

Where is my cache?

Architectural patterns for caching microservices by example



Rafał Leszko

 @RafalLeszko

rafalleszko.com

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
- Products:
 - Hazelcast IMDG
 - Hazelcast Jet
 - Hazelcast Cloud



hazelcast



@Hazelcast

www.hazelcast.com

Agenda

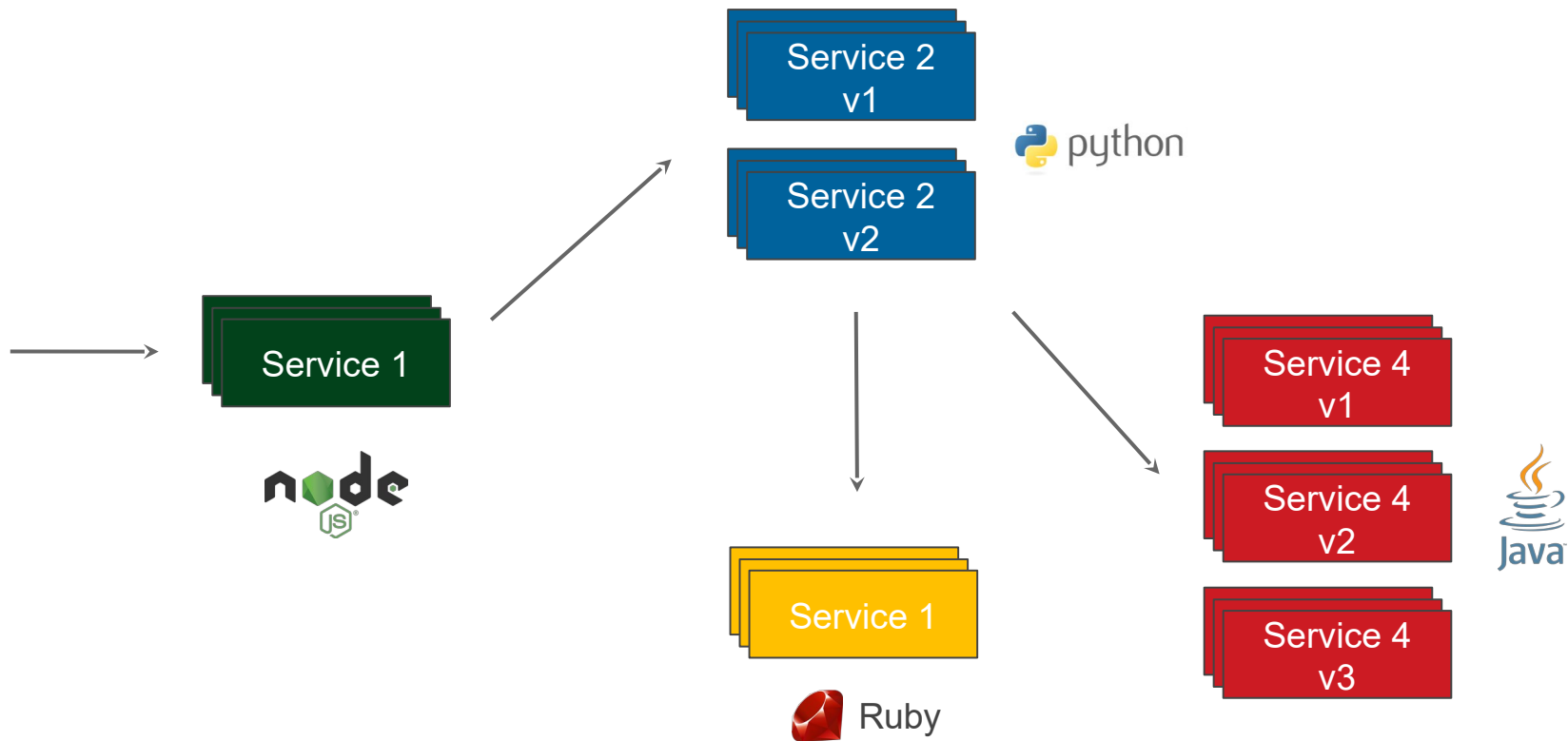
- Introduction
- Caching Architectural Patterns
 - Embedded
 - Embedded Distributed
 - Client-Server
 - Cloud
 - Sidecar
 - Reverse Proxy
 - Reverse Proxy Sidecar
- Summary

Why Caching?

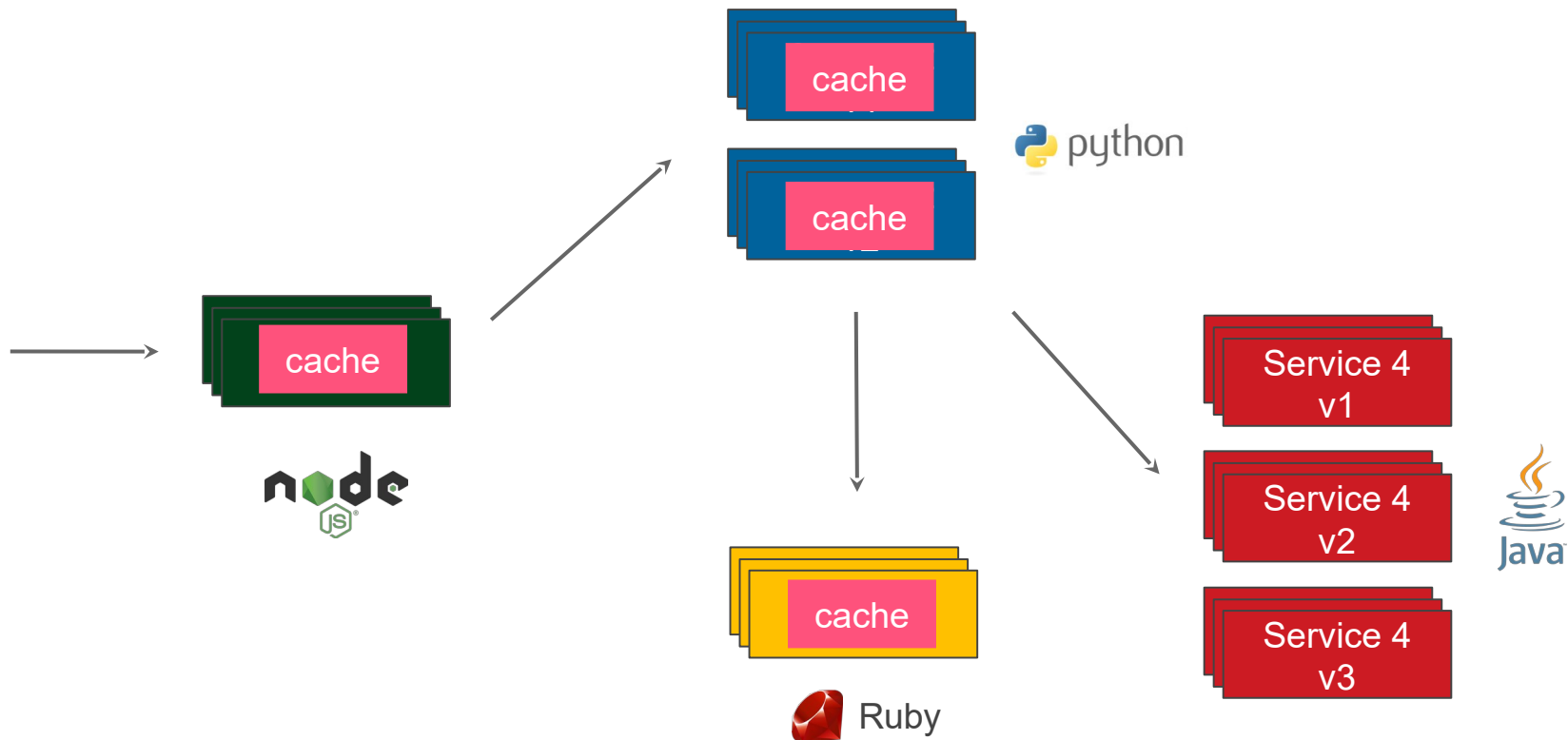
- Performance
 - Decrease latency
 - Reduce load
- Resilience
 - High availability
 - Lower downtime



