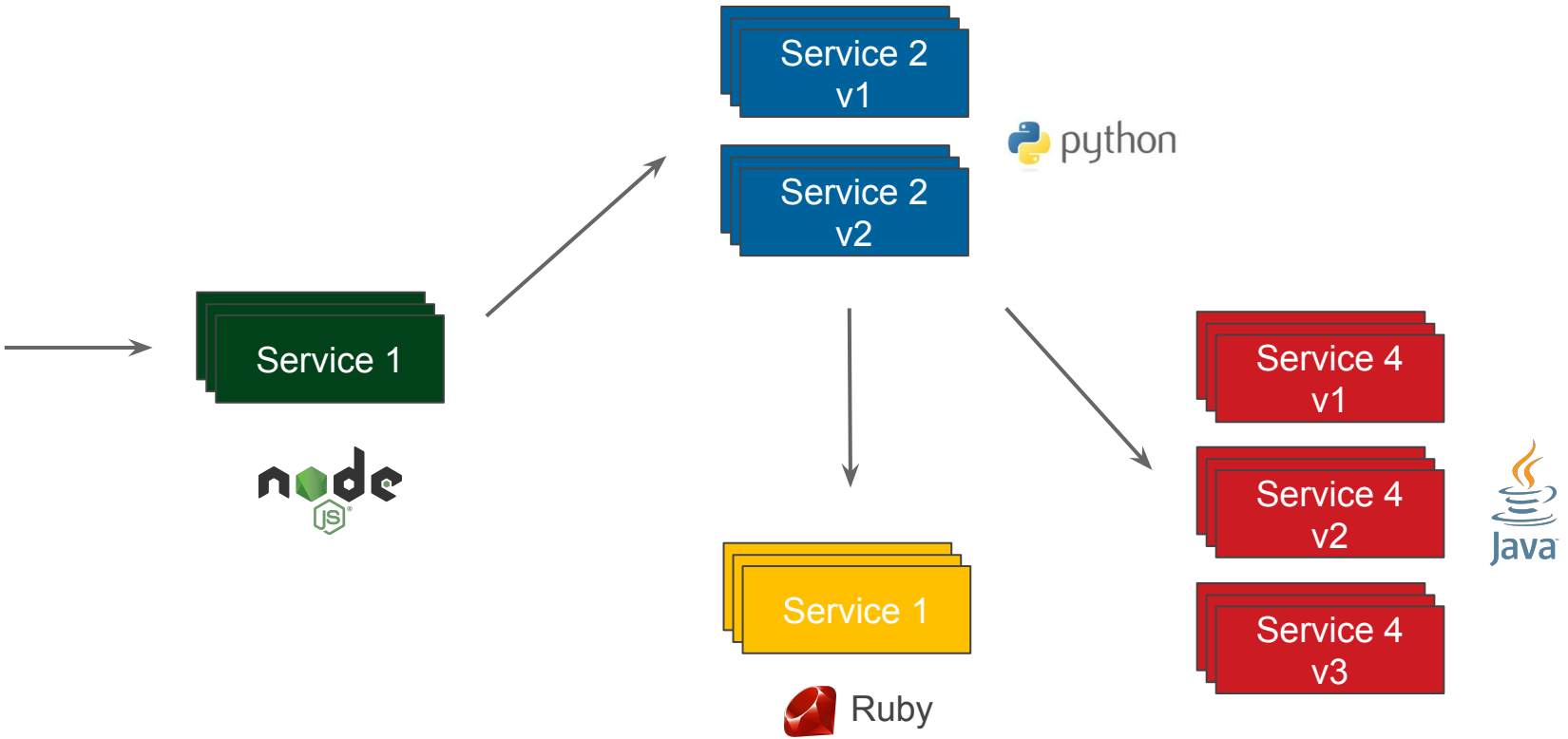
Agenda

- Introduction
- Caching Architectural Patterns
  - Embedded
  - Embedded Distributed
  - Client-Server
  - Cloud
  - Sidecar
  - Reverse Proxy
  - Reverse Proxy Sidecar
- Summary

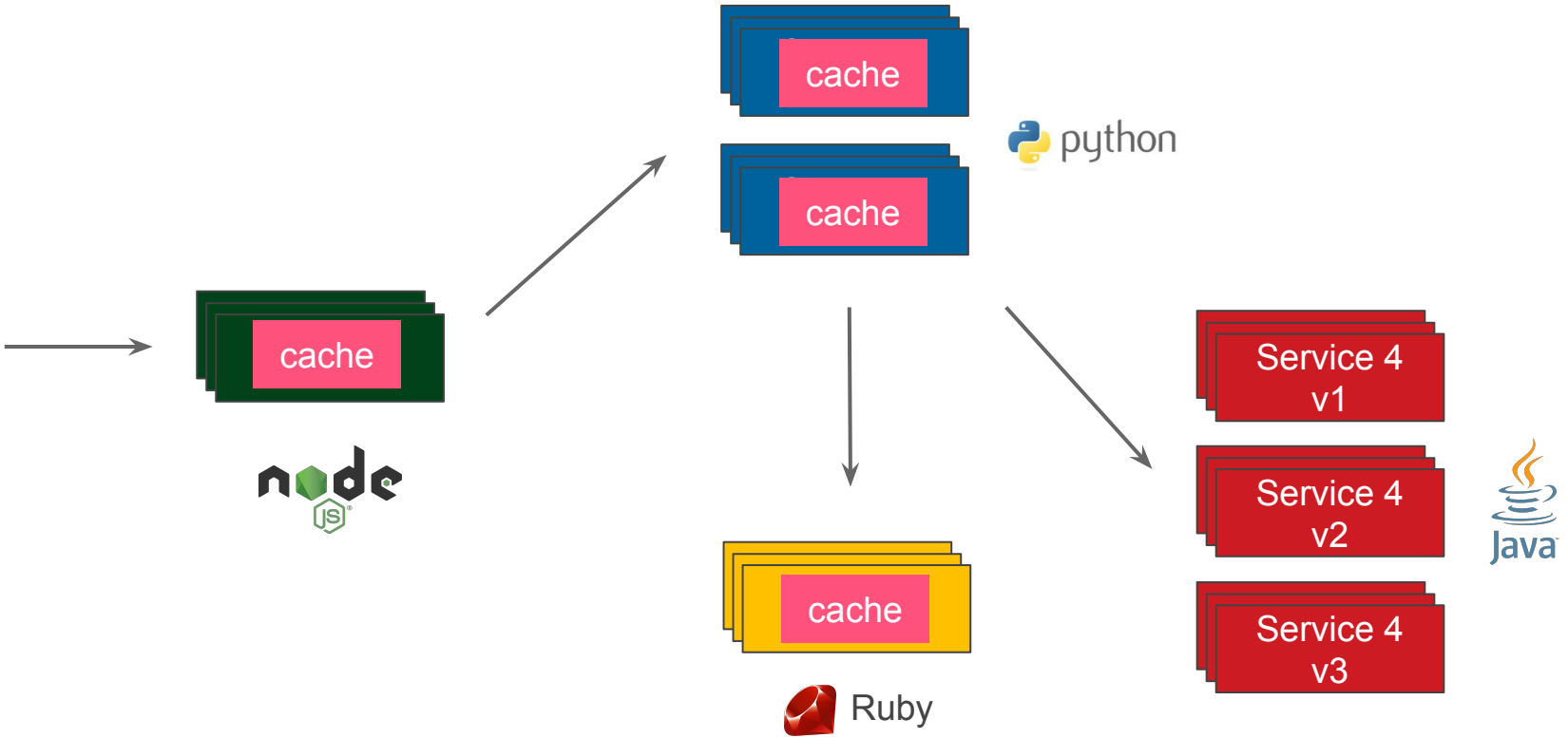
Why Caching?

- Performance
  - Decrease latency
  - Reduce load
- Resilience
  - High availability
  - Lower downtime