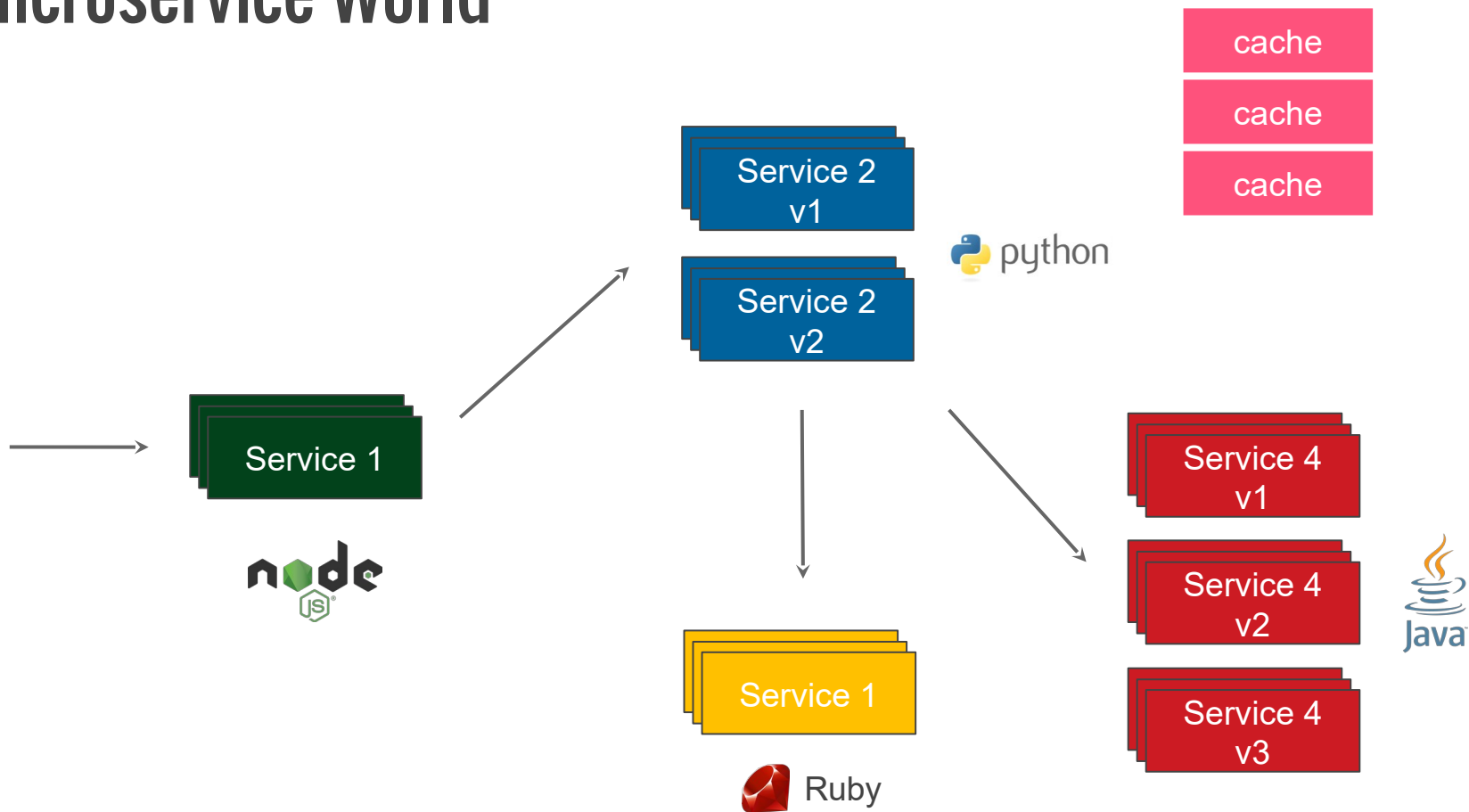
Microservice World



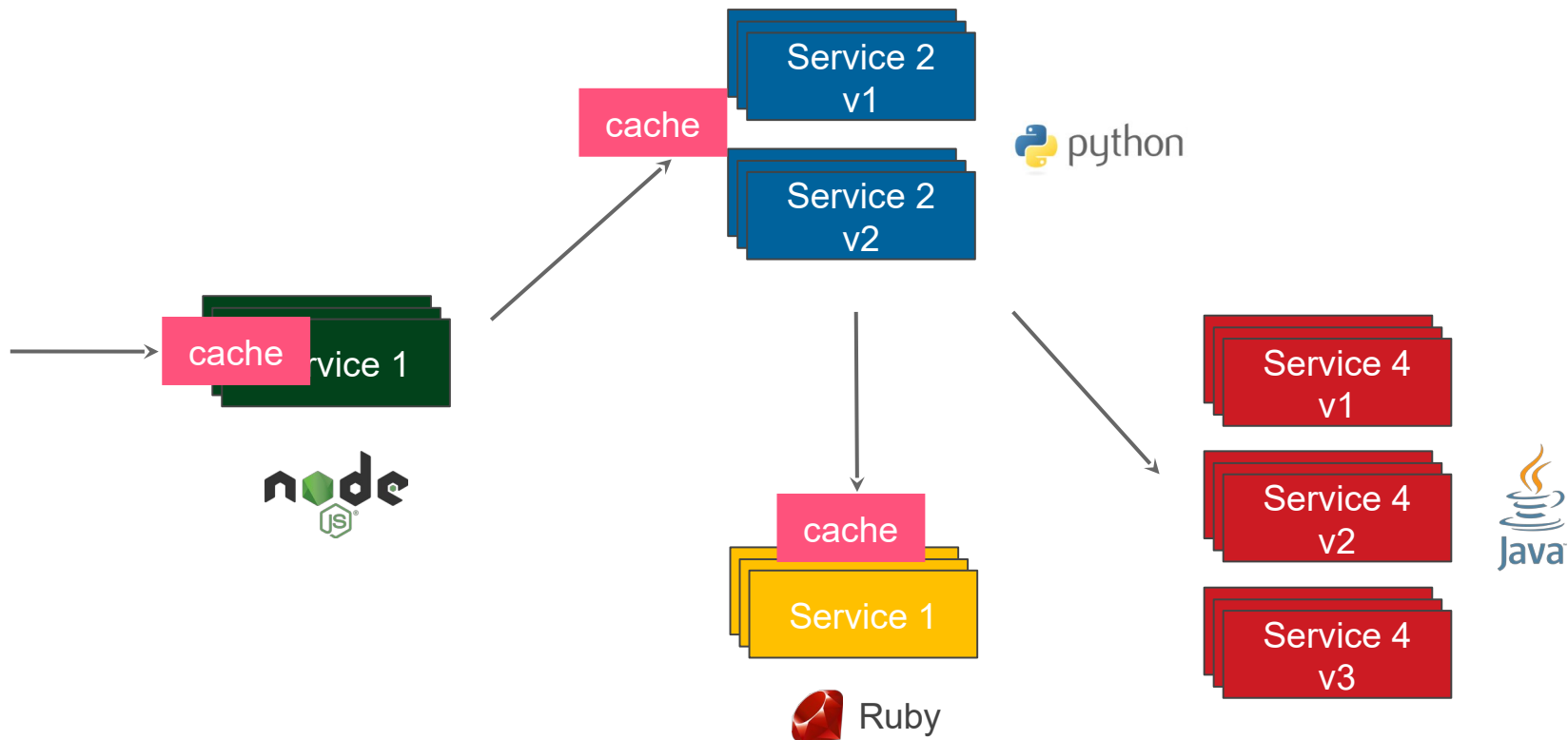
Microservice World



Microservice World



Microservice World



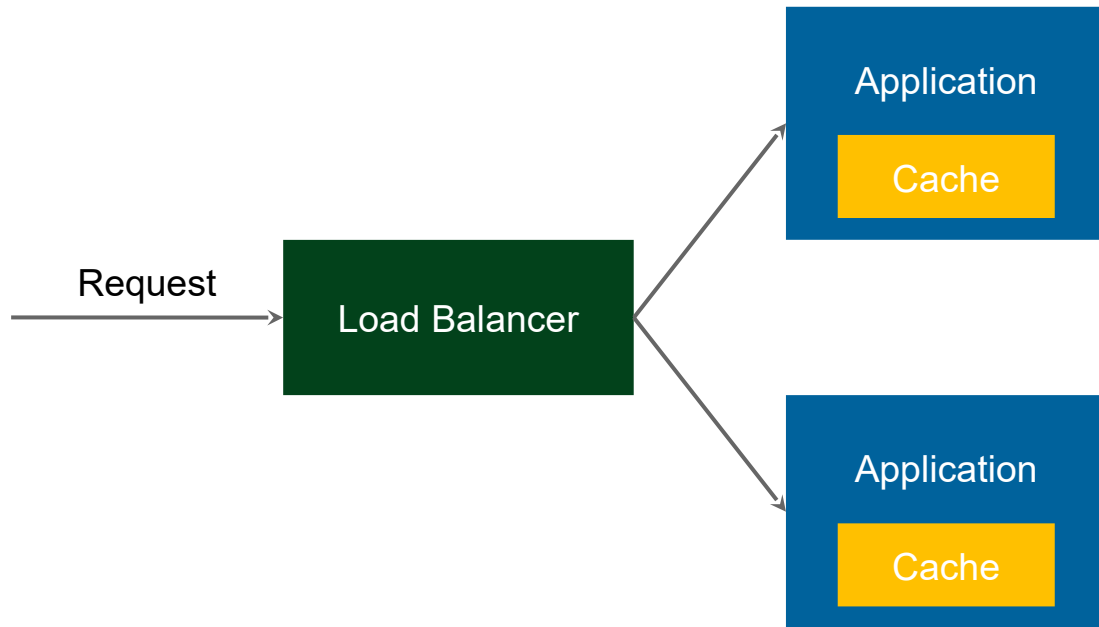
Agenda

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

A close-up photograph of an ancient stone Buddha head, likely from Southeast Asia, partially encased and surrounded by a dense network of thick, gnarled tree roots. The roots are light brown and weathered, with some showing signs of decay or insect damage. The Buddha's face is weathered, with a serene expression, and the top of the head is covered in a pattern of small, rounded, pebbled textures. A semi-transparent white horizontal bar is overlaid across the middle of the image, containing the text "1. Embedded".

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 ConcurrentMap<String, String> cache =  
    new ConcurrentHashMap<>();  
  
private String processRequest(String request) {  
    if (cache.containsKey(request)) {  
        return cache.get(request);  
    }  
    String response = process(request);  
    cache.put(request, response);  
    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

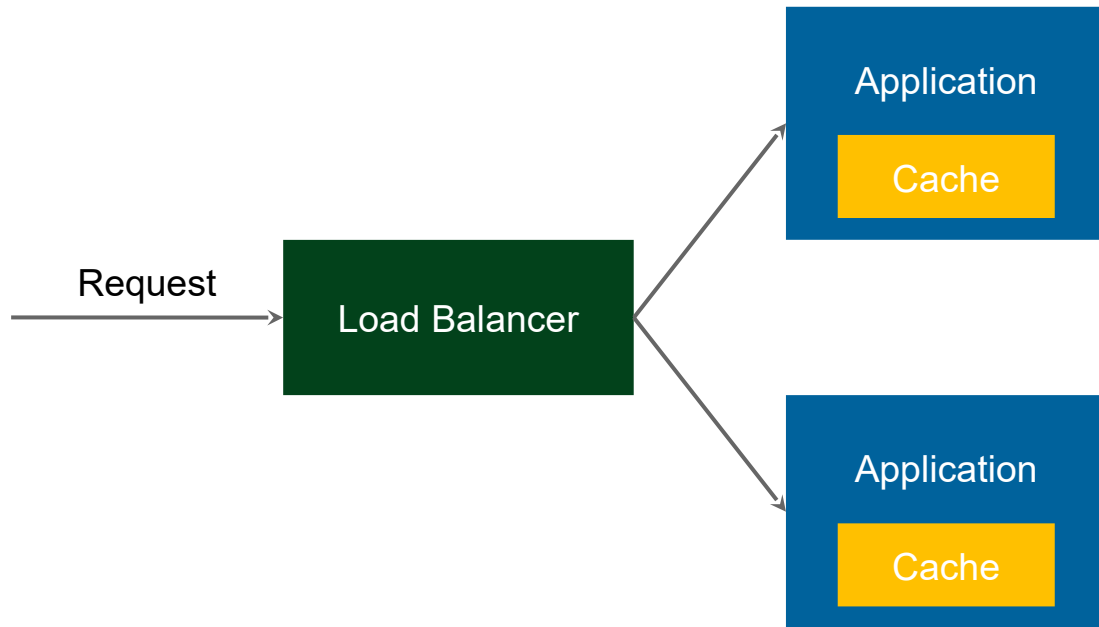


```
@Service
public class BookService {

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

Be Careful, Spring uses ConcurrentHashMap by default!

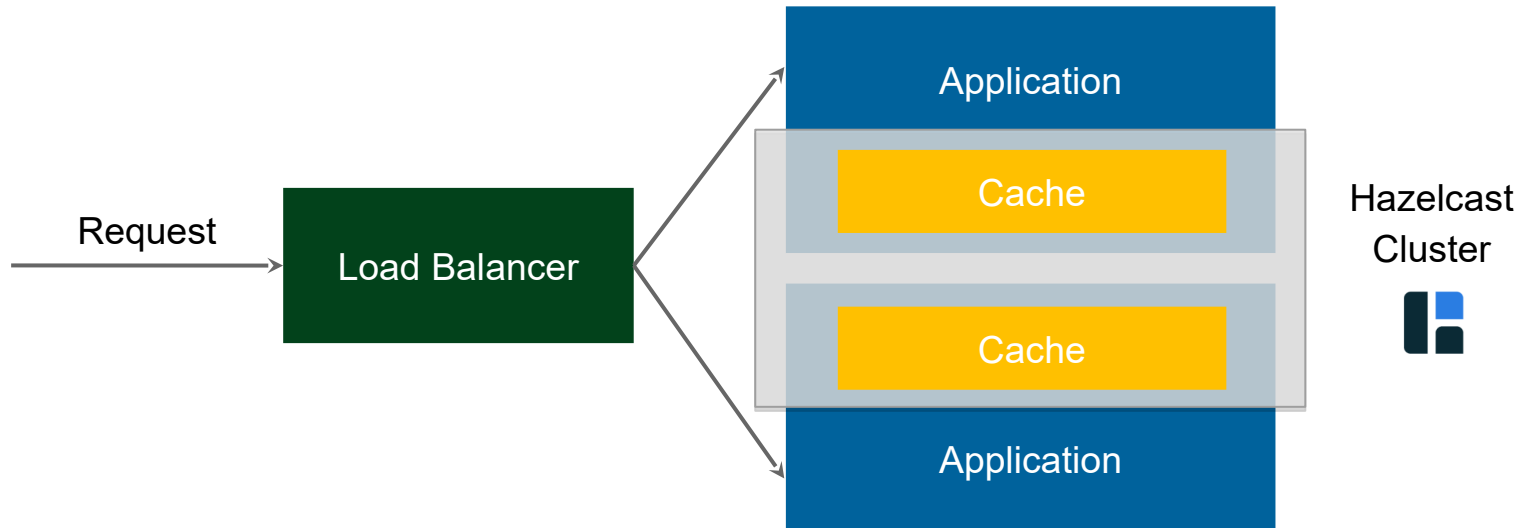
Embedded Cache



A photograph of ancient stone ruins, likely Mayan or Aztec, featuring several tall, tiered stone structures. A large, thick tree trunk stands prominently in the center, with its roots spreading out over the ground and some of the ruins. The scene is surrounded by lush green foliage.

1*. Embedded Distributed

Embedded Distributed Cache



Embedded Distributed Cache (Spring with Hazelcast)

```
@Configuration
public class HazelcastConfiguration {

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

DEMO



Hazelcast Discovery Plugins



Google Cloud Platform



kubernetes



Hazelcast Discovery Plugins

https://hazelcast.com/blog/how-to-use-embedded-hazelcast-on-kubernetes/



Open Source Projects: [IMDG](#) | [Jet](#) | [Training](#)



Products and Services

Use Cases


Resources



Get Hazelcast

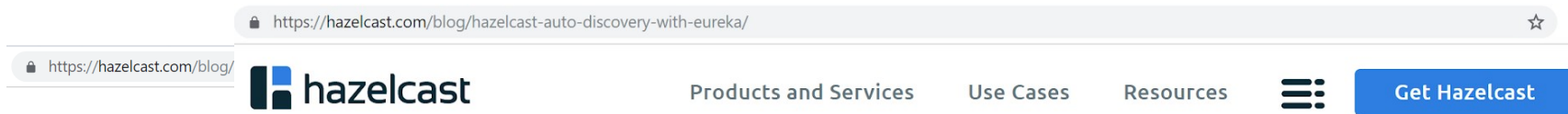
How to Use Embedded Hazelcast on Kubernetes

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



How to Use Hazelcast Auto-Discovery with Eureka

Rafal Leszko | April 24, 2019

How to Use

Rafal Leszko | February 06, 2019

Share

Share

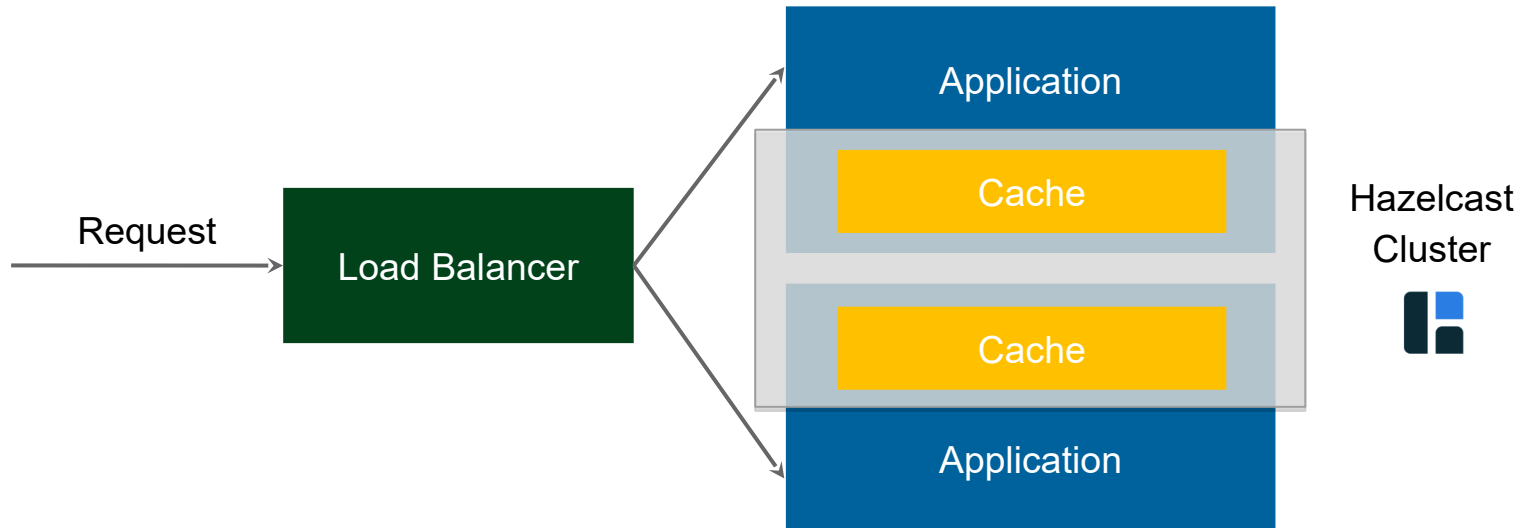
Hazelcast IMDG is a peer-to-peer data store that can therefore scale in and out. You can integrate Hazelcast into a Spring application [here](#).

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.

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

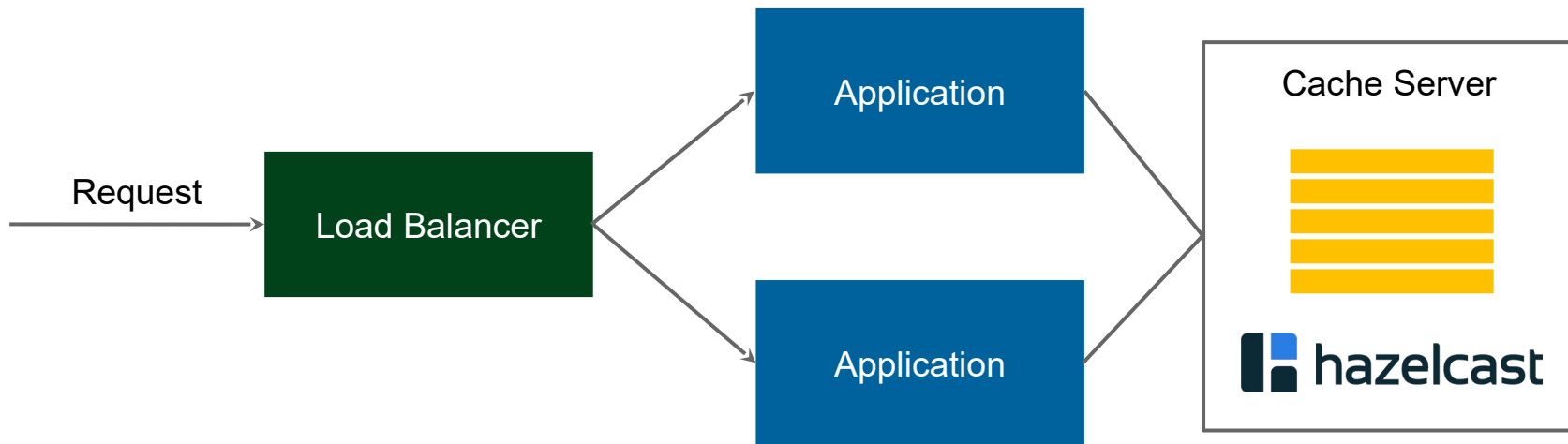
Agenda

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

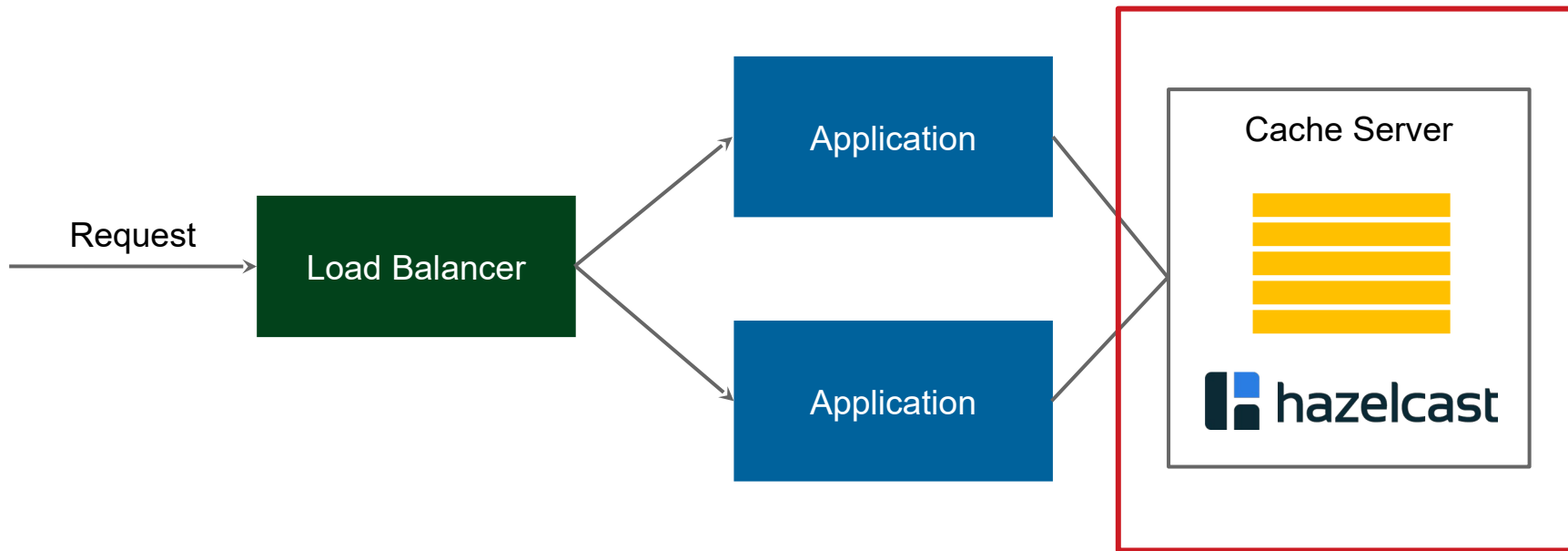
A group of five tourists, including a man in a white shirt and black cap, and four women in colorful traditional Burmese sarongs, are standing on a grassy lawn. One woman is pointing towards a large, dark, ornate building in the background. To the right, a Buddhist monk in maroon robes is smiling and taking a photo with a camera. The scene is set in a park-like area with large green trees and a clear sky.

2. Client-Server

Client-Server Cache



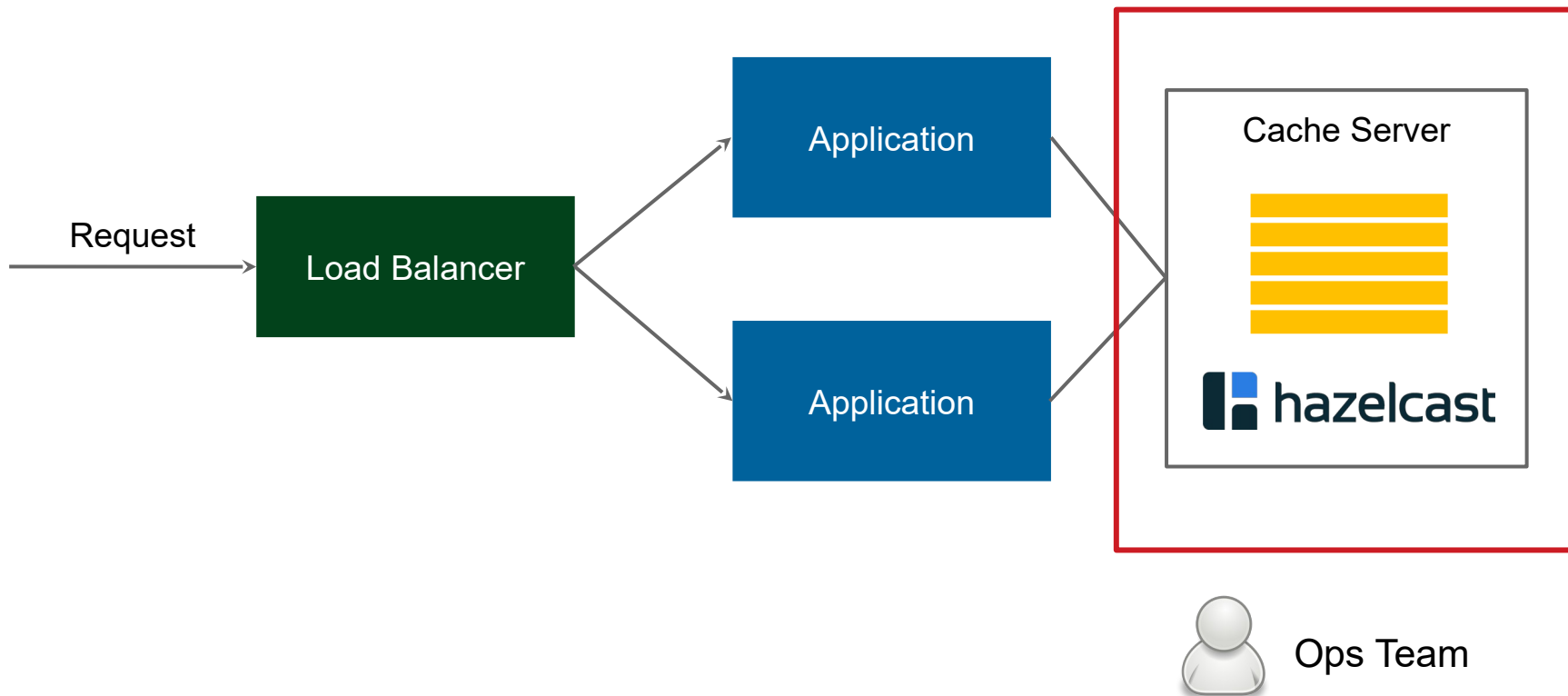
Client-Server Cache



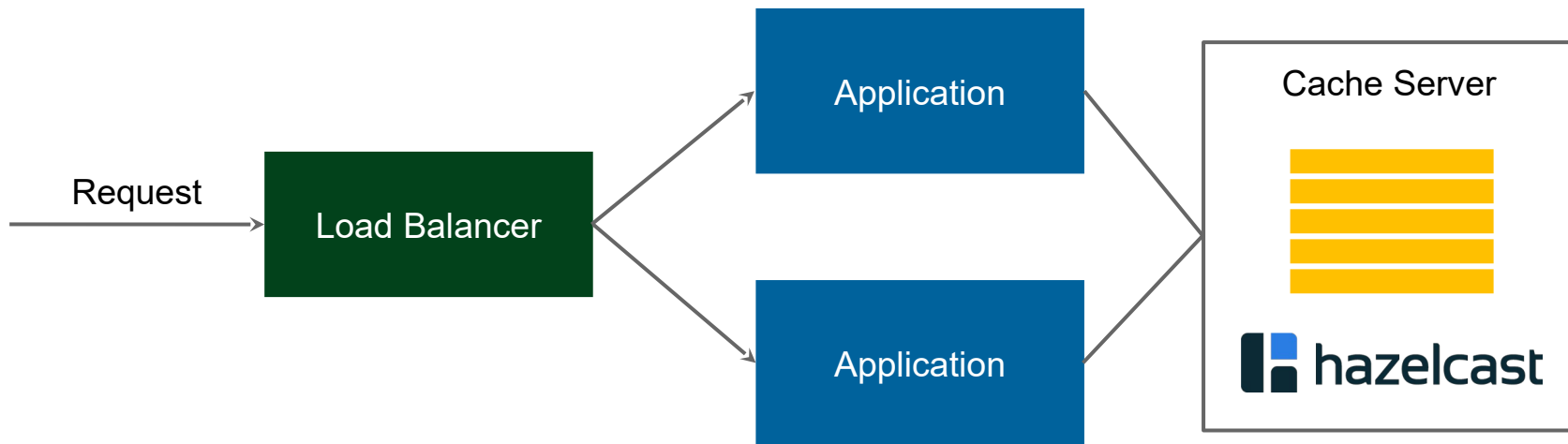
Client-Server Cache

Separate Management:

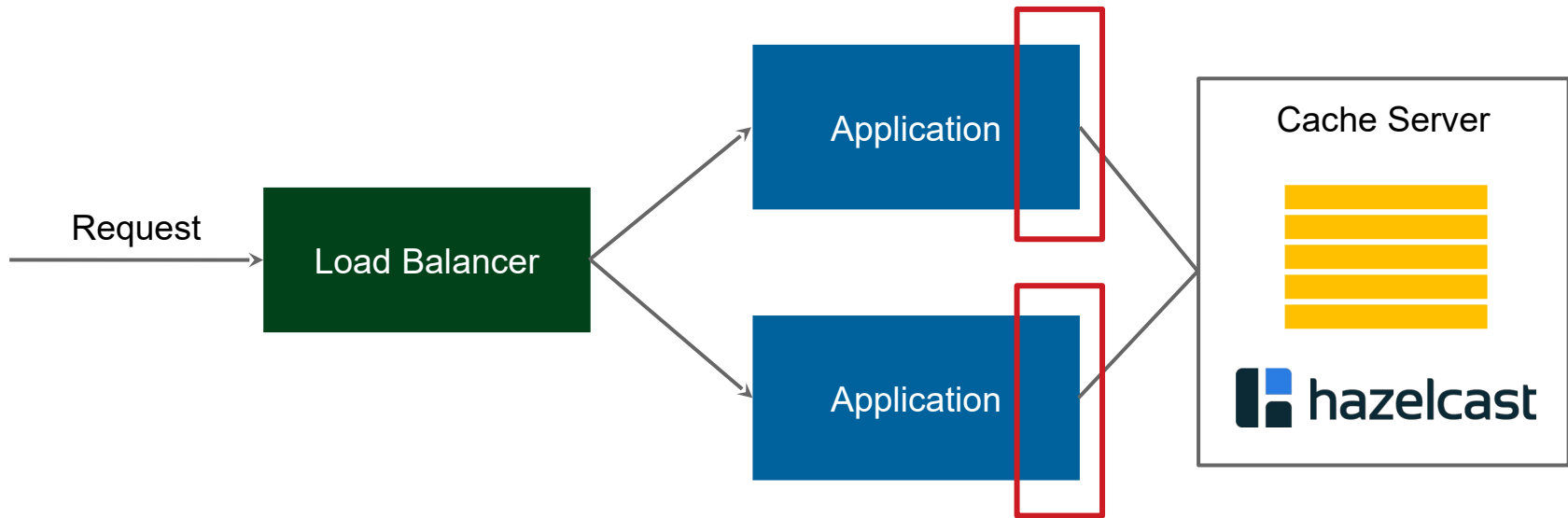
- backups
- (auto) scaling
- security



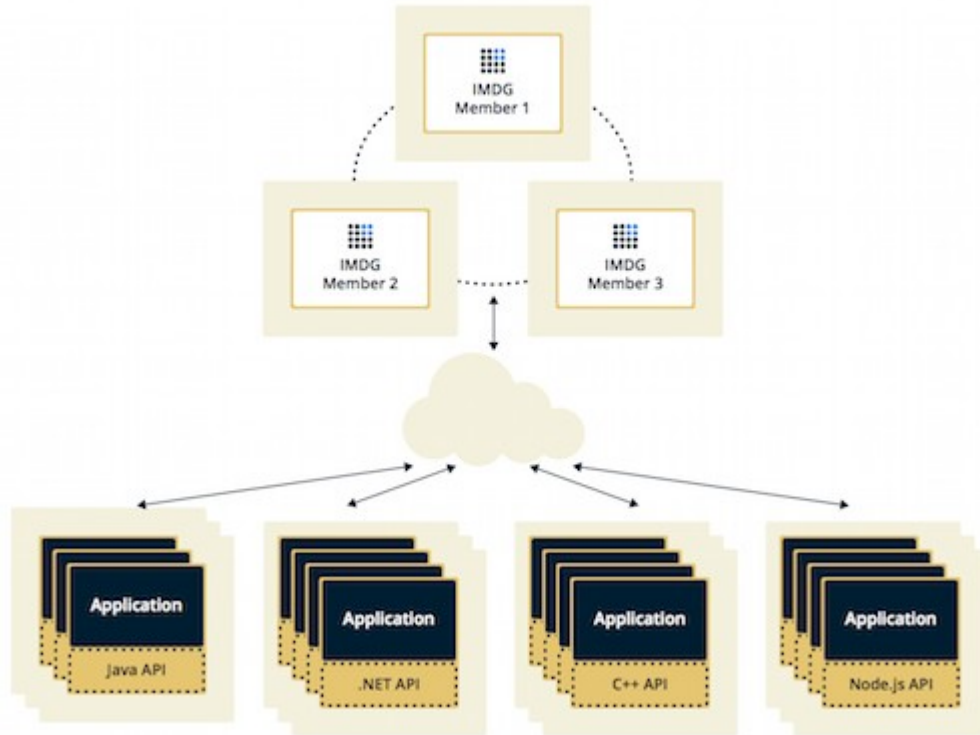
Client-Server Cache



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):