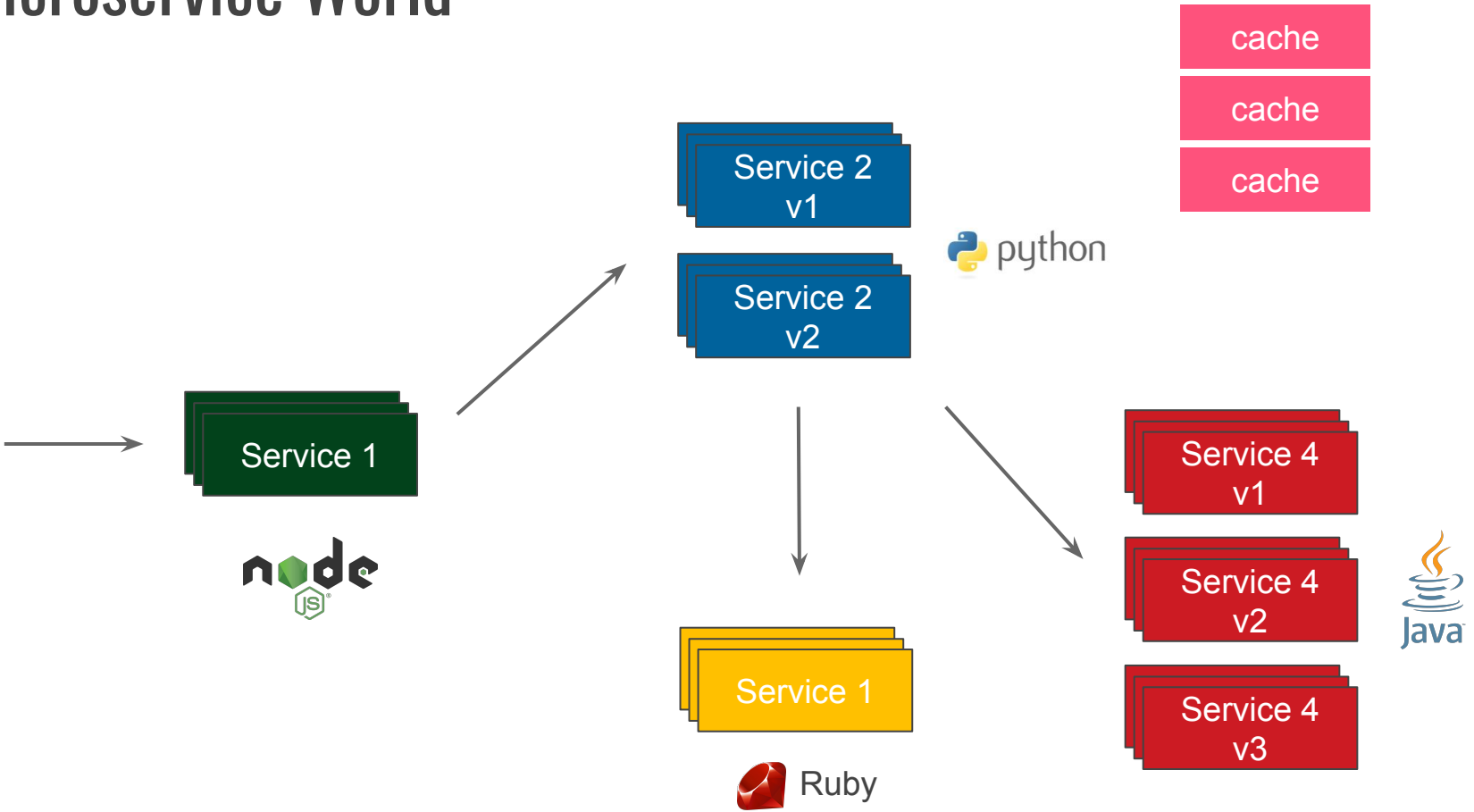
Microservice World



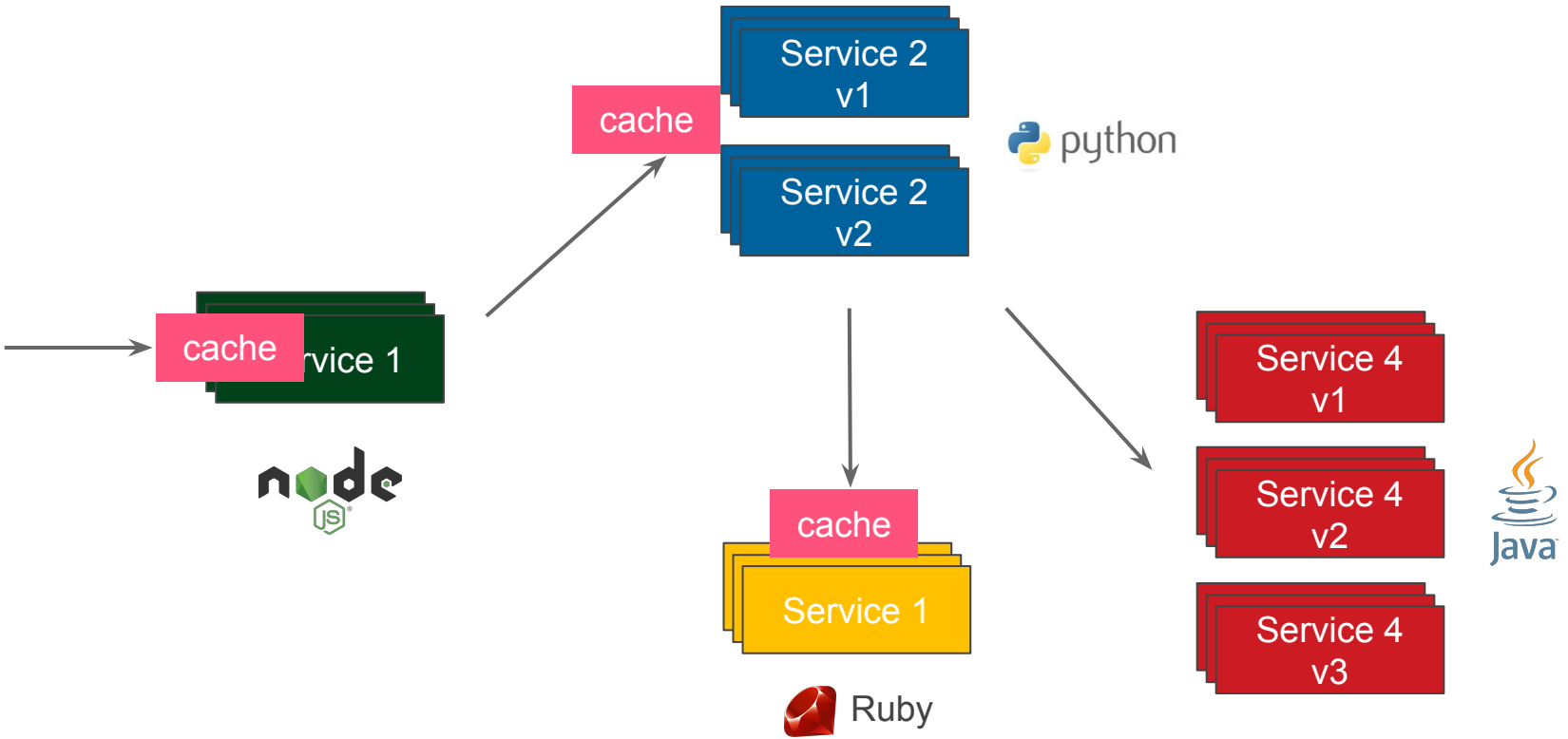
Microservice World



Microservice World

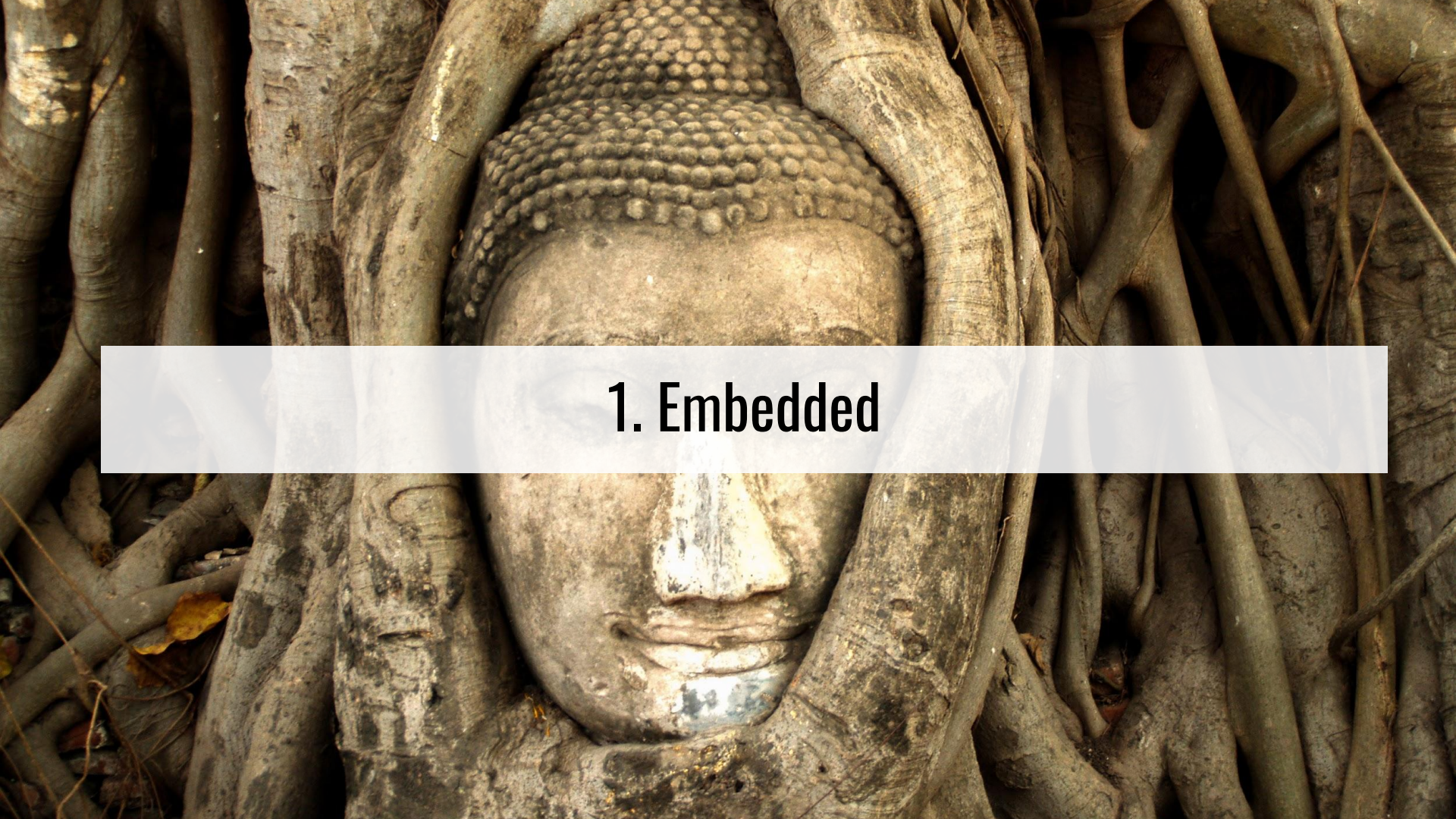


Microservice World



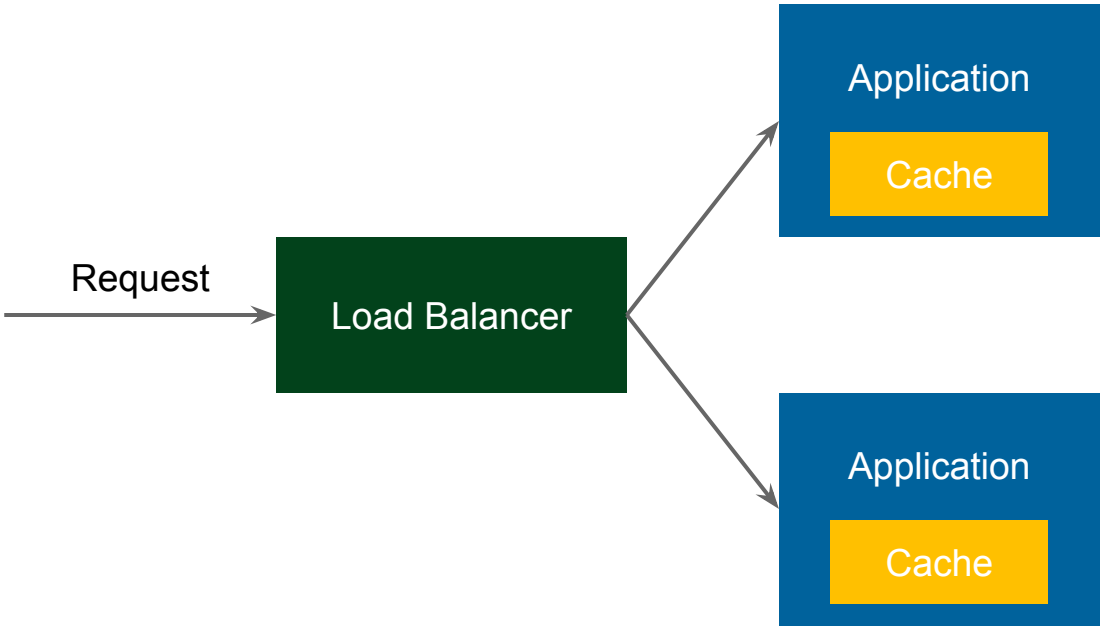
Agenda

- Introduction ✅
- Caching Architectural Patterns
  - Embedded
  - Embedded Distributed
  - Client-Server
  - Cloud
  - Sidecar
  - Reverse Proxy
  - Reverse Proxy Sidecar
- Summary



1. Embedded

Embedded Cache



Embedded Cache (Pure Java implementation)

```
private ConcurrentHashMap<String, String> cache =
                                        new ConcurrentHashMap<>();

private String processRequest(String request) {
    if (cache.contains(request)) {
        return cache.get(request);
    }
    String response = process(request);
    cache.put(request, response);
    return response;
}
```

Embedded Cache (Pure Java implementation)

```
private ConcurrentH      <String, String>      =
                                  n           tHashMap<>();

private String proces                   ) {
    if (cache.contains(rec
        return cache.ge
    }
    String respons            (request)
    cache.put(reque          ponse);
    return response;
}
```

Java Collection is not a Cache!

- No Eviction Policies
- No Max Size Limit (OutOfMemoryError)
- No Statistics
- No built-in Cache Loaders
- No Expiration Time
- No Notification Mechanism

Embedded Cache (Java libraries)


```
CacheBuilder.newBuilder()
    .initialCapacity(300)
    .expireAfterAccess(Duration.ofMinutes(10))
    .maximumSize(1000)
    .build();
```

Embedded Cache (Java libraries)


```
CacheBuilder.newBuilder()
    .initialCapacity(300)
    .expireAfterAccess(Duration.ofMinutes(10))
    .maximumSize(1000)
    .build();
```

Caching Application Layer


```
@Service
public class BookService {

    @Cacheable("books")
    public String getBookNameByIsbn(String isbn) {
        return findBookInSlowSource(isbn);
    }

}
```

Caching Application Layer

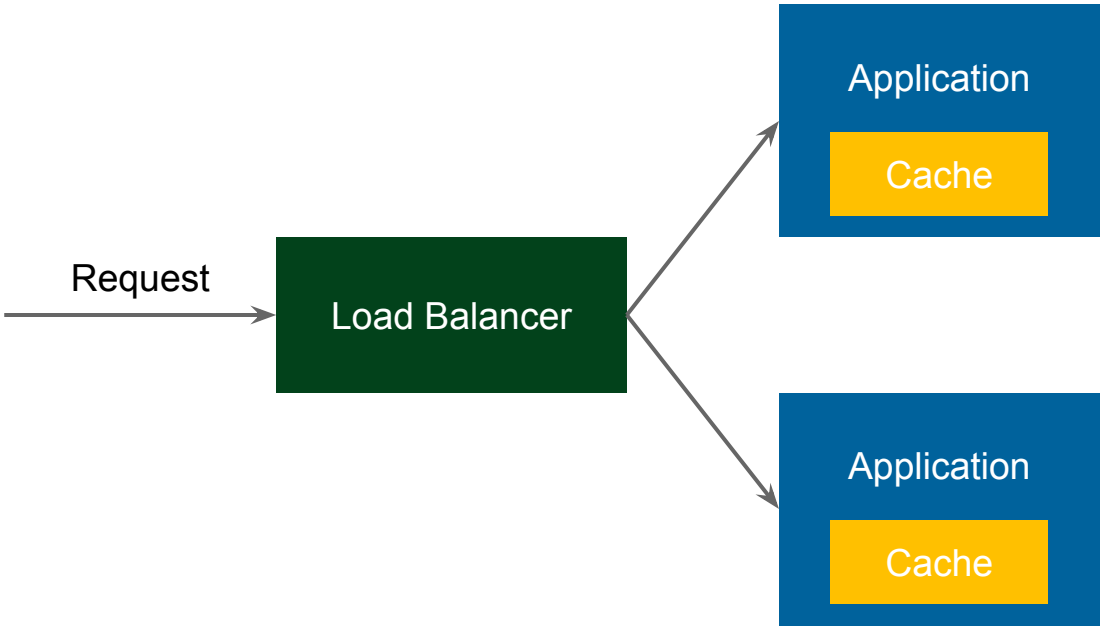
```
@Service
public class BookService {

    @Cacheable("books")
    public String getBookNameByIsbn(String isbn) {
        return findBookInSlowSource(isbn);
    }

}
```

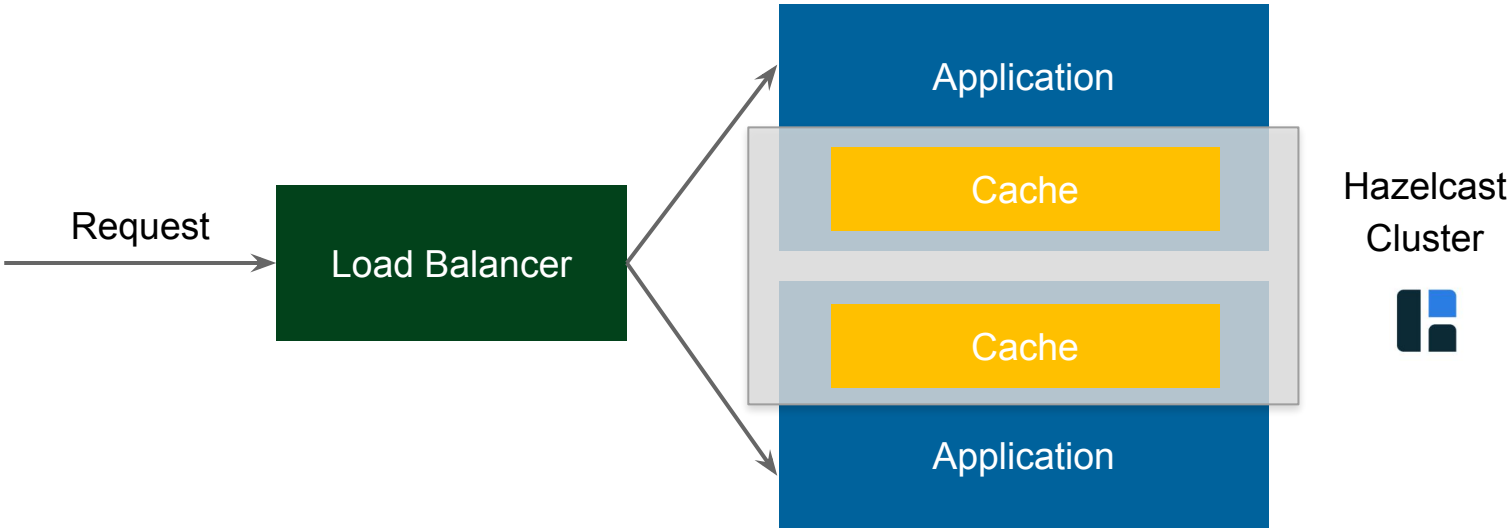
Be Careful, Spring uses ConcurrentHashMap by default!

Embedded Cache



1*. Embedded Distributed

Embedded Distributed Cache



Embedded Distributed Cache (Spring with Hazelcast)

```
@Configuration
public class HazelcastConfiguration {

    @Bean
    CacheManager cacheManager() {
        return new HazelcastCacheManager(
                        Hazelcast.newHazelcastInstance());
    }

}
```

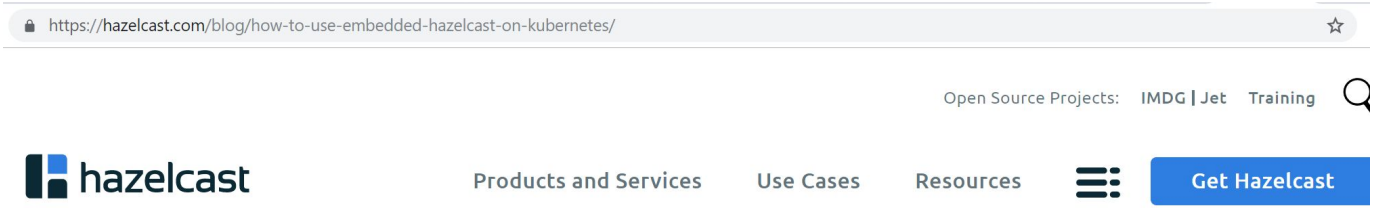
Hazelcast Discovery Plugins

Google Cloud Platform

RED HAT®
OPENSHIFT

kubernetes

Hazelcast Discovery Plugins



How to Use Embedded Hazelcast on Kubernetes

Rafal Leszko | February 06, 2019

Share

Hazelcast IMDG is a perfect fit for your (micro)services running on Kubernetes since it can be used in the embedded mode and therefore scale in and out together with your service replicas. This blog post presents a step-by-step description of how to embed Hazelcast into a Spring Boot application and deploy it in the Kubernetes cluster. The source code for this example can be found here.

Hazelcast Discovery Plugins

https://hazelcast.com/blog/hazelcast-auto-discovery-with-eureka/

https://hazelcast.com/blog/

hazelcast

Products and Services Use Cases Resources Get Hazelcast

hazelcast

How to Use Hazelcast Auto-Discovery with Eureka

Rafal Leszko | April 24, 2019

How to Us

Rafal Leszko | February 06, 2

Share

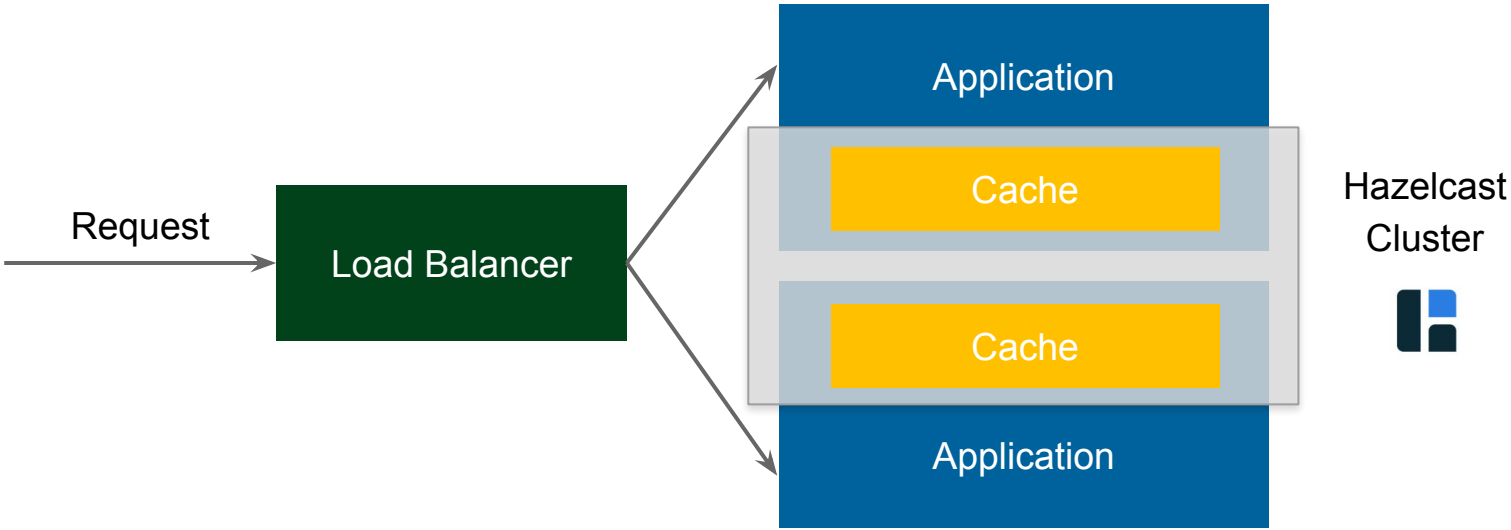
Share

Hazelcast IMDG supports auto-discovery for many different environments. Since we introduced the generic discovery SPI, a lot of plugins were developed so you can use Hazelcast seamlessly on Kubernetes, AWS, Azure, GCP, and more. Should you need a custom plugin, you are also able to create your own.

Hazelcast IMDG is a pe
therefore scale in and
Hazelcast into a Spring
here.

If your infrastructure is not based on any popular Cloud environment, but you still want to take advantage of the dynamic discovery rather than static IP configuration, you can set up your service registry. One of the more popular

Embedded Distributed Cache



Embedded Cache

Pros

- Simple configuration / deployment
- Low-latency data access
- No separate Ops Team needed

Cons

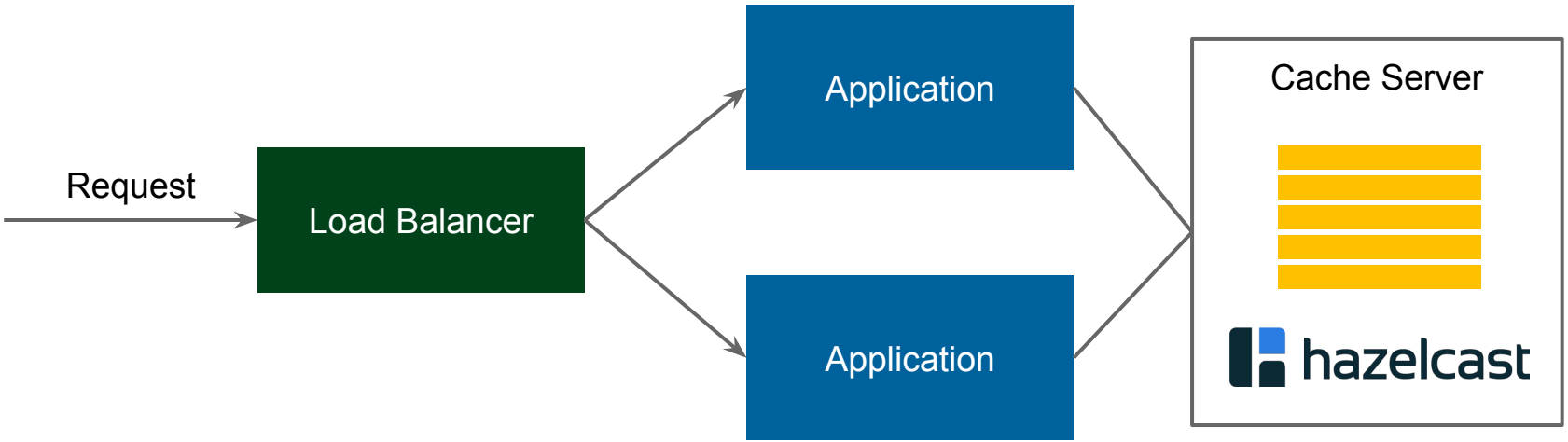
- Not flexible management (scaling, backup)
- Limited to JVM-based applications
- Data collocated with applications

Agenda

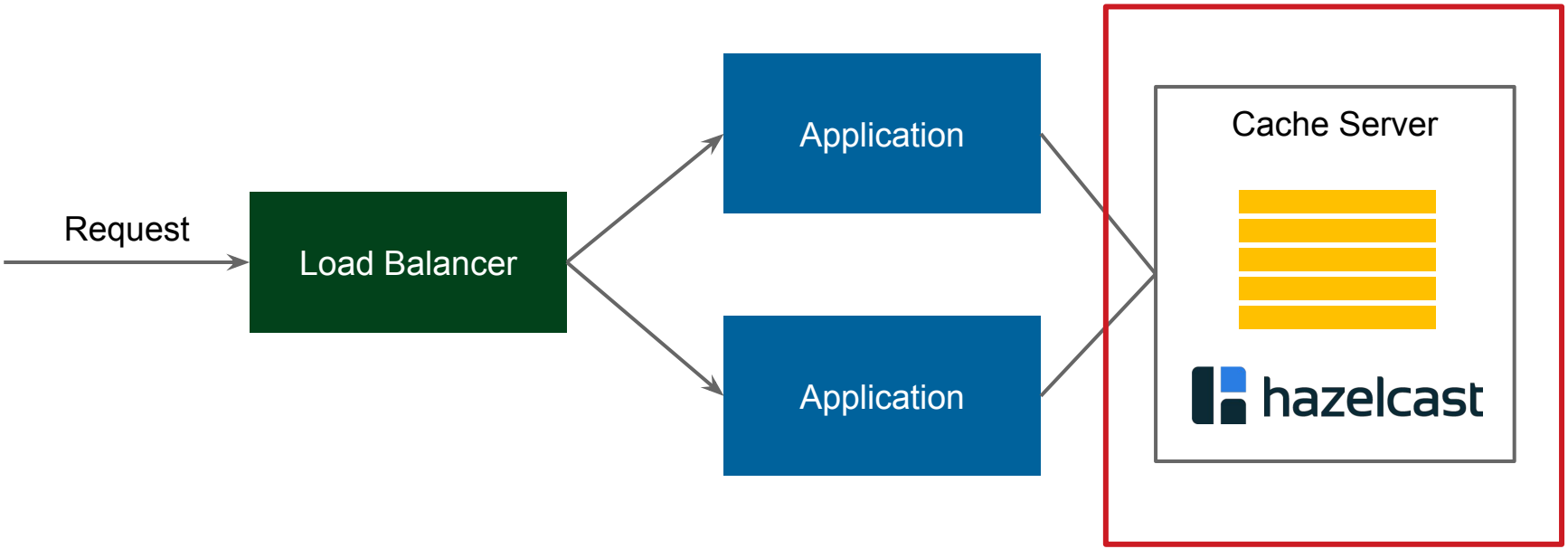
- Introduction ✅
- Caching Architectural Patterns
  - Embedded ✅
  - Embedded Distributed ✅
  - Client-Server
  - Cloud
  - Sidecar
  - Reverse Proxy
  - Reverse Proxy Sidecar
- Summary

2. Client-Server

Client-Server Cache



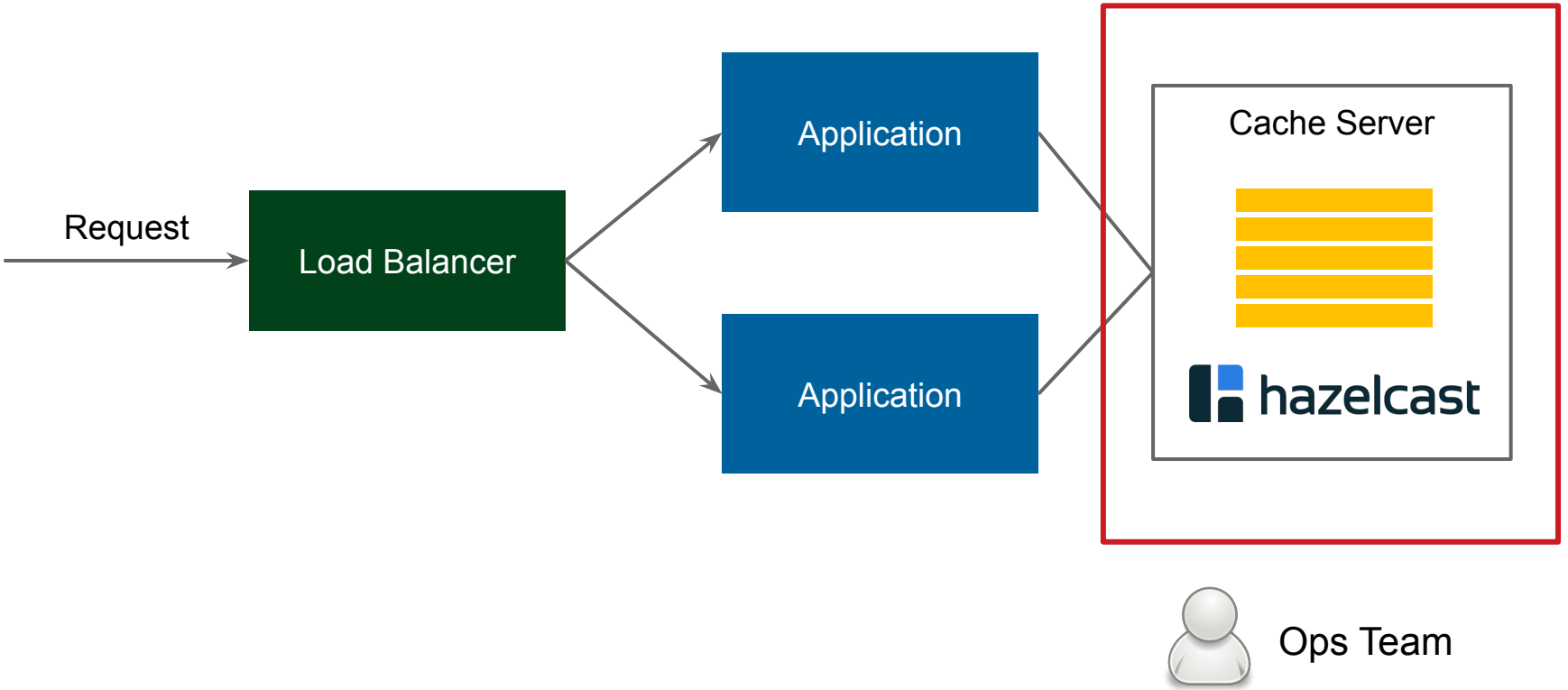
Client-Server Cache



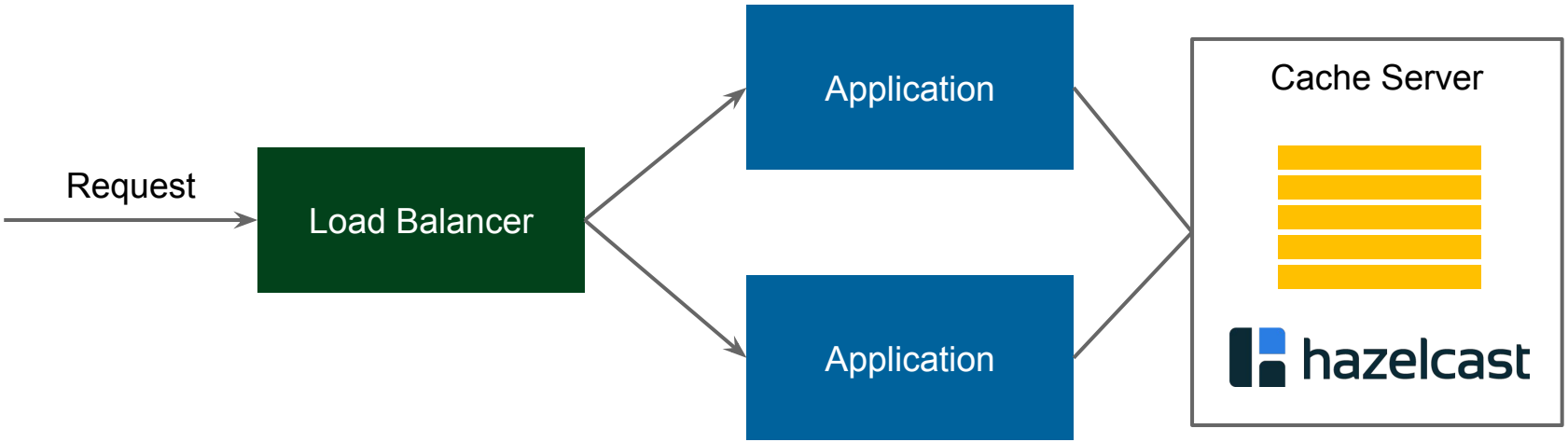
Client-Server Cache

Separate Management:

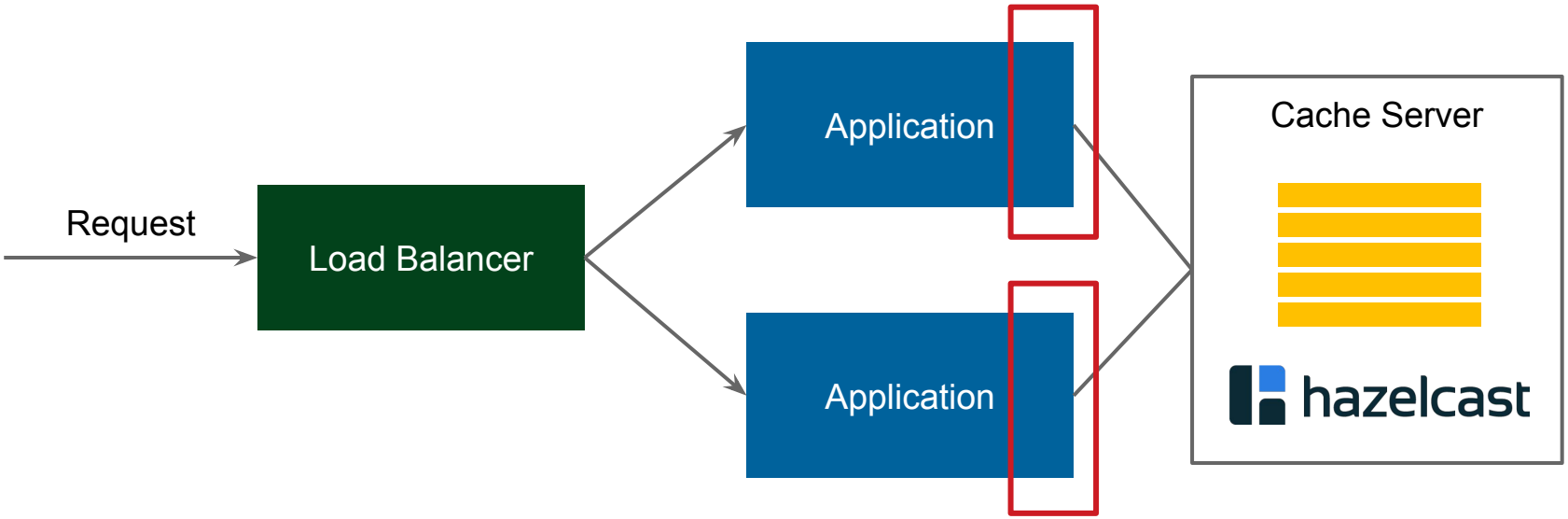
- backups
- (auto) scaling
- security



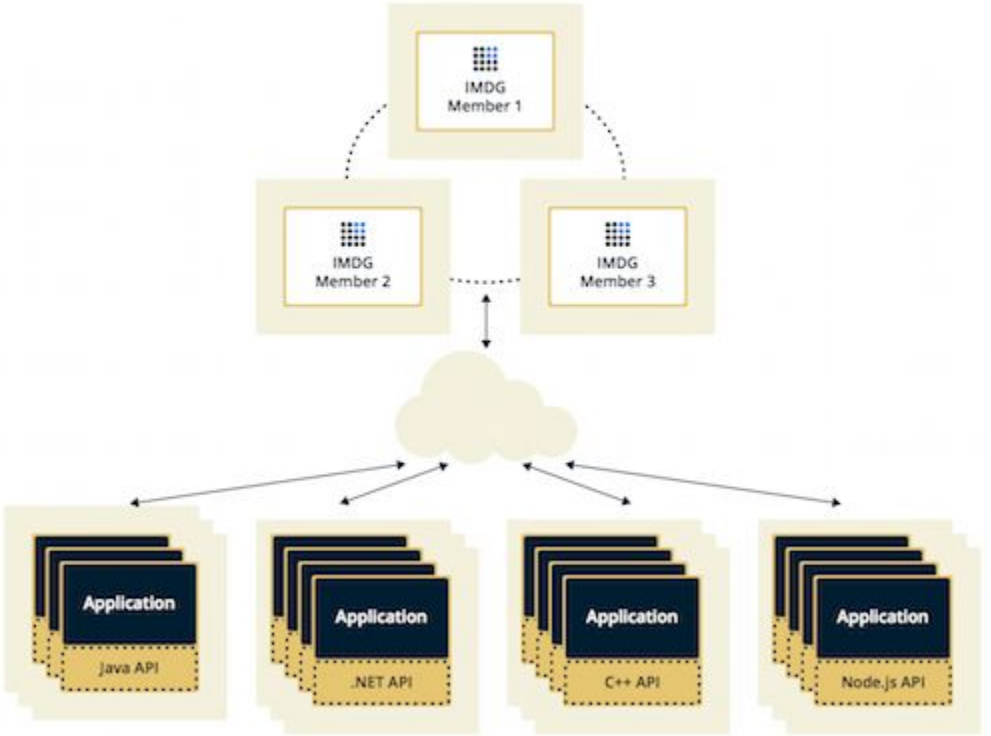
Client-Server Cache



Client-Server Cache



Client-Server Cache



Client-Server Cache

Client-Server Cache

Starting Hazelcast Cache Server (standalone)

```
$ ./start.sh
```

Client-Server Cache

Starting Hazelcast Cache Server (Kubernetes)

```
$ helm install hazelcast/hazelcast
```