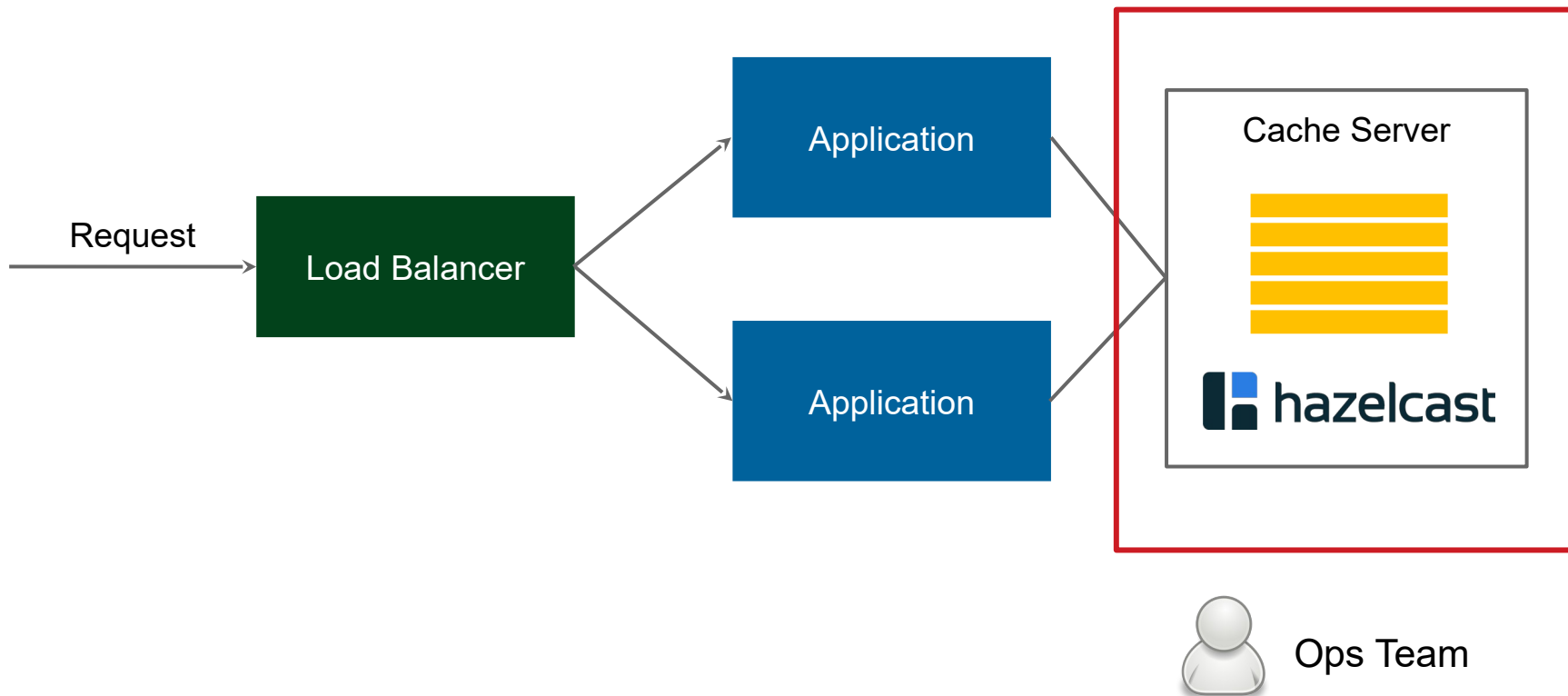
```
@Configuration
```

```
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:

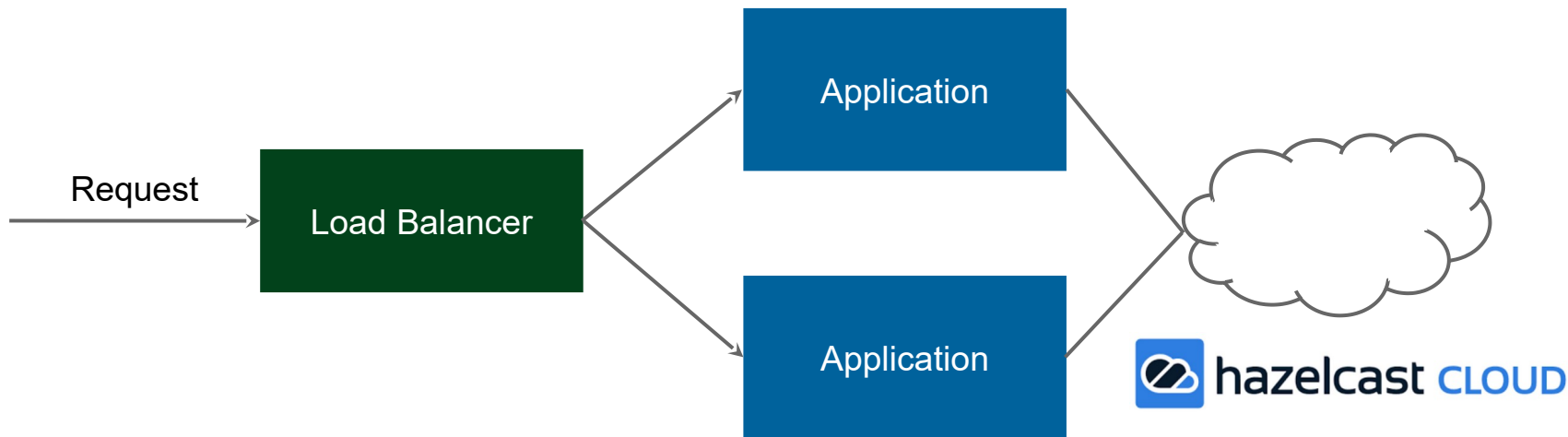
- backups
- (auto) scaling
- security



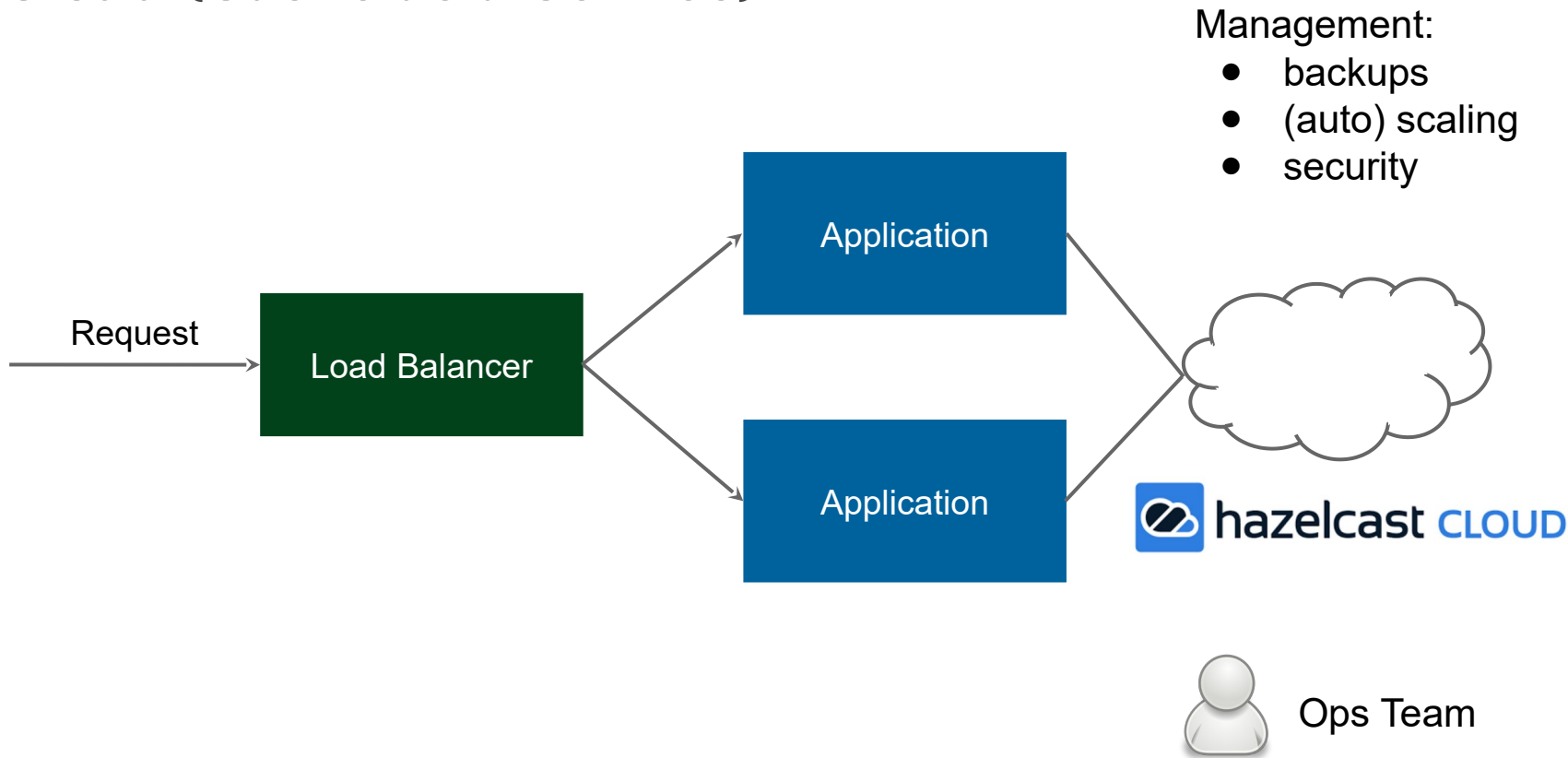


2*. Cloud

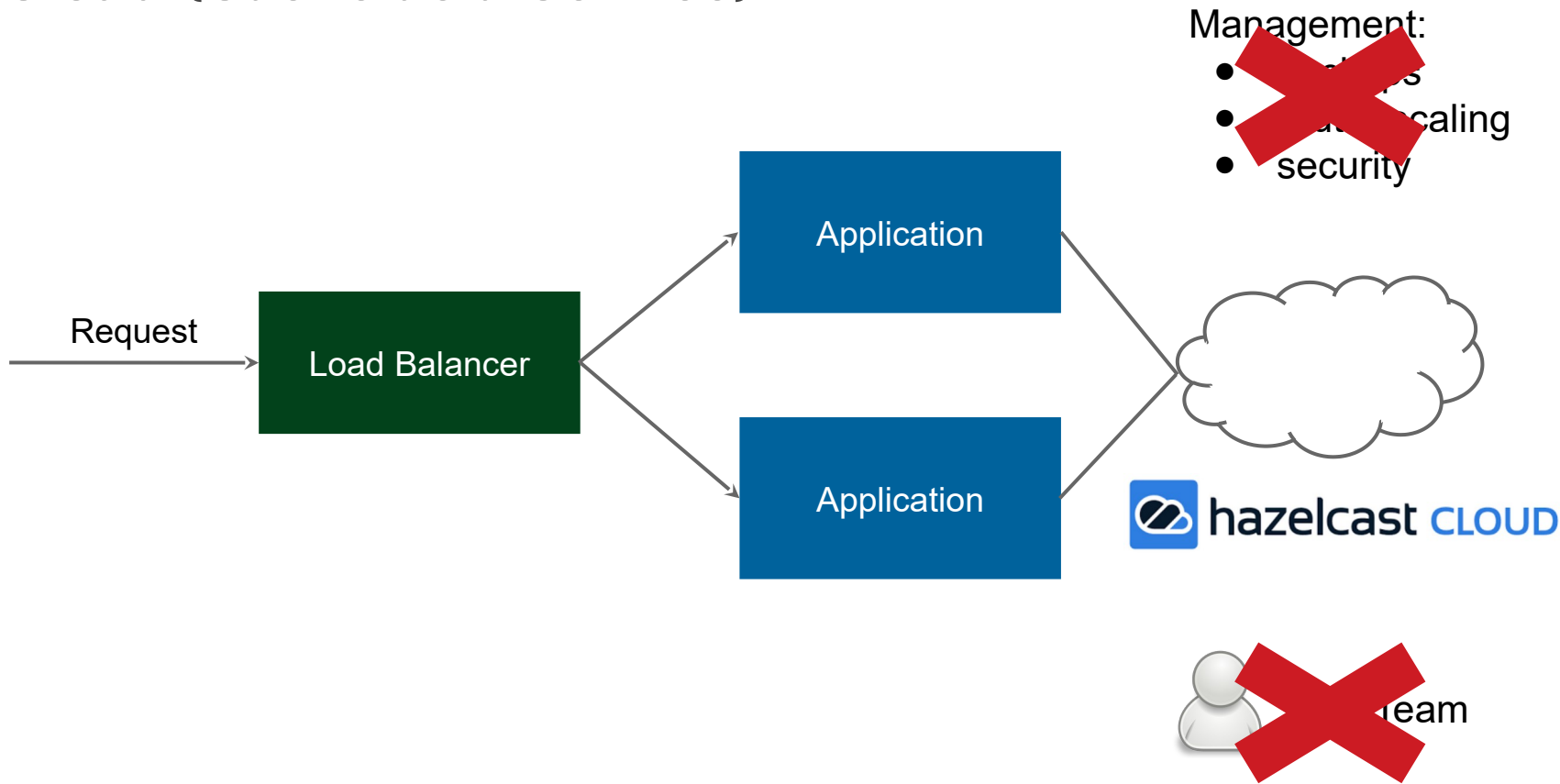
Cloud (Cache as a Service)



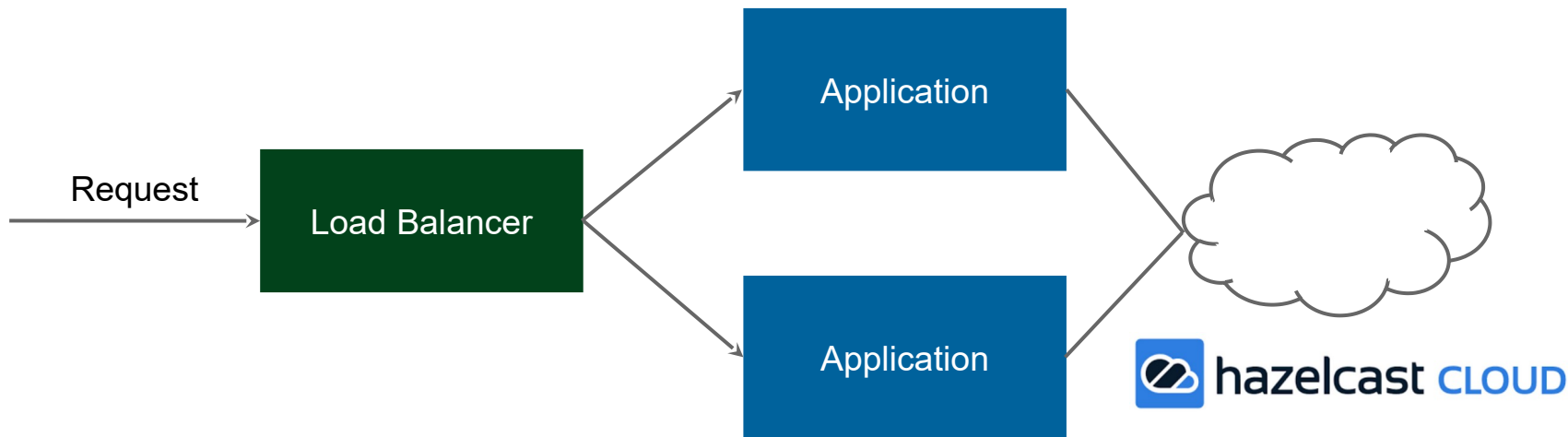
Cloud (Cache as a Service)



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)

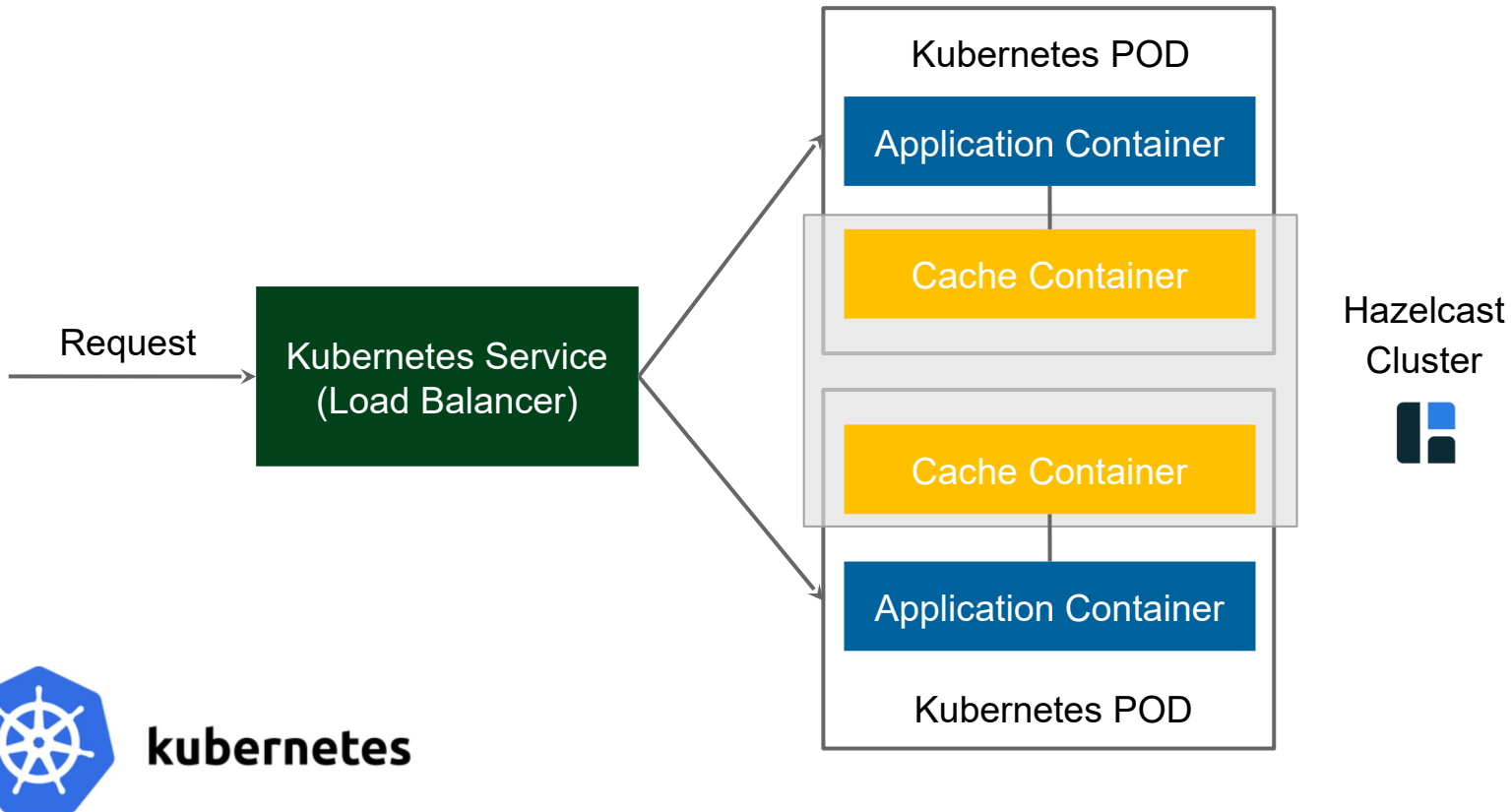
Agenda

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

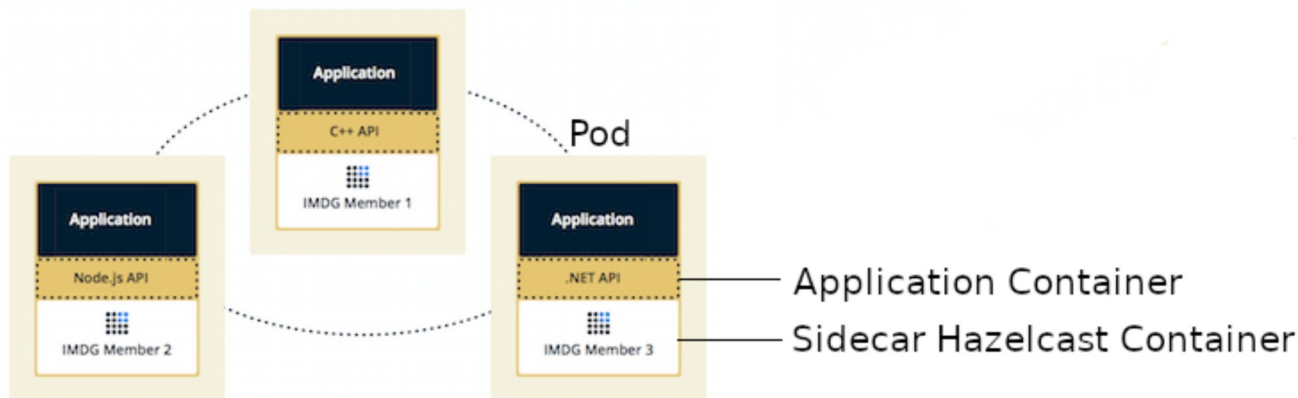


3. Sidecar

Sidecar Cache



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

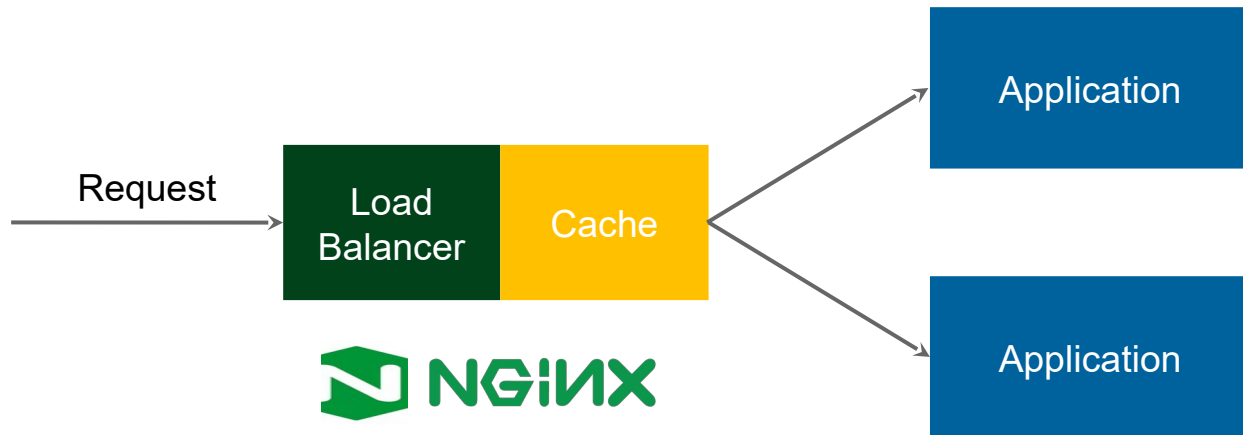
Agenda

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



4. Reverse Proxy

Reverse Proxy Cache



NGINX Reverse Proxy Cache Issues

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