Client-Server Cache

Starting Hazelcast Cache Server (Kubernetes)

```
$ helm install hazelcast/hazelcast
```

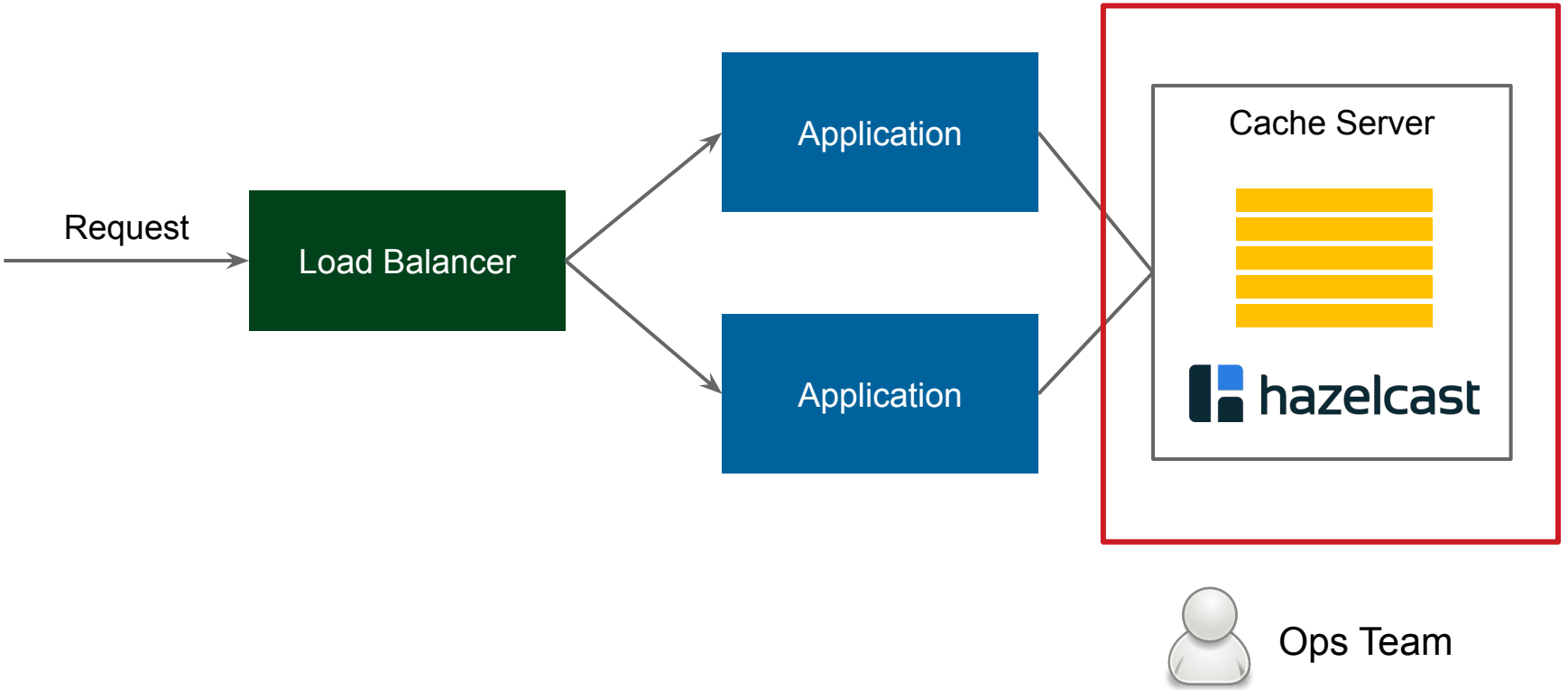
Hazelcast Client (Kubernetes):

```
@Configuration
public class HazelcastClientConfiguration {
    @Bean
    CacheManager cacheManager() {
        ClientConfig clientConfig = new ClientConfig();
        clientConfig.getNetworkConfig().getKubernetesConfig()
                                        .setEnabled(true);
        return new HazelcastCacheManager(HazelcastClient
                                .newHazelcastClient(clientConfig));
    }
}
```

Client-Server Cache

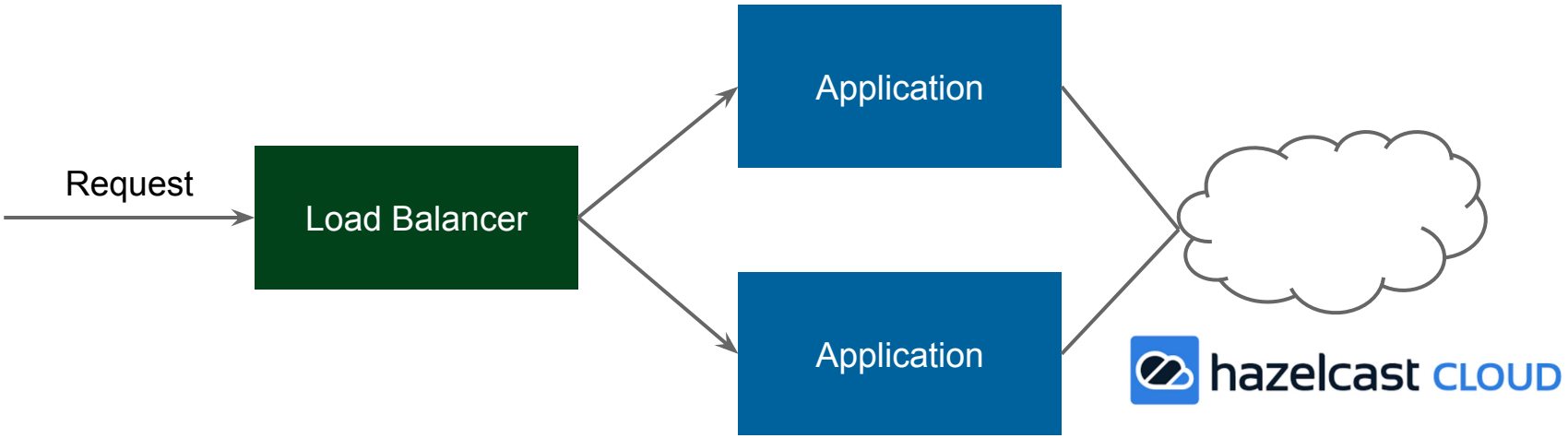
Separate Management:

- backups
- (auto) scaling
- security



2*. Cloud

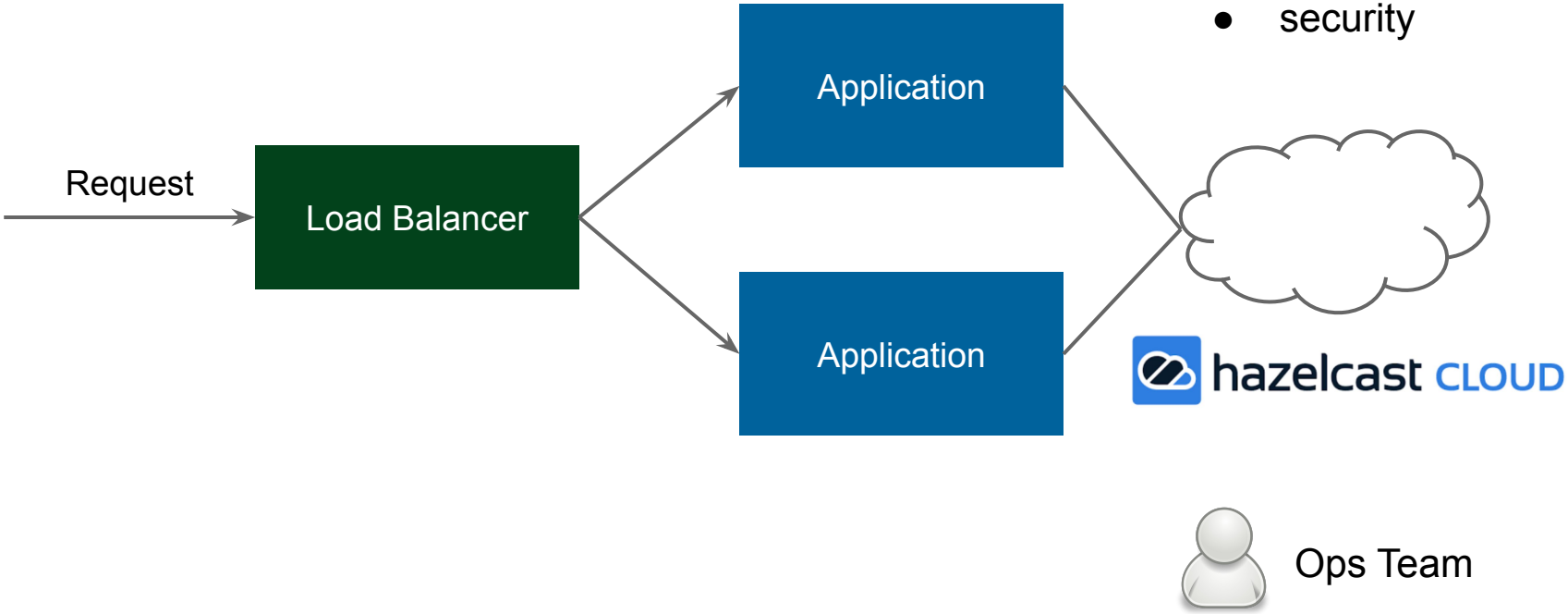
Cloud (Cache as a Service)



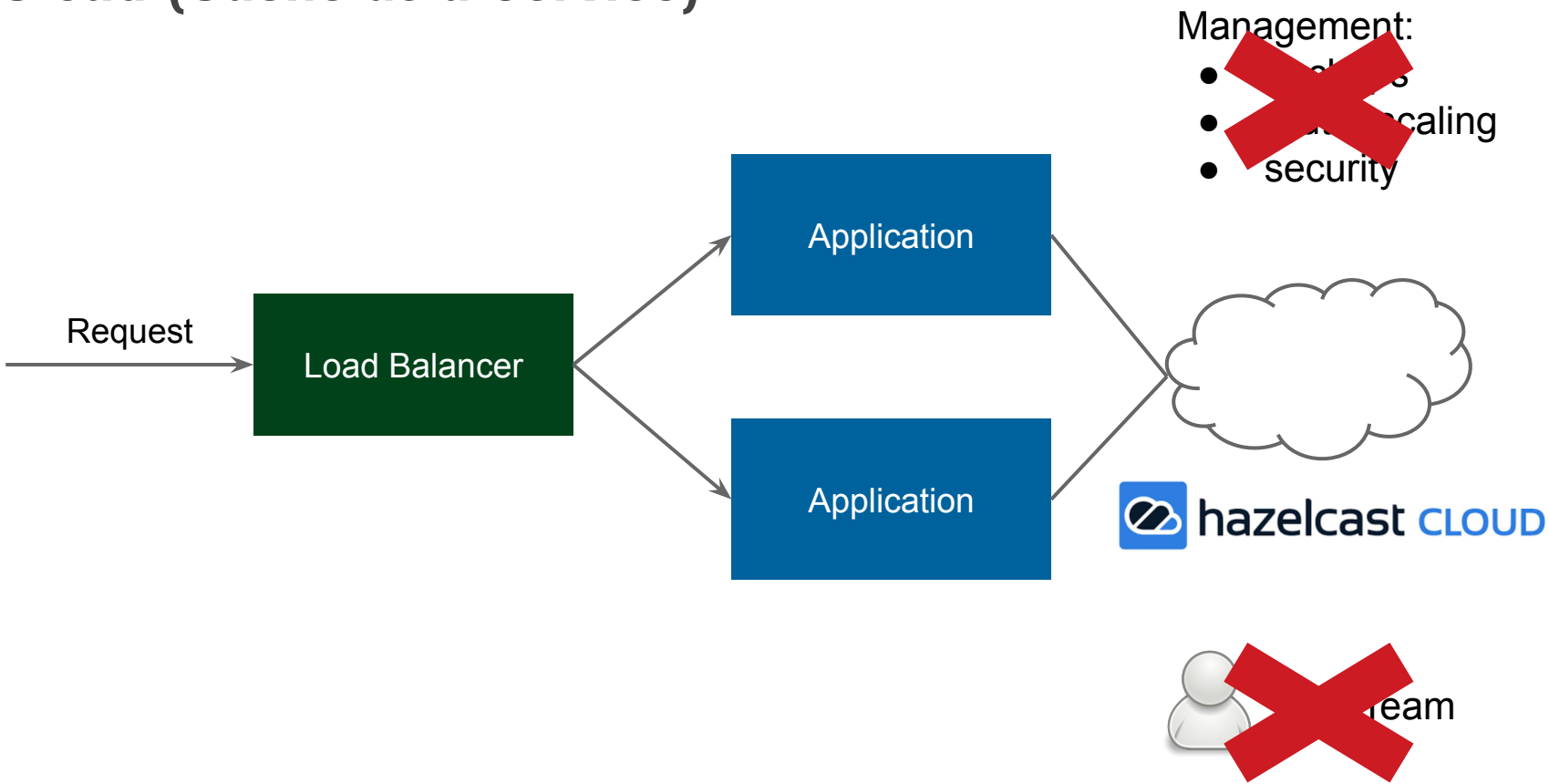
Cloud (Cache as a Service)

Management:

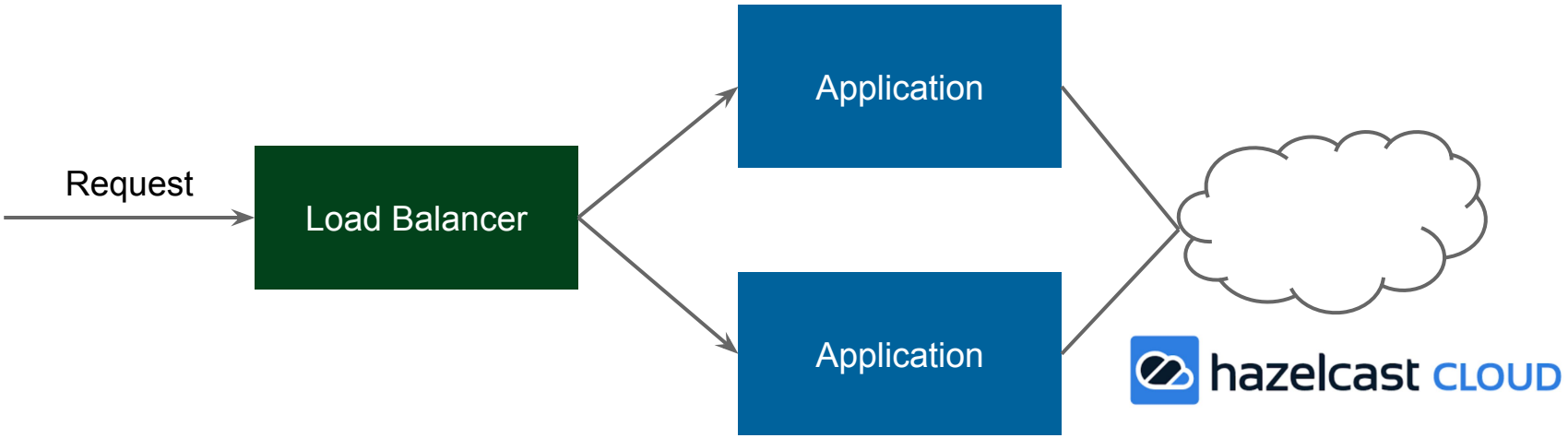
- backups
- (auto) scaling
- security



Cloud (Cache as a Service)



Cloud (Cache as a Service)



Cloud (Cache as a Service)

```
@Configuration
public class HazelcastCloudConfiguration {
    @Bean
    CacheManager cacheManager() {
        ClientConfig clientConfig = new ClientConfig();
        clientConfig.getNetworkConfig().getCloudConfig()
            .setEnabled(true)
            .setDiscoveryToken("KSXFDTi5HXPJGR0wRAjLgKe45tvEEhd");
        clientConfig.setGroupConfig(
            new GroupConfig("test-cluster", "b2f984b5dd3314"));

        return new HazelcastCacheManager(
                HazelcastClient.newHazelcastClient(clientConfig));
    }
}
```