NGINX Reverse Proxy Cache Issues

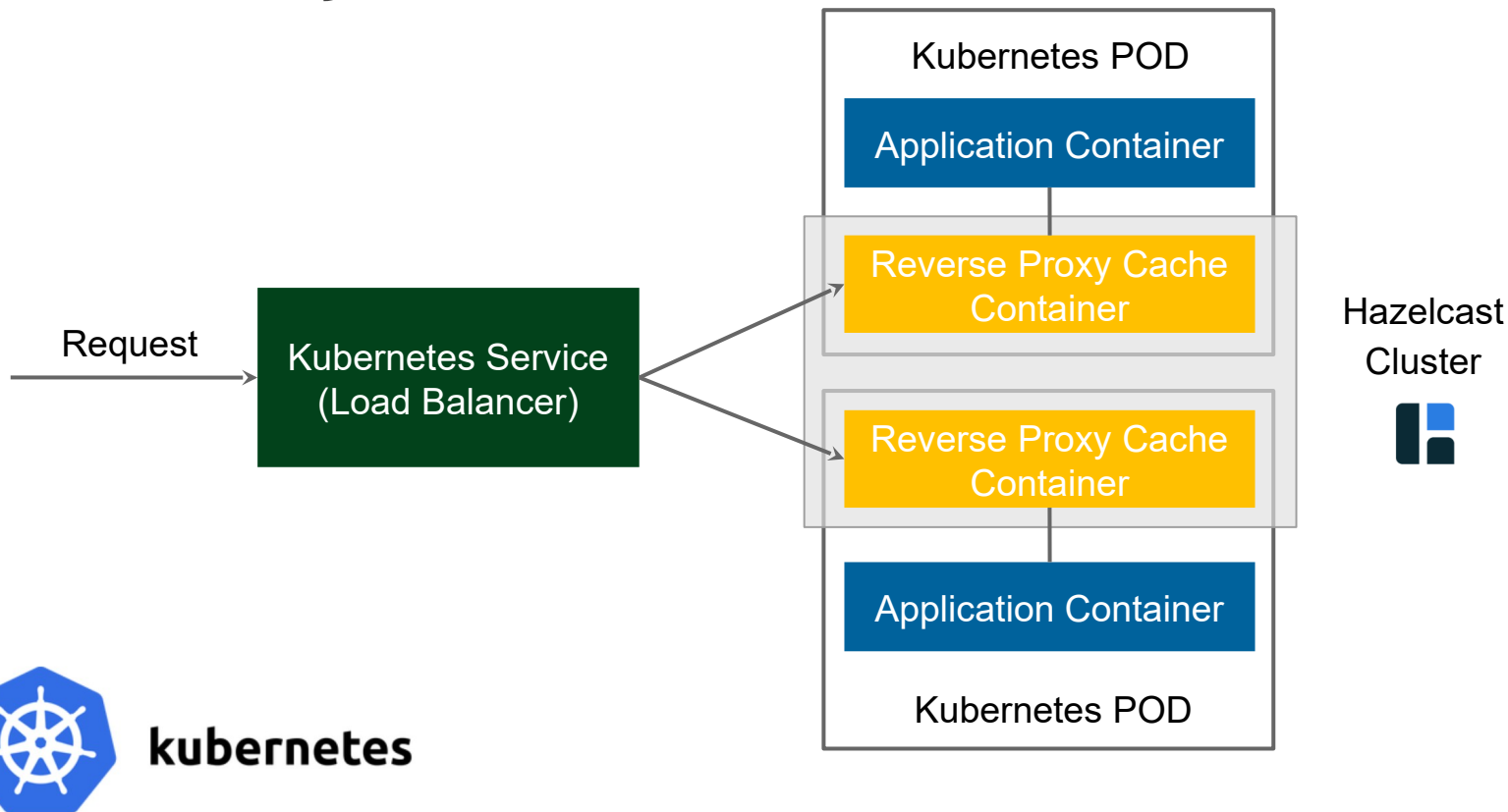
- Only for HTTP
- Not distributed
- No High Availability
- Data stored on the disk



The background of the slide is a photograph of numerous dates, likely Medjool, scattered across a piece of fabric with a colorful geometric pattern in shades of blue, red, and white. The dates are mostly a golden-brown color, indicating they are ripe. A white horizontal banner is superimposed over the center of the image, containing the text.

4*. Reverse Proxy Sidecar

Reverse Proxy Sidecar Cache

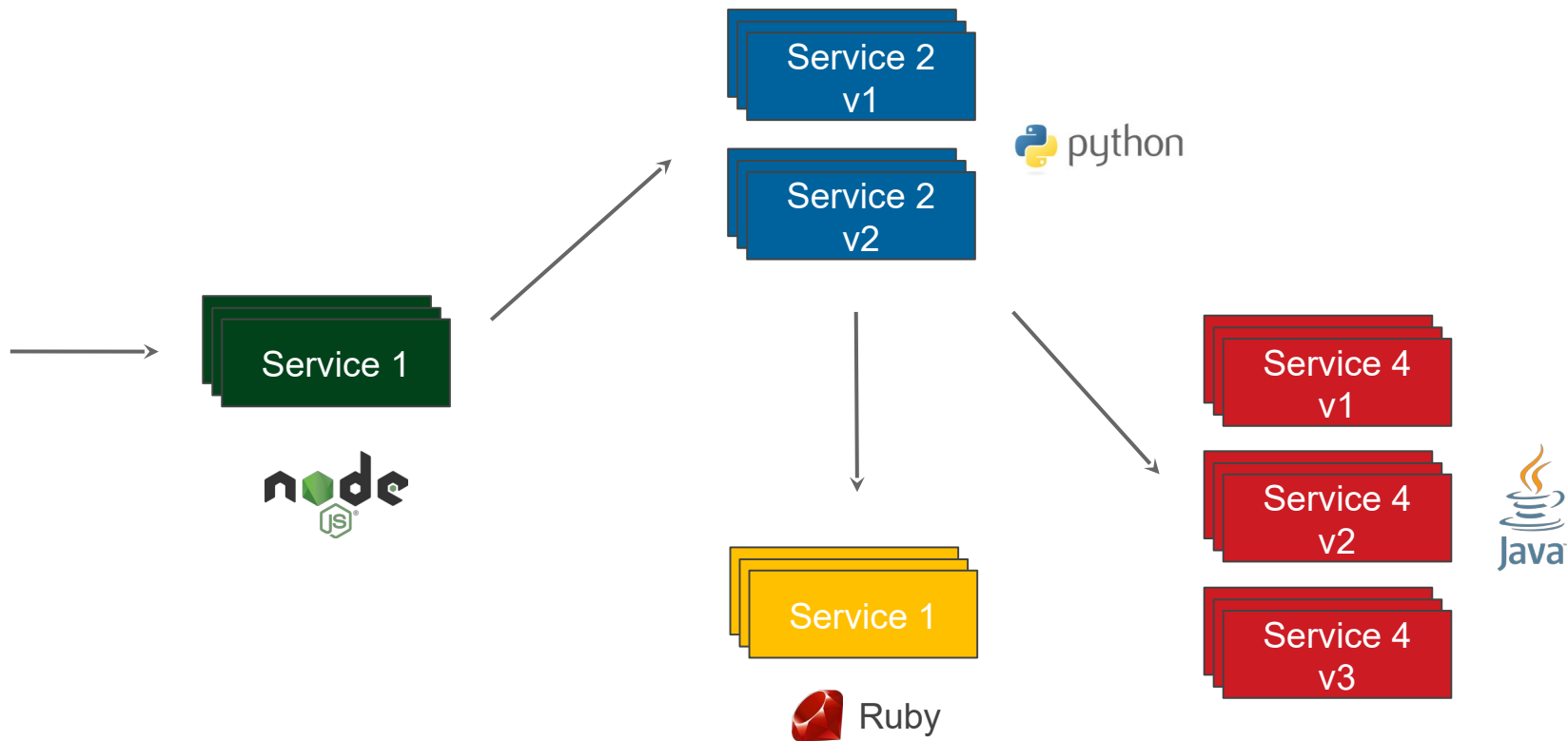




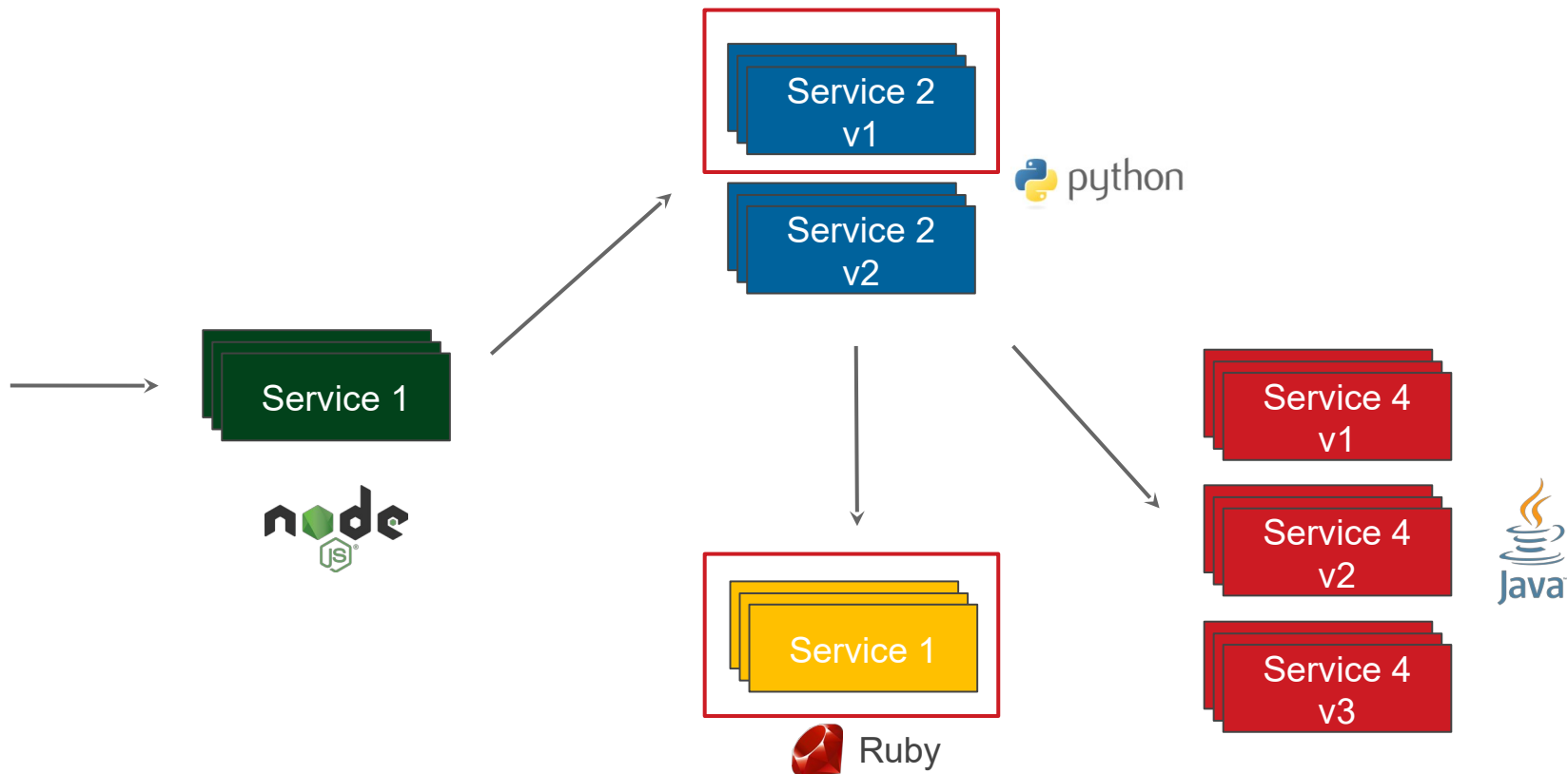


Good

Reverse Proxy Sidecar Cache



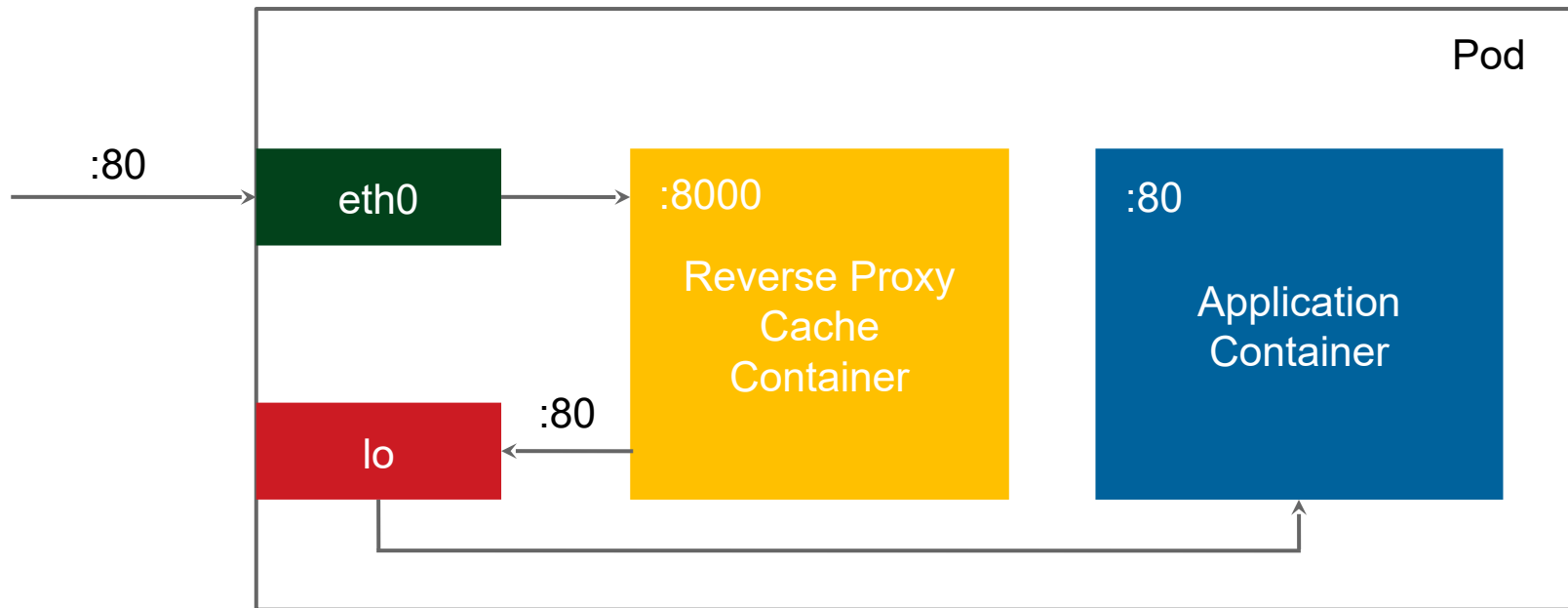
Reverse Proxy Sidecar Cache



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

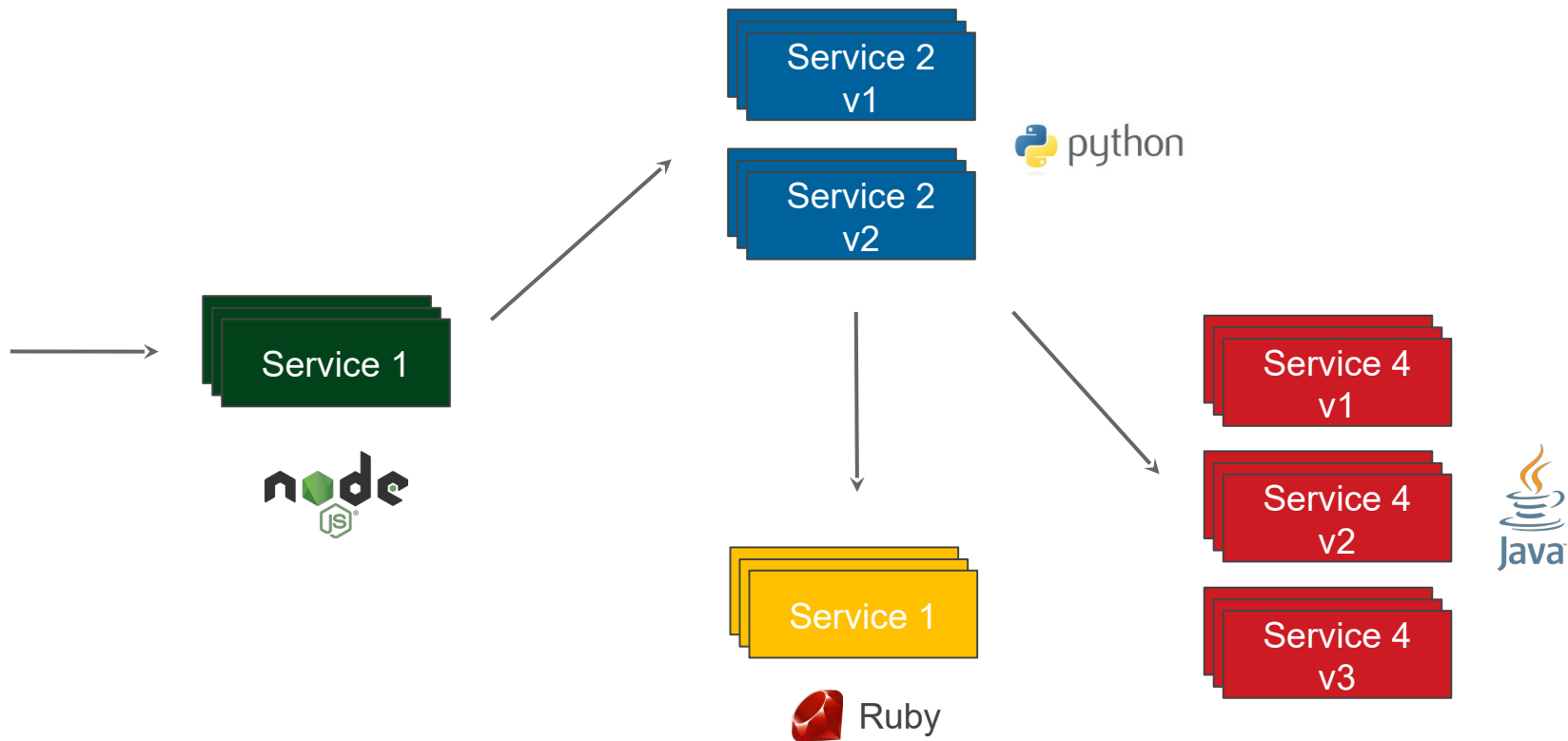


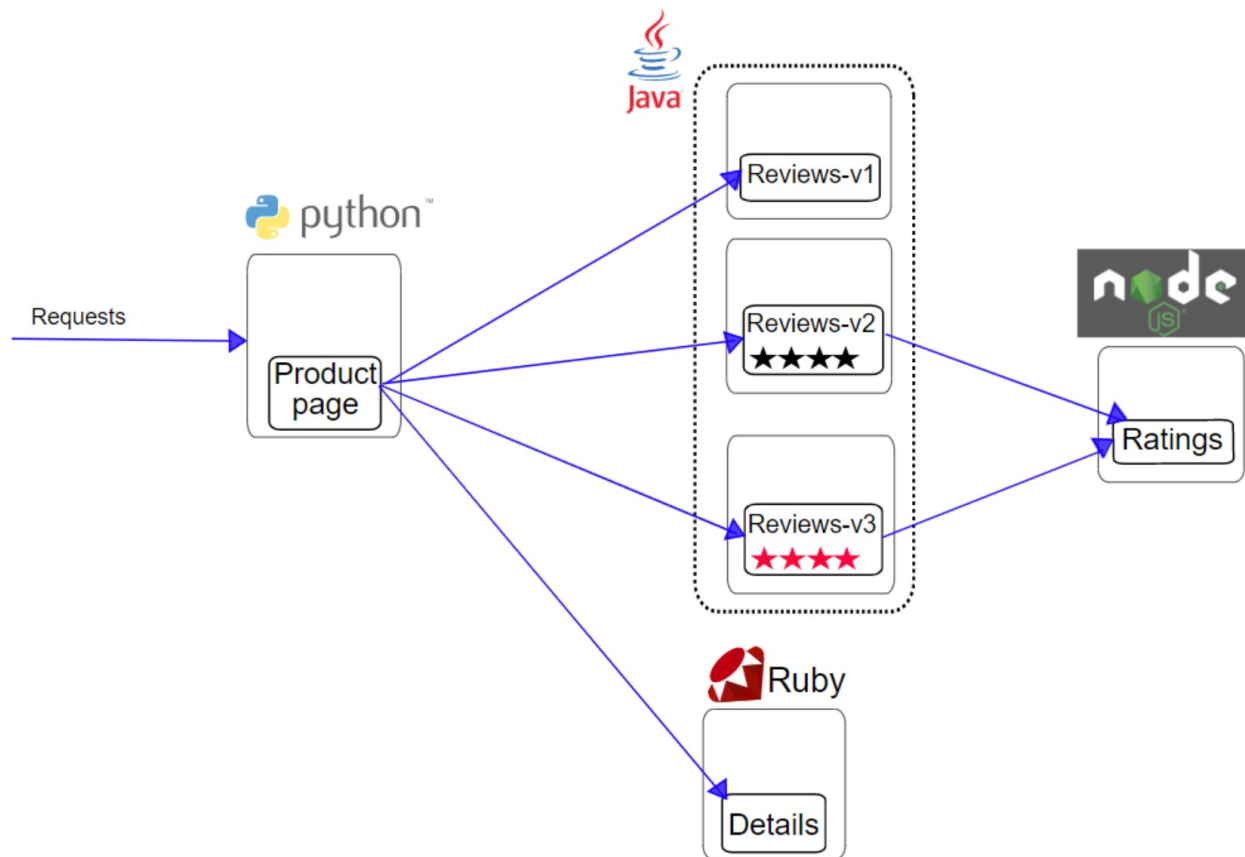
Reverse Proxy Sidecar Cache