DEMO

cloud.hazelcast.com

Client-Server (Cloud) Cache

Pros

- Data separate from applications
- Separate management (scaling, backup)
- Programming-language agnostic

Cons

- Separate Ops effort
- Higher latency
- Server network requires adjustment (same region, same VPC)

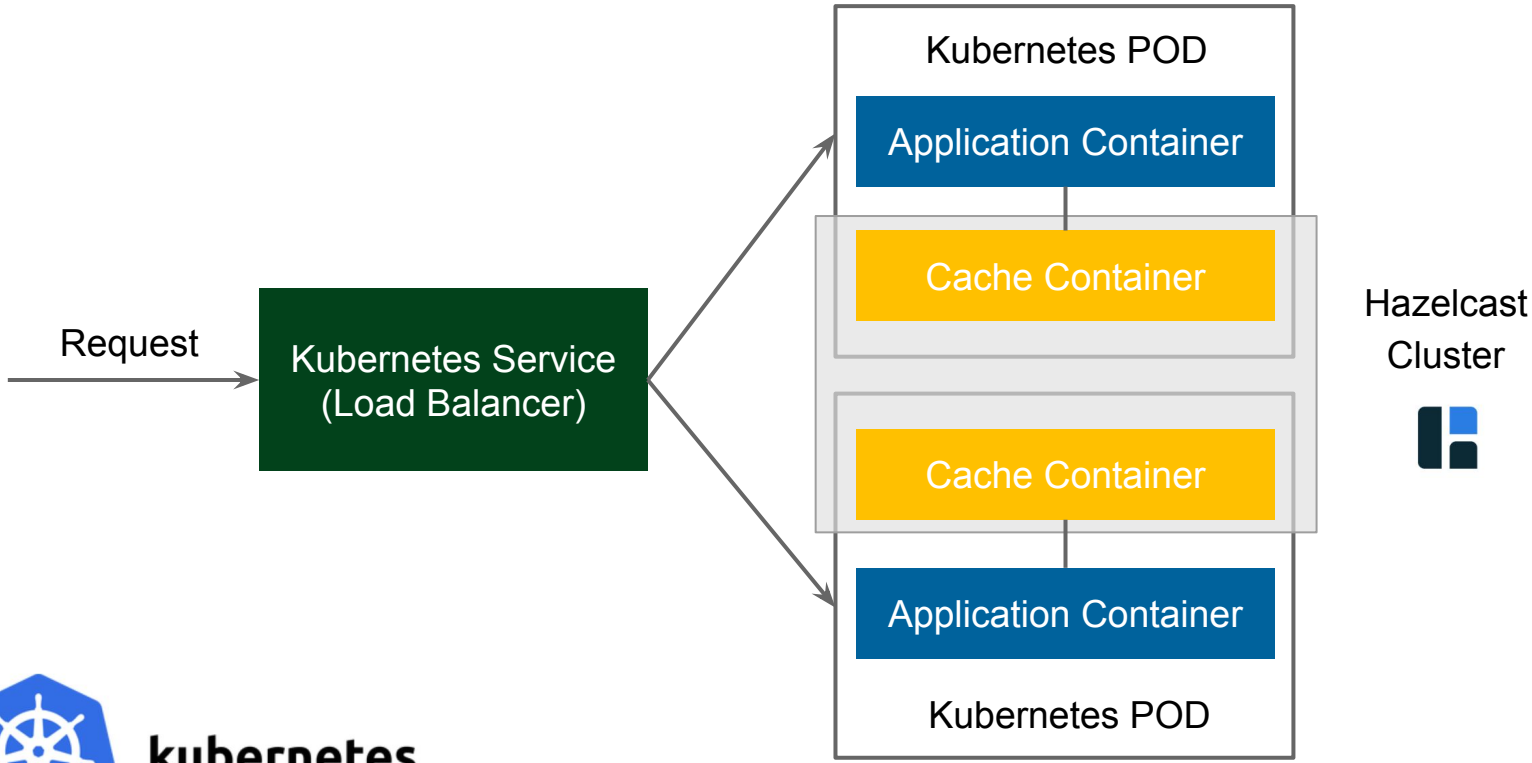
Agenda

- Introduction ✅
- Caching Architectural Patterns
  - Embedded ✅
  - Embedded Distributed ✅
  - Client-Server ✅
  - Cloud ✅
  - Sidecar
  - Reverse Proxy
  - Reverse Proxy Sidecar
- Summary


Arai
Arai
VOLT
HILLAR KOOSAPOEG
Nebori
uruvana
ProCoating
RAIMO PRII
Rokiti
AAVE TRANSPORT
31

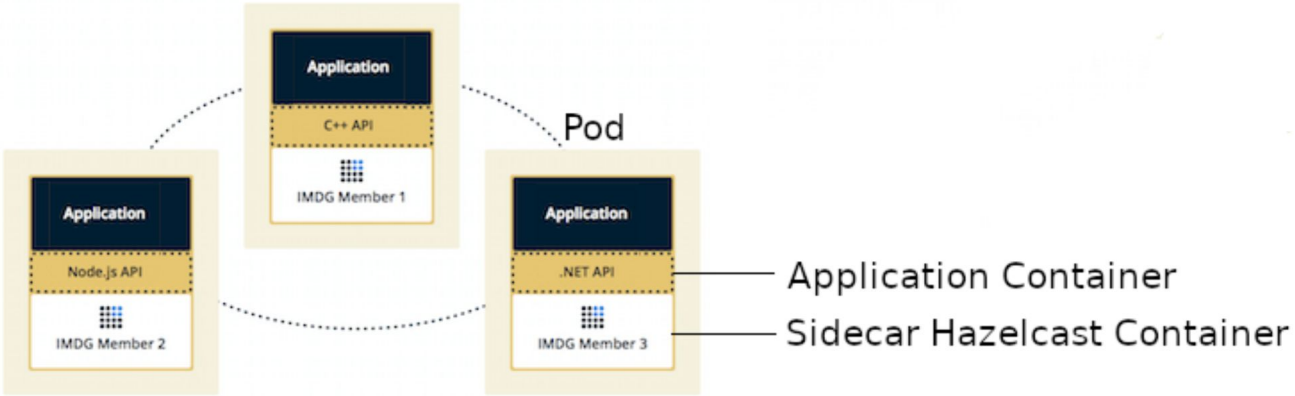
3. Sidecar

Sidecar Cache



kubernetes

Sidecar Cache



Similar to **Embedded**:

- the same physical machine
- the same resource pool
- scales up and down together
- no discovery needed (always localhost)

Similar to **Client-Server**:

- different programming language
- uses cache client to connect
- clear isolation between app and cache

Sidecar Cache

```
@Configuration
public class HazelcastSidecarConfiguration {
    @Bean
    CacheManager cacheManager() {
        ClientConfig clientConfig = new ClientConfig();
        clientConfig.getNetworkConfig()
                            .addAddress("localhost:5701");
        return new HazelcastCacheManager(HazelcastClient
                            .newHazelcastClient(clientConfig));
    }
}
```

Sidecar Cache

```
apiVersion: apps/v1
kind: Deployment
...
spec:
 template:
   spec:
     containers:
       - name: application
         image: leszko/application
       - name: hazelcast
         image: hazelcast/hazelcast
```

Sidecar Cache

Pros

- Simple configuration
- Programming-language agnostic
- Low latency
- Some isolation of data and applications

Cons

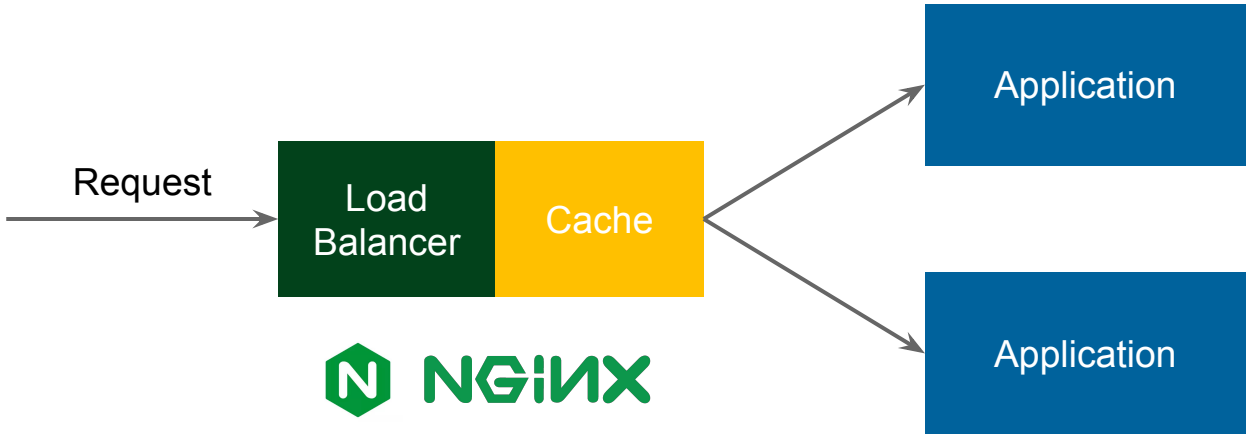
- Limited to container-based environments
- Not flexible management (scaling, backup)
- Data collocated with application PODs

Agenda

- Introduction ✅
- Caching Architectural Patterns
  - Embedded ✅
  - Embedded Distributed ✅
  - Client-Server ✅
  - Cloud ✅
  - Sidecar ✅
  - Reverse Proxy
  - Reverse Proxy Sidecar
- Summary

4. Reverse Proxy

Reverse Proxy Cache



Reverse Proxy Cache


```
http {
    ...
    proxy_cache_path /data/nginx/cache
                                keys_zone=one:10m;
    ...
}
```

NGINX Reverse Proxy Cache Issues

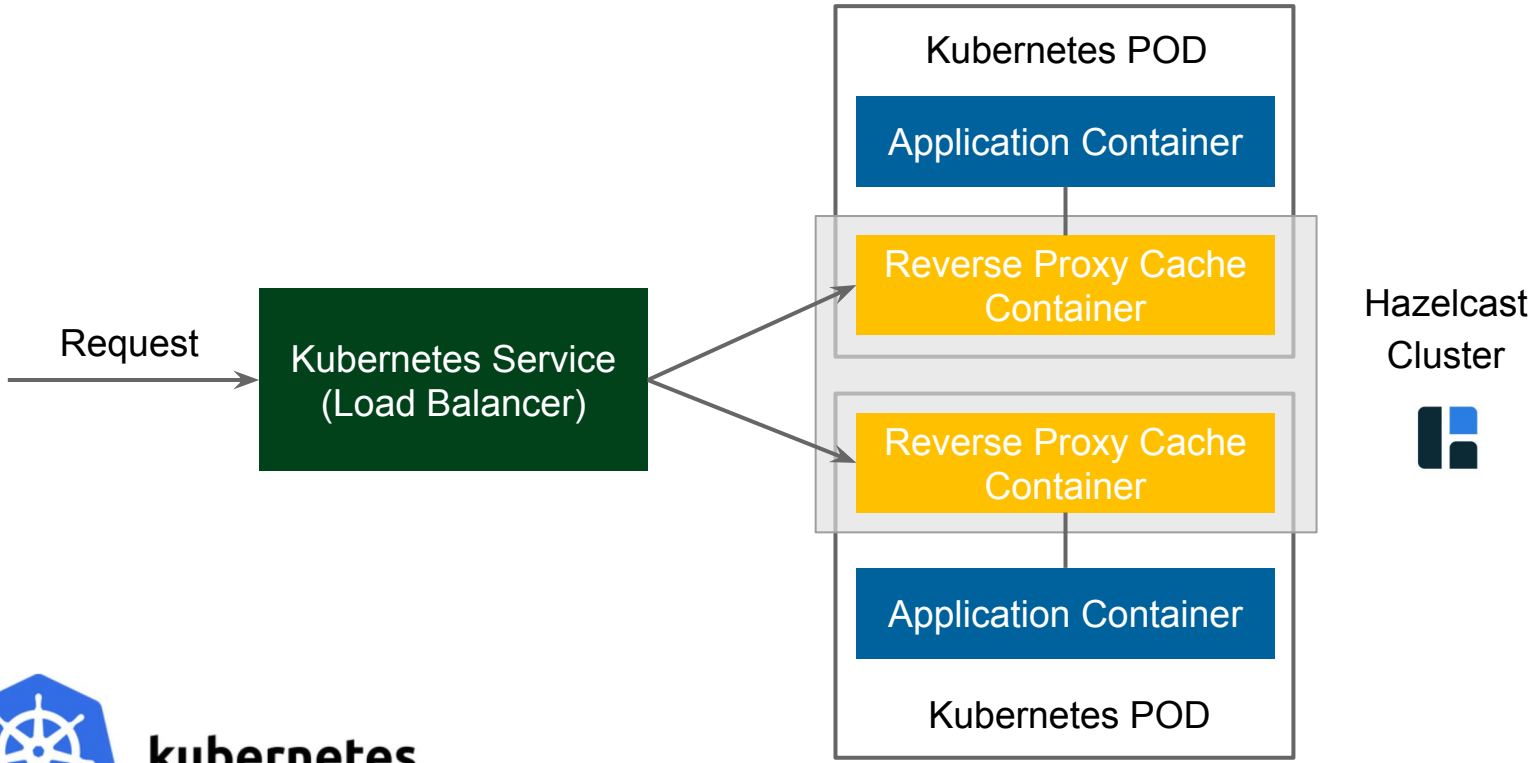
- Only for HTTP
- Not distributed
- No High Availability
- Data stored on the disk

NGINX Reverse Proxy Cache Issues

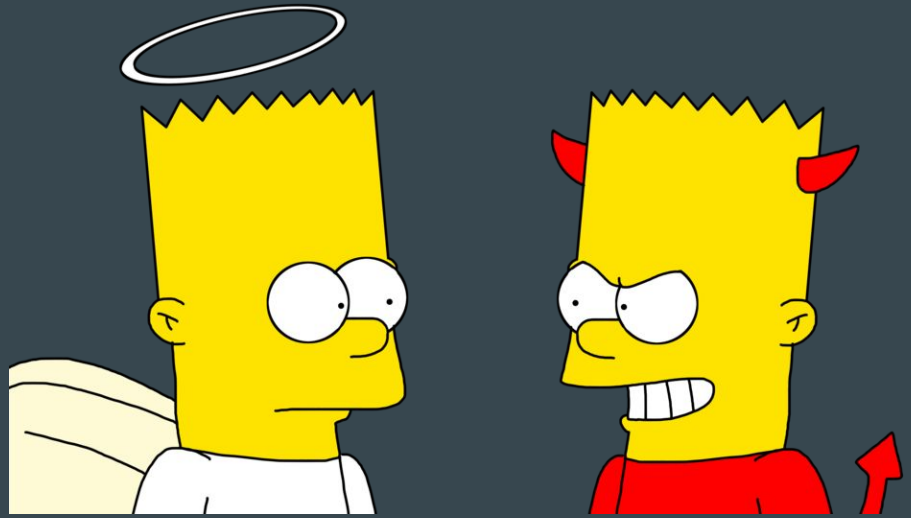
- Only for HTTP
- Not distributed
- No High Availability
- Data stored on the disk