```
#!/bin/bash
```


```
# Forward TCP traffic on port 80 to port 8000 on the eth0 interface.  
iptables -t nat -A PREROUTING -p tcp -i eth0 --dport 80 -j REDIRECT  
--to-port 8000
```


Reverse Proxy Sidecar Cache





Reverse Proxy Sidecar Cache (Istio)

 [envoyproxy](#) / [envoy](#)

[Watch](#) 509 [Star](#)

[Code](#) [Issues 438](#) [Pull requests 51](#) [Projects 1](#) [Insights](#)

Support http caching #868

 Open

tschroed opened this issue on May 1, 2017 · 10 comments



tschroed commented on May 1, 2017

Contributor



As a generic http proxy, it would be useful for Envoy to support http caching. It seems like this could probably be implemented as a filter.



21

Bad



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

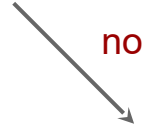
- Introduction ✓
- Caching Architectural Patterns
 - Embedded ✓
 - Embedded Distributed ✓
 - Client-Server ✓
 - Cloud ✓
 - Sidecar ✓
 - Reverse Proxy ✓
 - Reverse Proxy Sidecar ✓
- Summary



Summary

application-aware?

application-aware?



containers?

application-aware?

no

containers?

no

Reverse Proxy