4*. Reverse Proxy Sidecar

Reverse Proxy Sidecar Cache

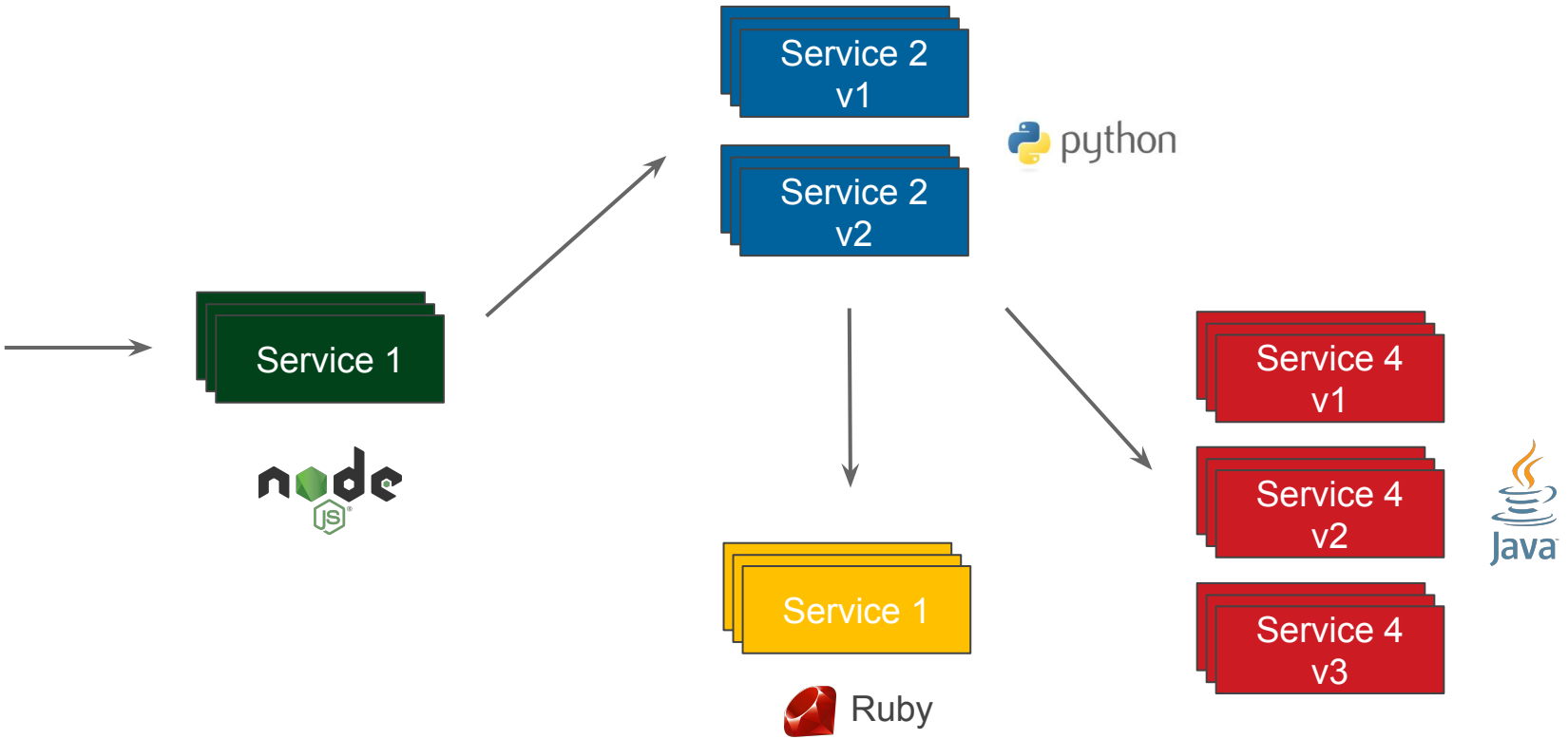


kubernetes

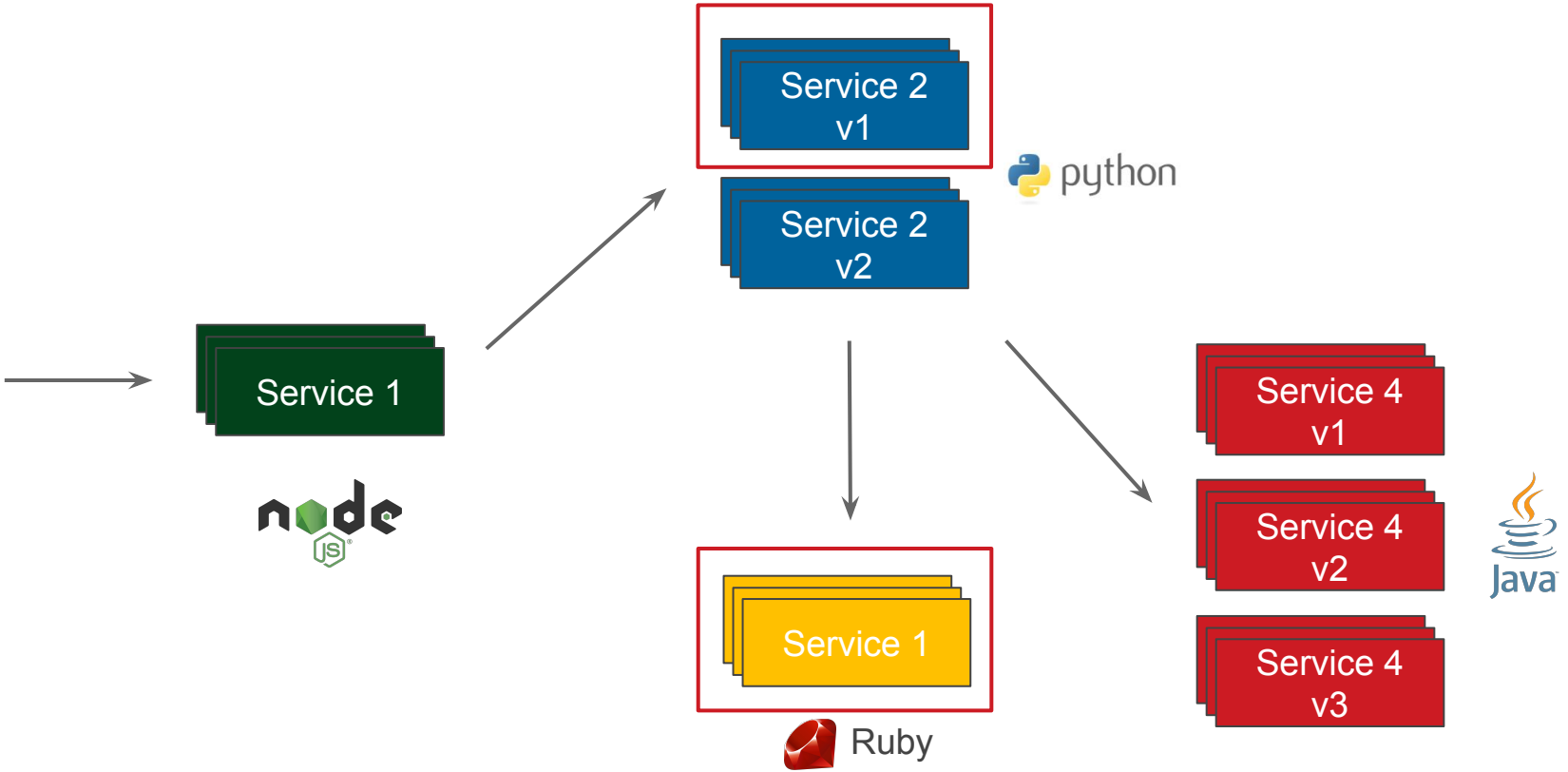


Good

Reverse Proxy Sidecar Cache



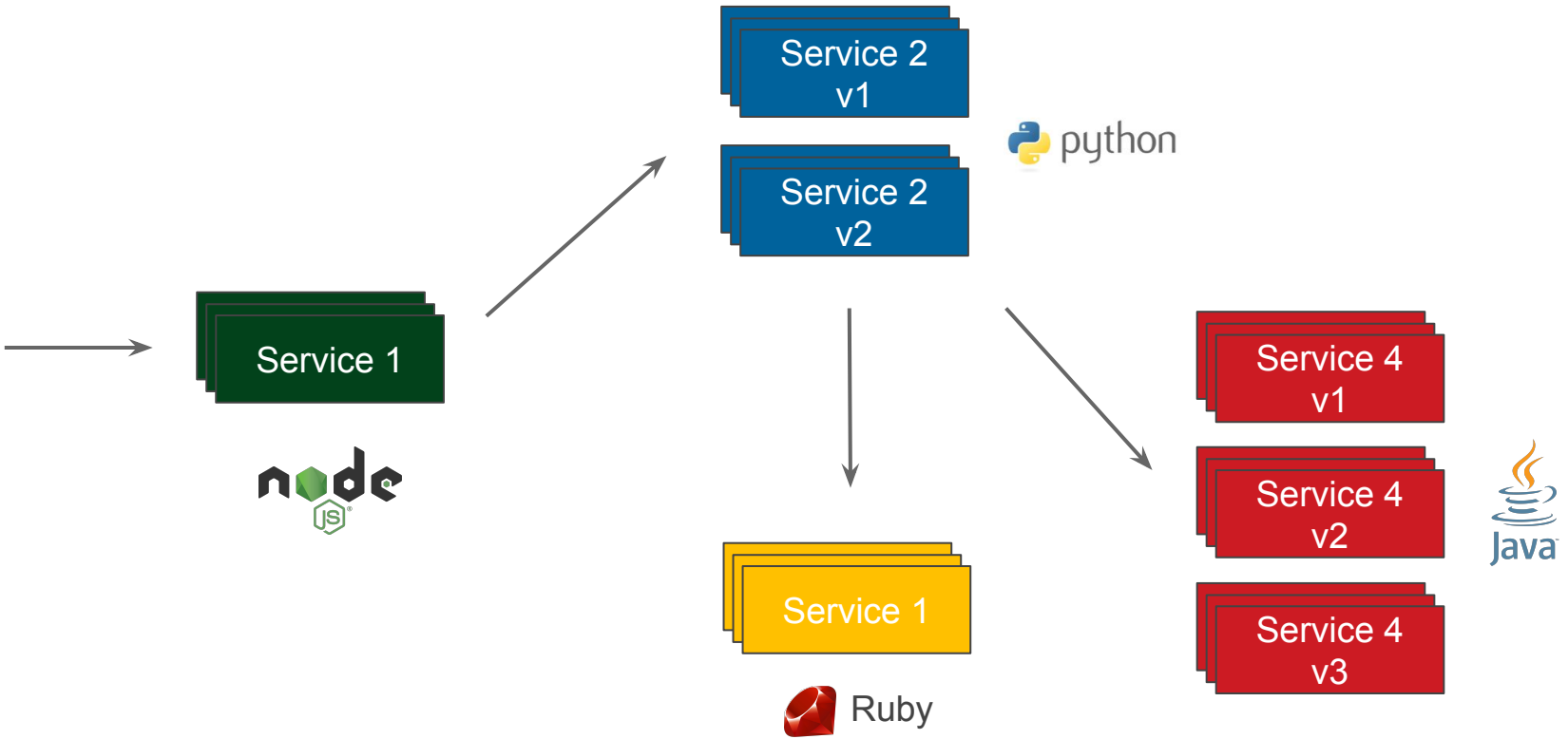
Reverse Proxy Sidecar Cache

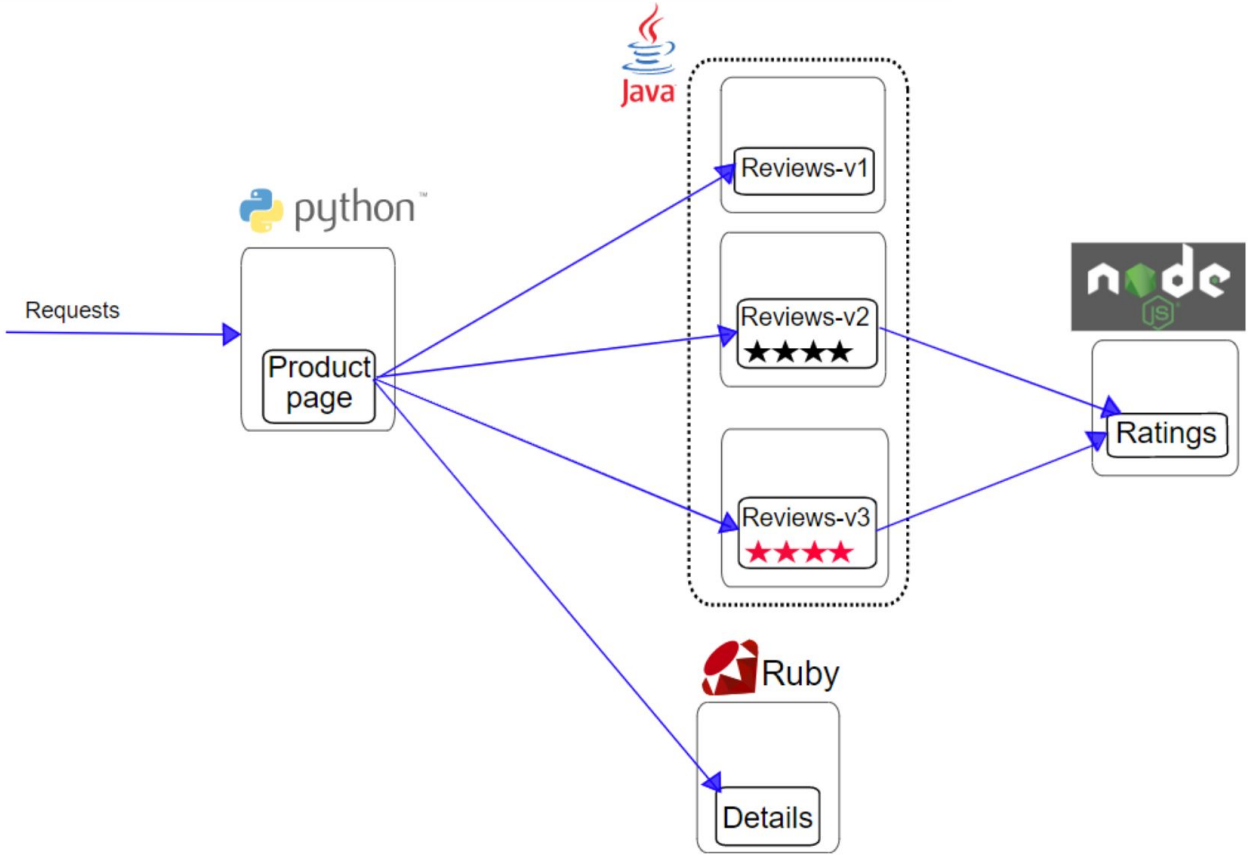


Reverse Proxy Sidecar Cache

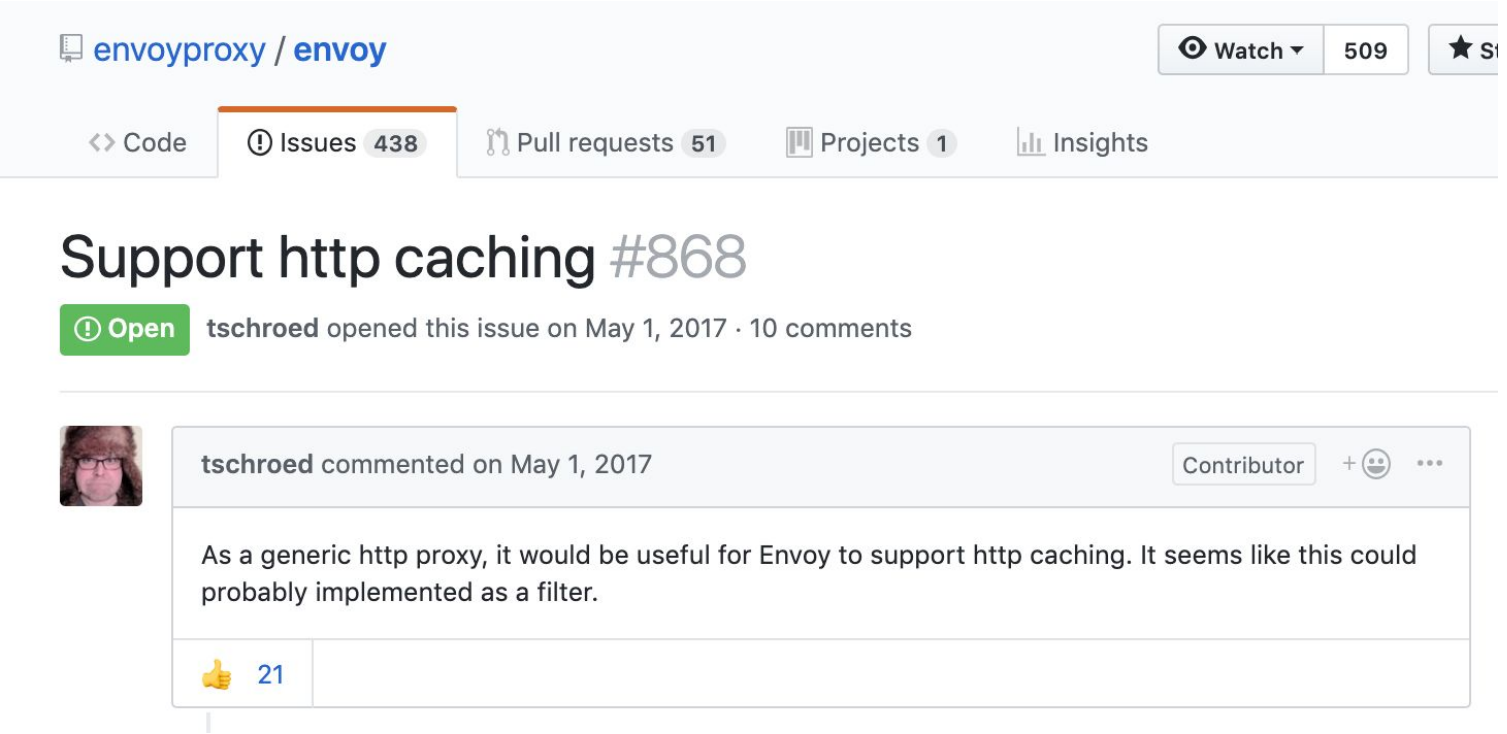
```
apiVersion: apps/v1
kind: Deployment
...
spec:
 template:
   spec:
     initContainers:
       - name: init-networking
         image: leszko/init-networking
     containers:
       - name: caching-proxy
         image: leszko/caching-proxy
       - name: application
         image: leszko/application
```

Reverse Proxy Sidecar Cache




Java
python
Requests
Product page
Reviews-v1
Reviews-v2
★★★★
Reviews-v3
★★★★
Ratings
Ruby
Details

Reverse Proxy Sidecar Cache (Istio)



Bad

Quora Home 1 Answer Spaces Notifications Search Quora

What are some best practices for caching in a typical web app?

This question previously had details. They are now in a comment.

Answer Follow · 48 Request 1

4 Answers

Kellan Elliott-McCrea
Answered Sep 4, 2010

The hardest part of caching is cache invalidation. If you're a content driven site, then your job is trivial. If you are building anything resembling social software caching involves a series of complex trade offs.

Reverse Proxy Cache

Application Cache:

```
@CacheEvict(value = "someValue", allEntries = true)
public void evictAllCacheValues() {}
```

Reverse Proxy Cache

Application Cache:

```
@CacheEvict(value = "someValue", allEntries = true)
public void evictAllCacheValues() {}
```

Proxy Cache:

```
http {
    ...
    location / {
        add_header Cache-Control public;
        expires 86400;
        etag on;
    }
}
```

Reverse Proxy (Sidecar) Cache

Pros

- Configuration-based (no need to change applications)
- Programming-language agnostic
- Consistent with containers and microservice world

Cons

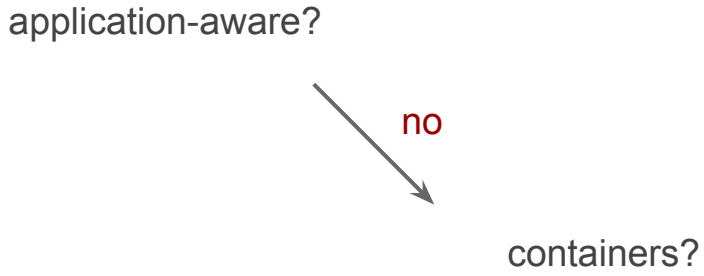
- Difficult cache invalidation
- No mature solutions yet
- Protocol-based (e.g. works only with HTTP)

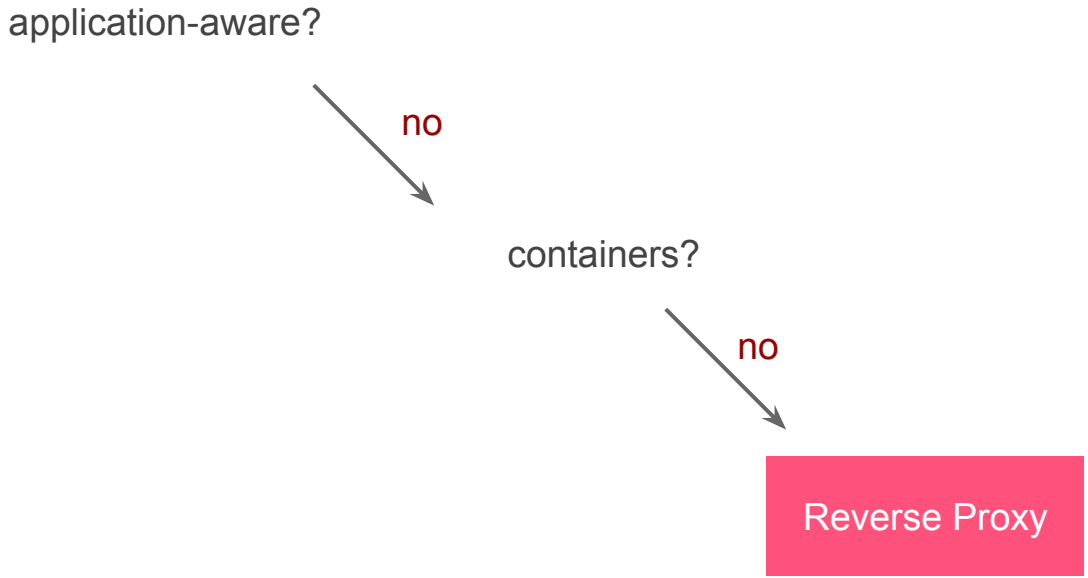
Agenda

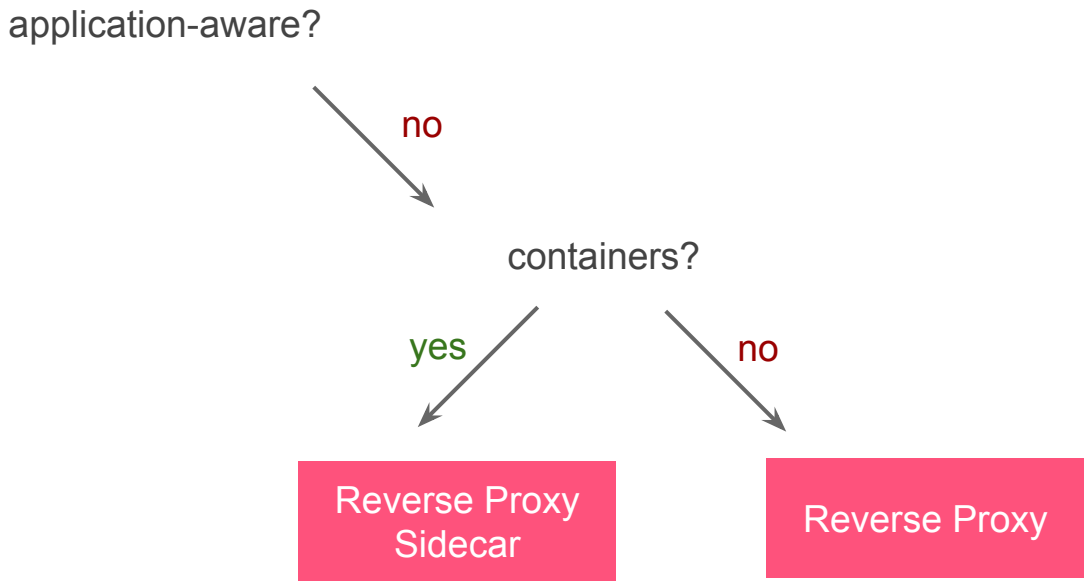
- Introduction ✅
- Caching Architectural Patterns
  - Embedded ✅
  - Embedded Distributed ✅
  - Client-Server ✅
  - Cloud ✅
  - Sidecar ✅
  - Reverse Proxy ✅
  - Reverse Proxy Sidecar ✅
- Summary

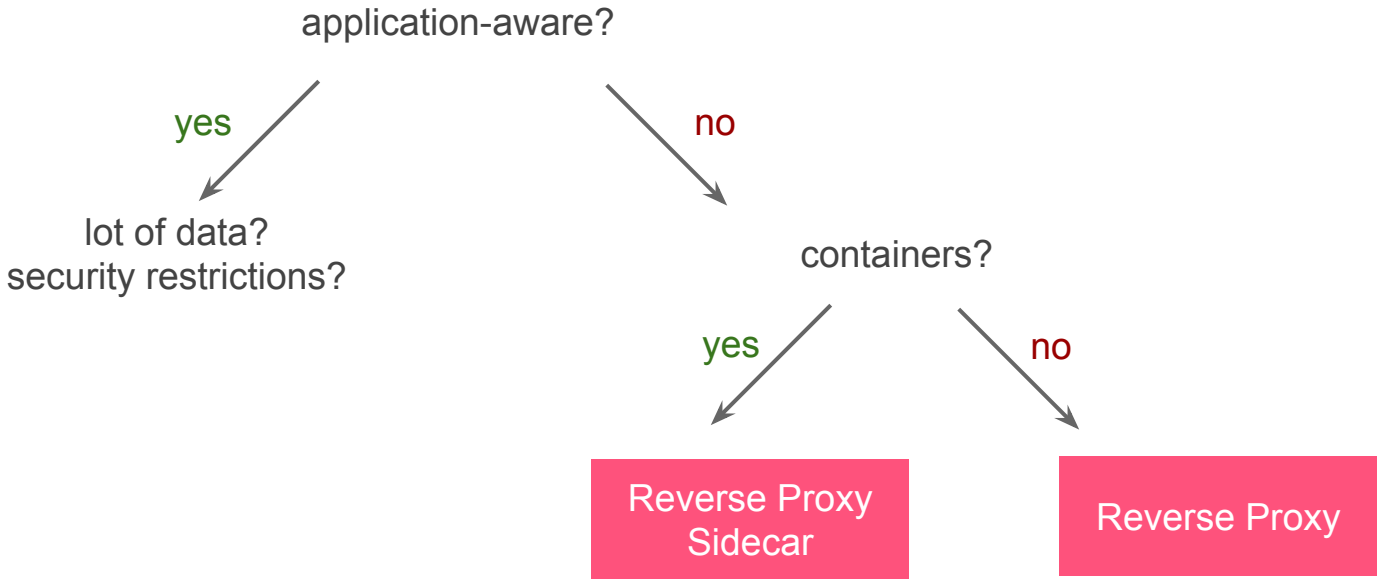
Summary

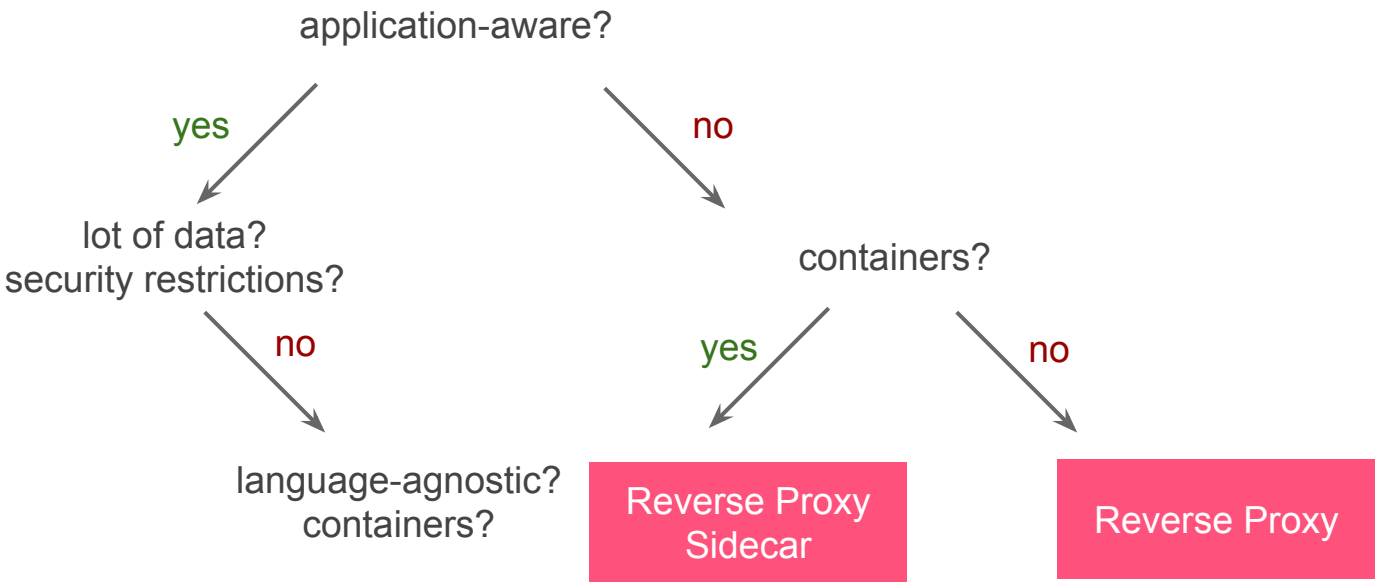
application-aware?

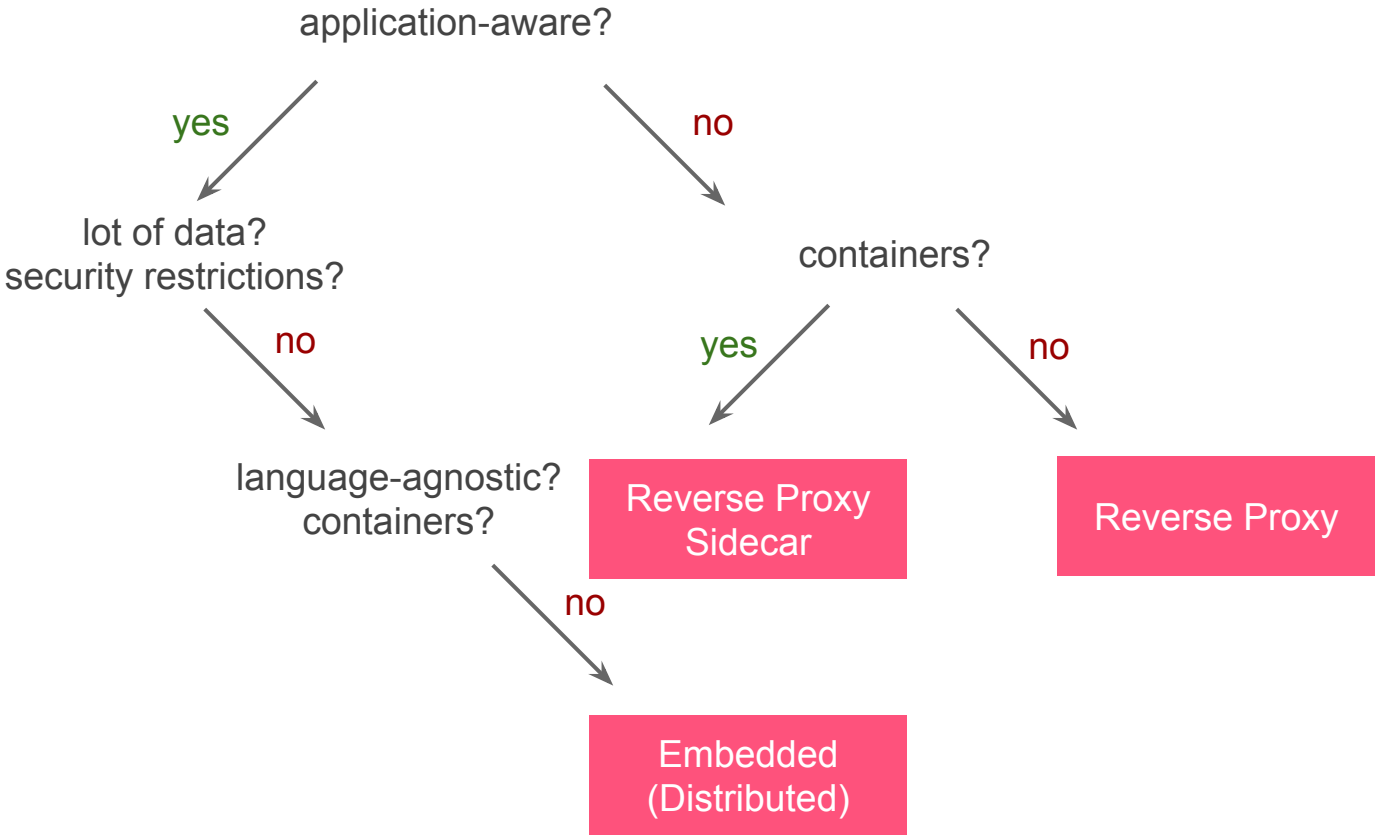

application-aware?
no
containers?

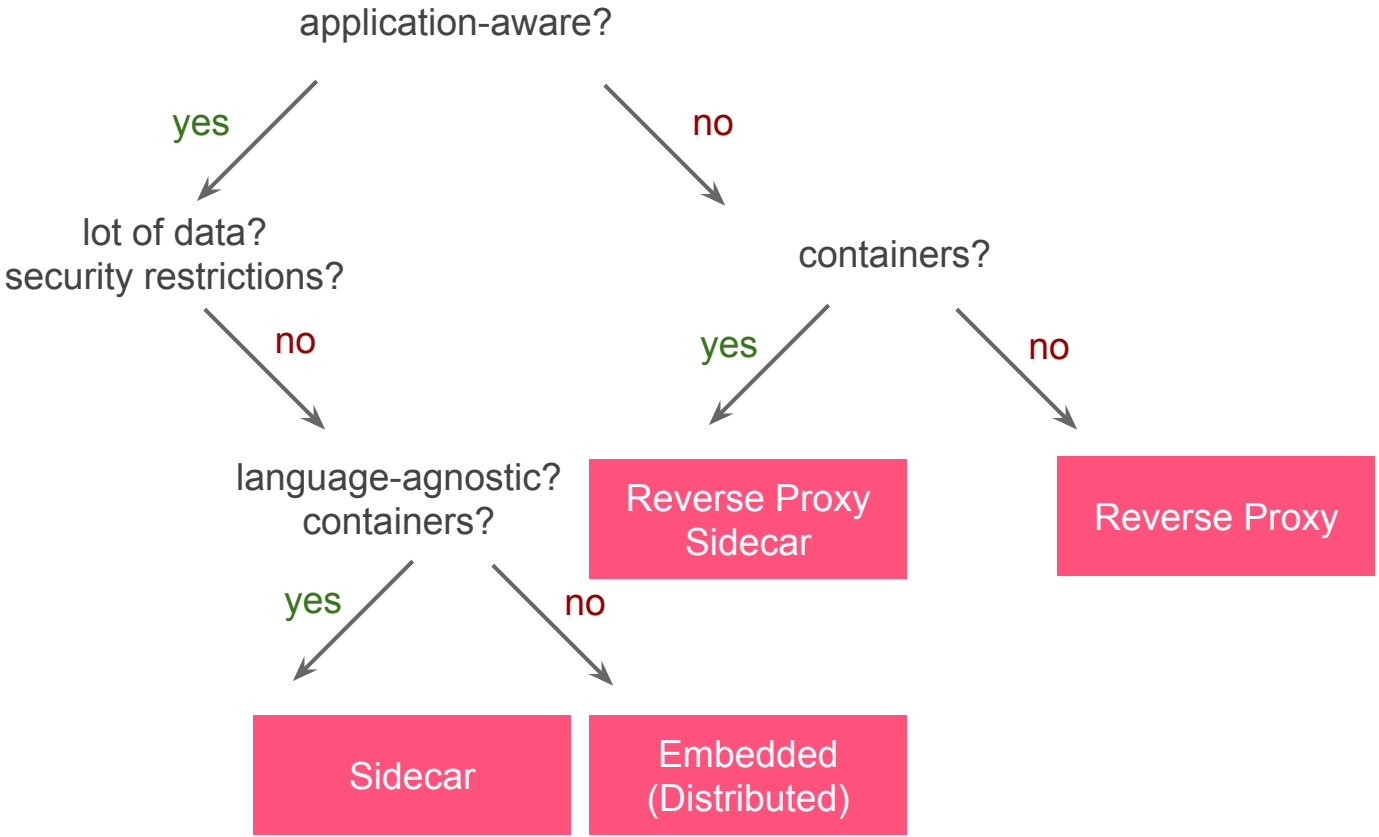

application-aware?
no
containers?
no
Reverse Proxy

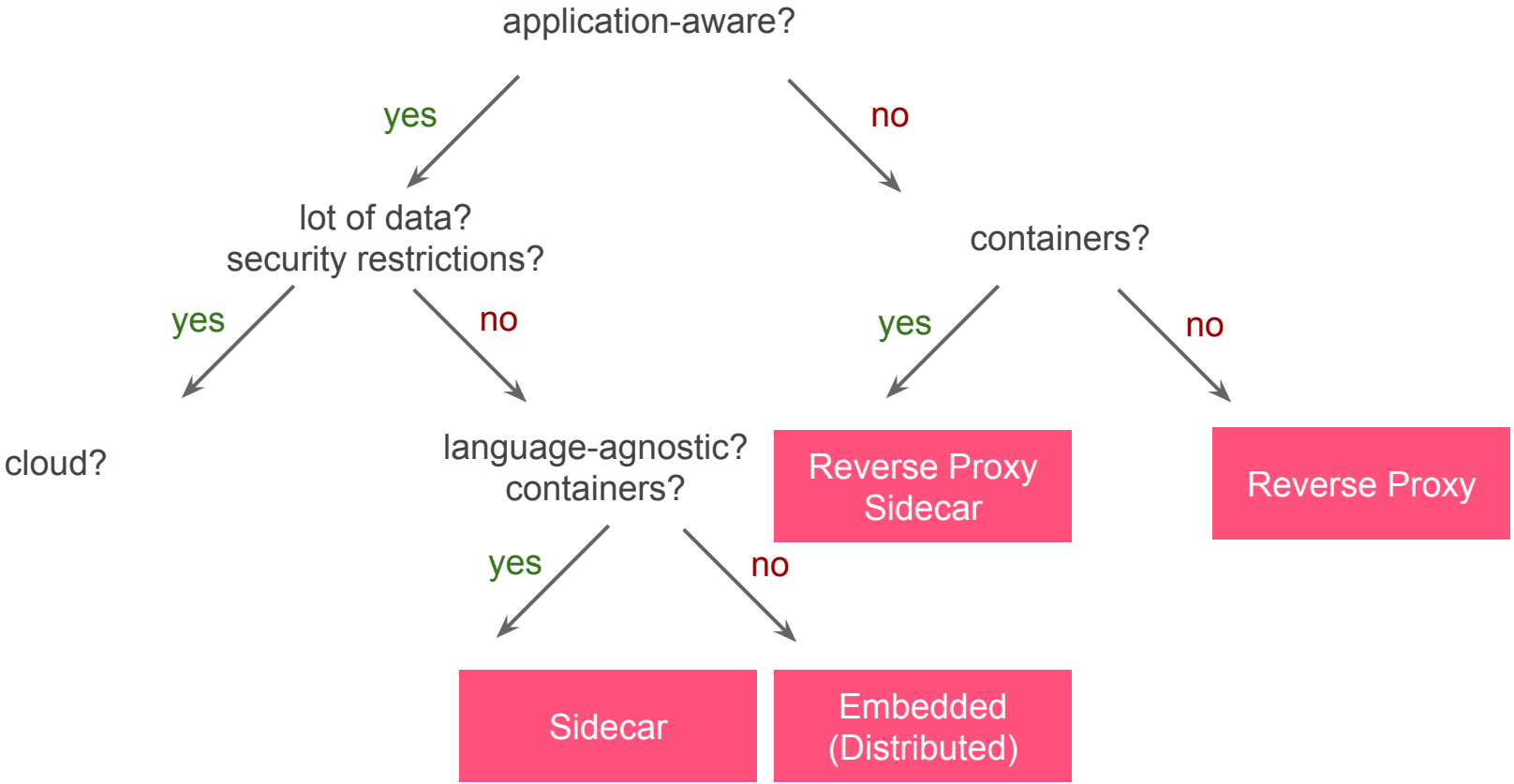

application-aware?
no
containers?
yes
no
Reverse Proxy
Sidecar
Reverse Proxy

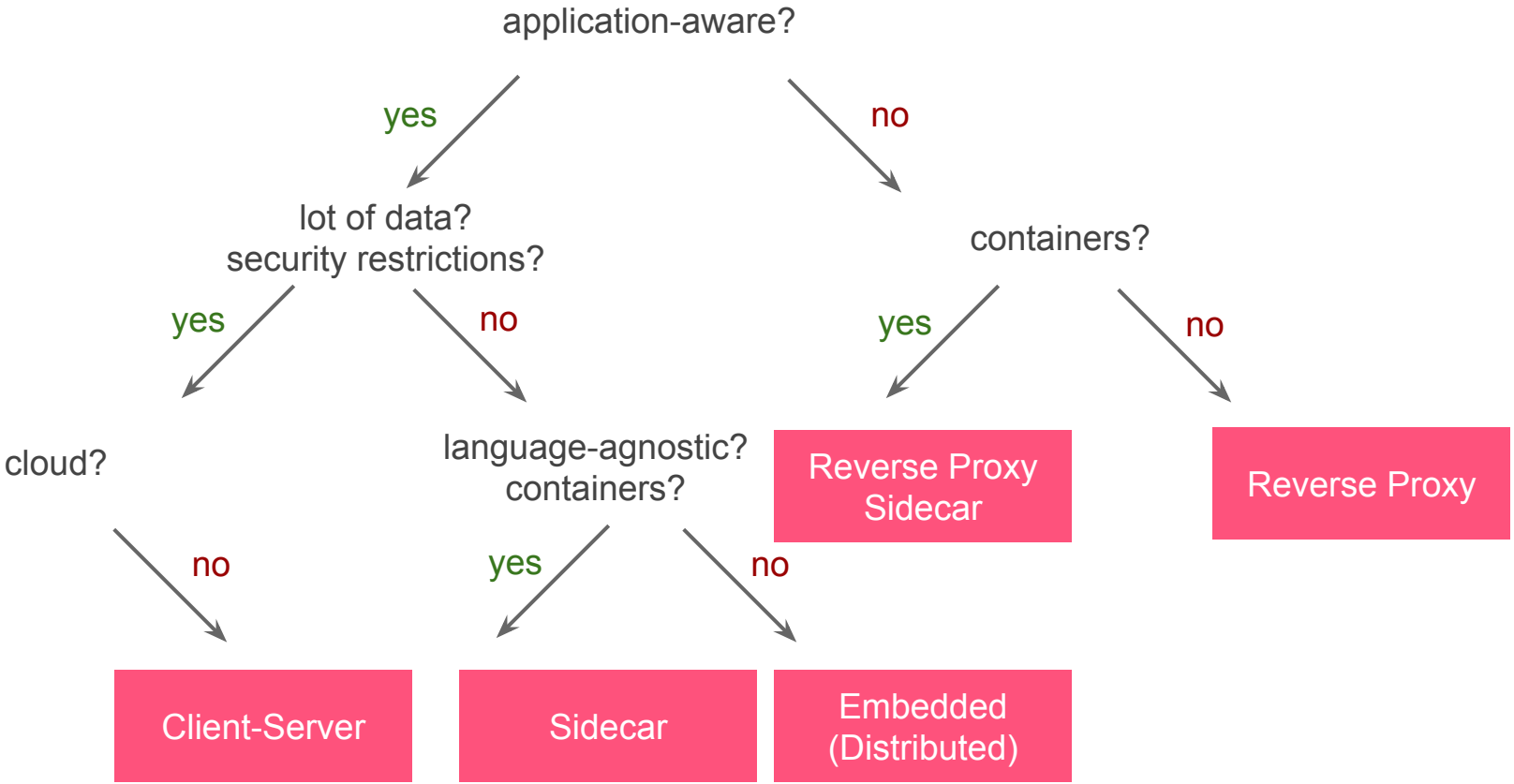

application-aware?
yes
no
lot of data?
security restrictions?
containers?
yes
no
Reverse Proxy
Sidecar
Reverse Proxy

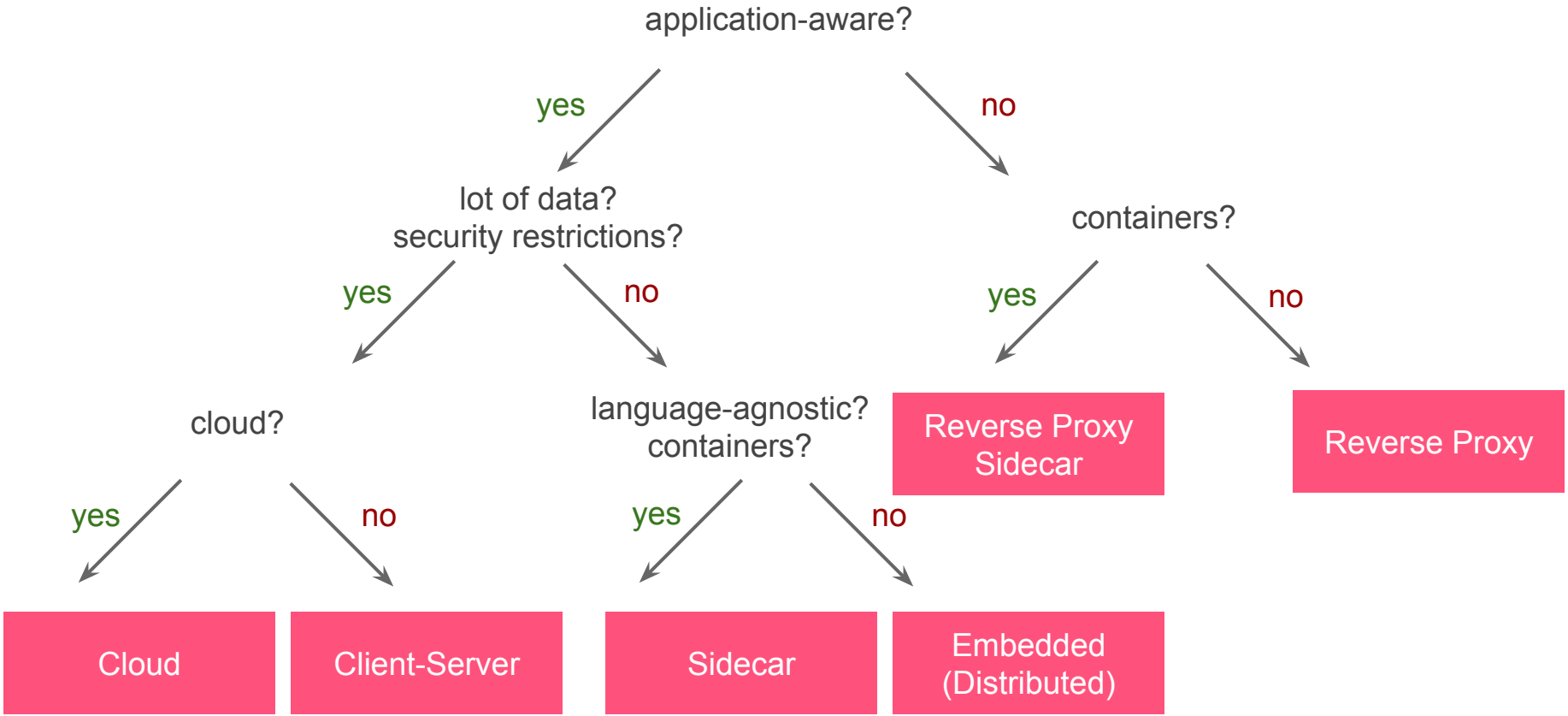

application-aware?
yes
no
lot of data?
security restrictions?
containers?
no
yes
no
language-agnostic?
containers?
Reverse Proxy
Sidecar
Reverse Proxy


application-aware?
yes
no
lot of data?
security restrictions?
containers?
no
yes
no
language-agnostic?
containers?
Reverse Proxy
Sidecar
Reverse Proxy
no
Embedded
(Distributed)


application-aware?
yes
no
lot of data?
security restrictions?
containers?
no
yes
no
language-agnostic?
containers?
Reverse Proxy
Sidecar
Reverse Proxy
yes
no
Sidecar
Embedded
(Distributed)


application-aware?
yes
no
lot of data?
security restrictions?
containers?
yes
no
yes
no
cloud?
language-agnostic?
containers?
Reverse Proxy
Sidecar
Reverse Proxy
yes
no
Sidecar
Embedded
(Distributed)


application-aware?
yes
no
lot of data?
security restrictions?
containers?
yes
no
yes
no
cloud?
language-agnostic?
containers?
Reverse Proxy
Sidecar
Reverse Proxy
no
yes
no
Client-Server
Sidecar
Embedded
(Distributed)


application-aware?
yes
no
lot of data?
security restrictions?
containers?
yes
no
yes
no
cloud?
language-agnostic?
containers?
Reverse Proxy
Sidecar
Reverse Proxy
yes
no
yes
no
Cloud
Client-Server
Sidecar
Embedded
(Distributed)

Resources

- Hazelcast Sidecar Container Pattern: https://hazelcast.com/blog/hazelcast-sidecar-container-pattern/
- Hazelcast Reverse Proxy Sidecar Caching Prototype: https://github.com/leszko/caching-injector
- Caching Best Practices: https://vladmihalcea.com/caching-best-practices/
- NGINX HTTP Reverse Proxy Caching: https://www.nginx.com/resources/videos/best-practices-for-caching/

Thank You!

Rafał Leszko

@RafalLeszko