```
graph TD; A[application-aware?] -- no --> B[containers?]; B -- no --> C[Reverse Proxy];
```

application-aware?

no

containers?

yes

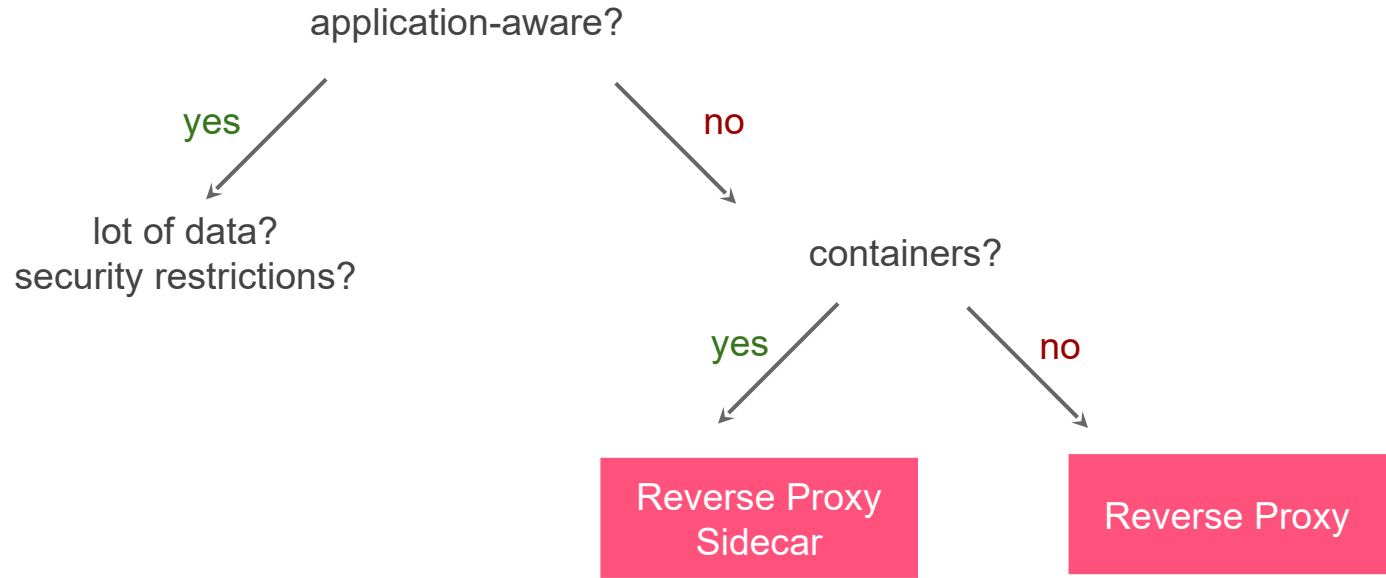
no

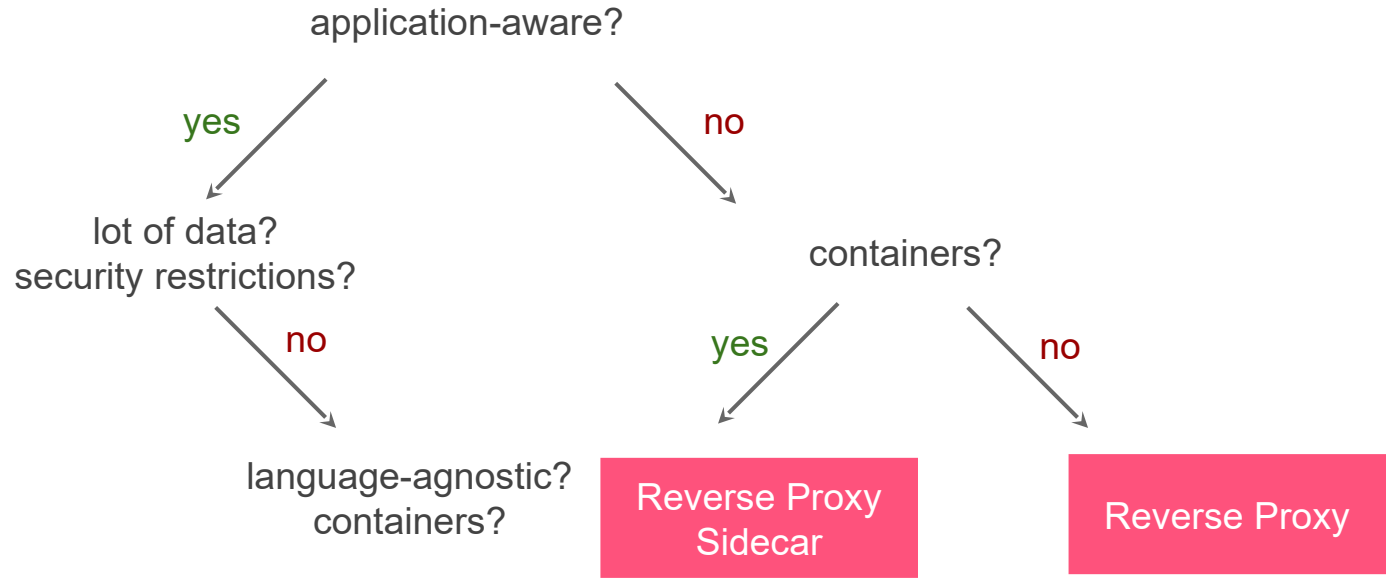
Reverse Proxy
Sidecar

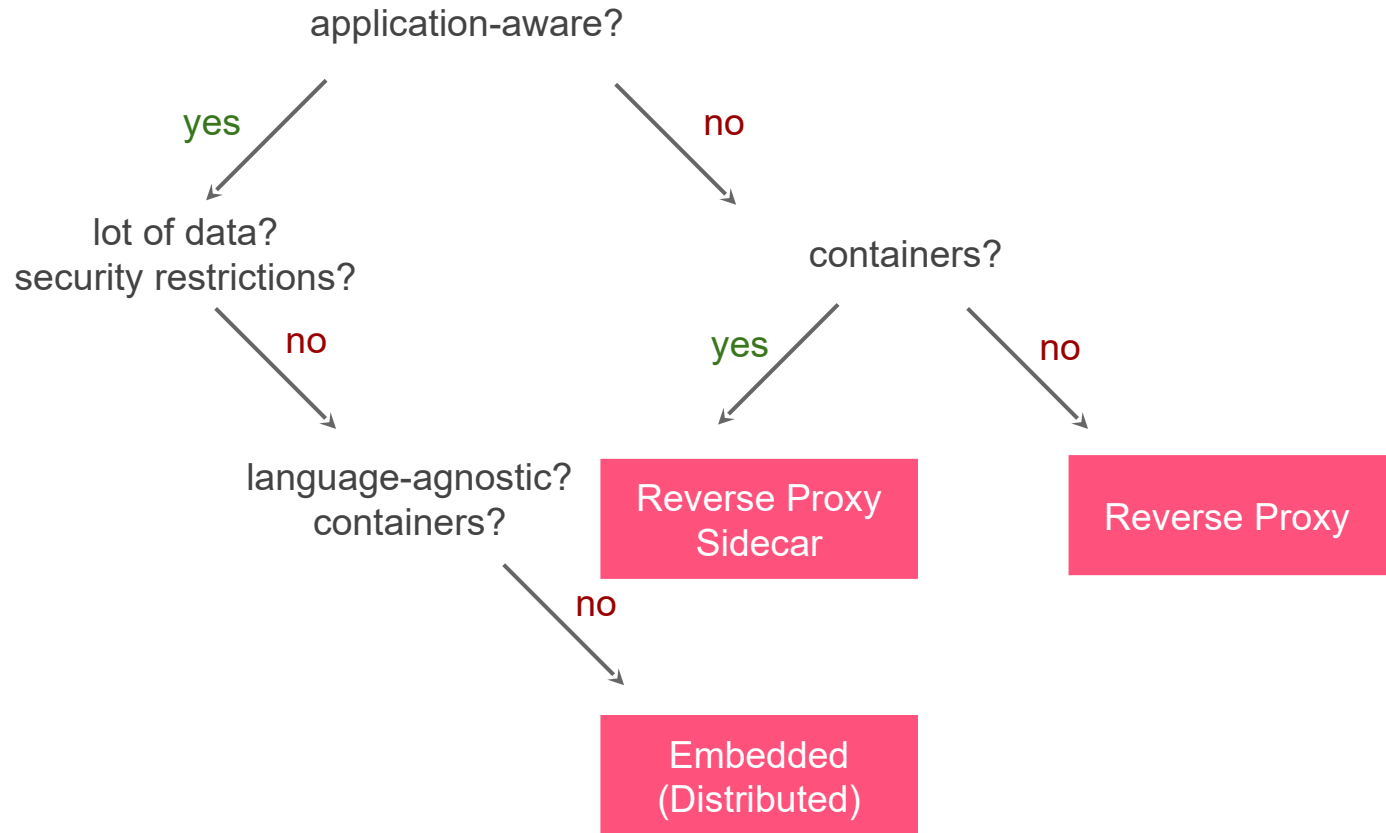
Reverse Proxy

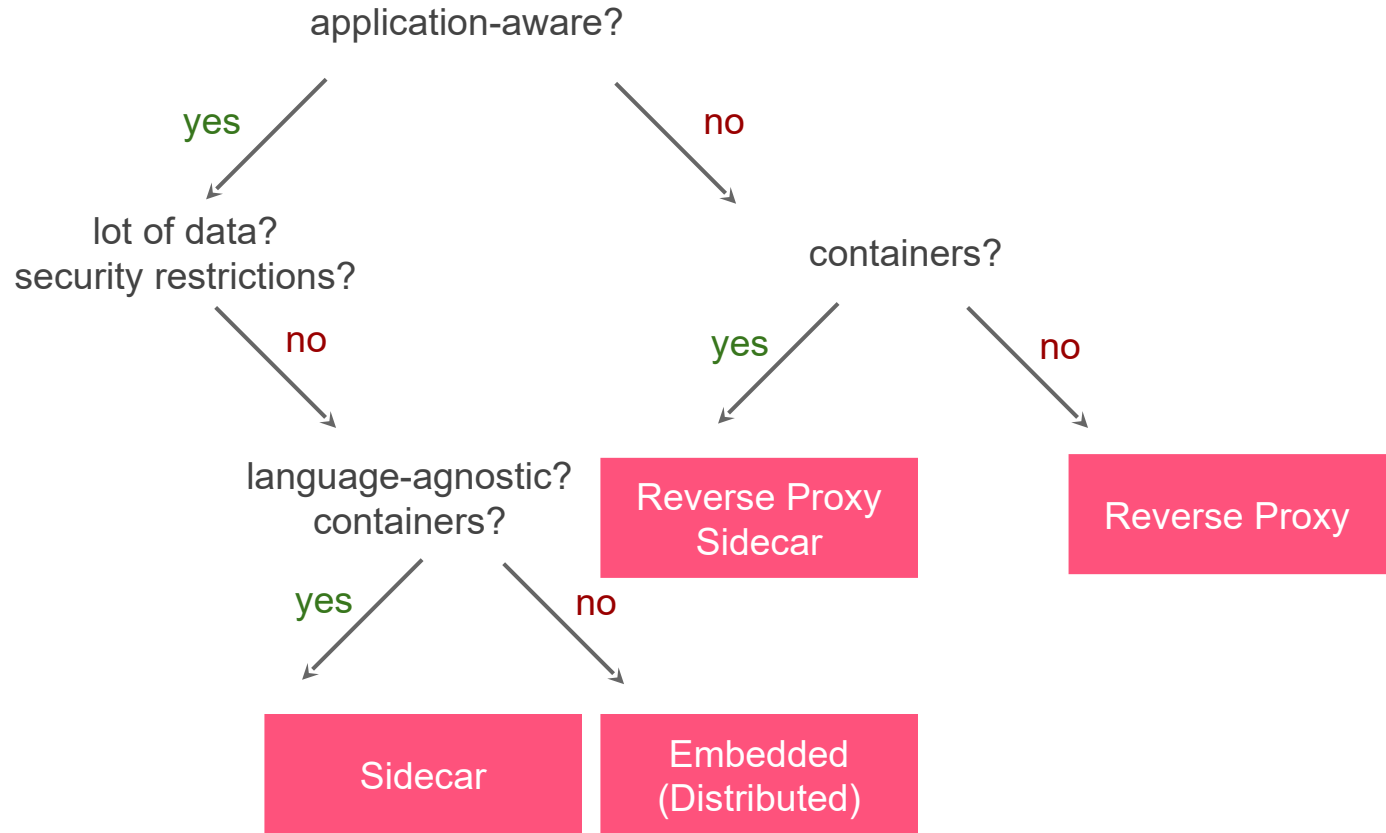
```
graph TD; A[application-aware?] -- no --> B[containers?]; B -- yes --> C[Reverse Proxy Sidecar]; B -- no --> D[Reverse Proxy];
```

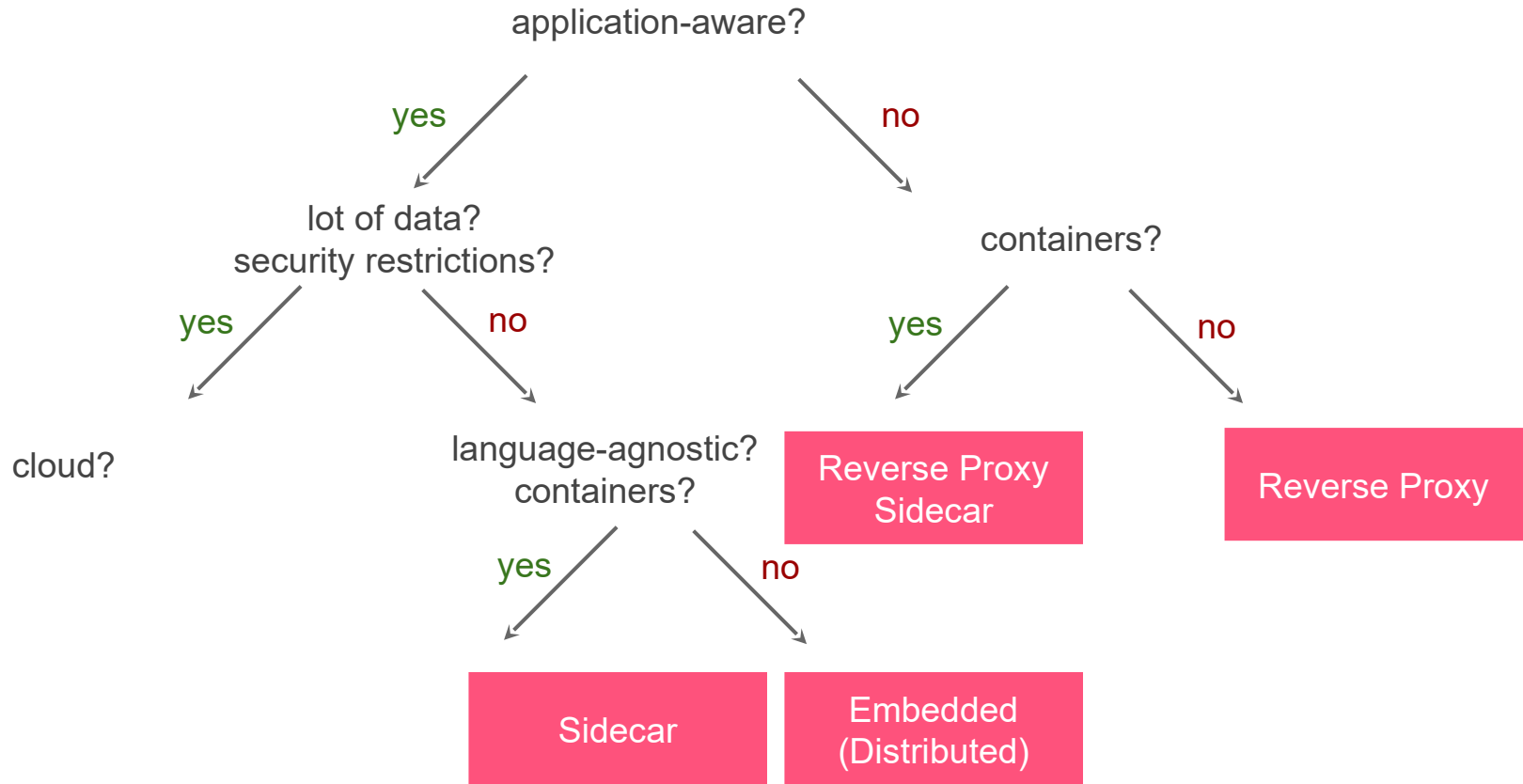
The diagram is a flowchart that starts with the question 'application-aware?'. If the answer is 'no', it leads to the question 'containers?'. From 'containers?', if the answer is 'yes', it leads to a pink box labeled 'Reverse Proxy Sidecar'. If the answer is 'no', it leads to a pink box labeled 'Reverse Proxy'. The 'no' and 'yes' labels are in red and green respectively, while the final outcomes are in white text on pink backgrounds.

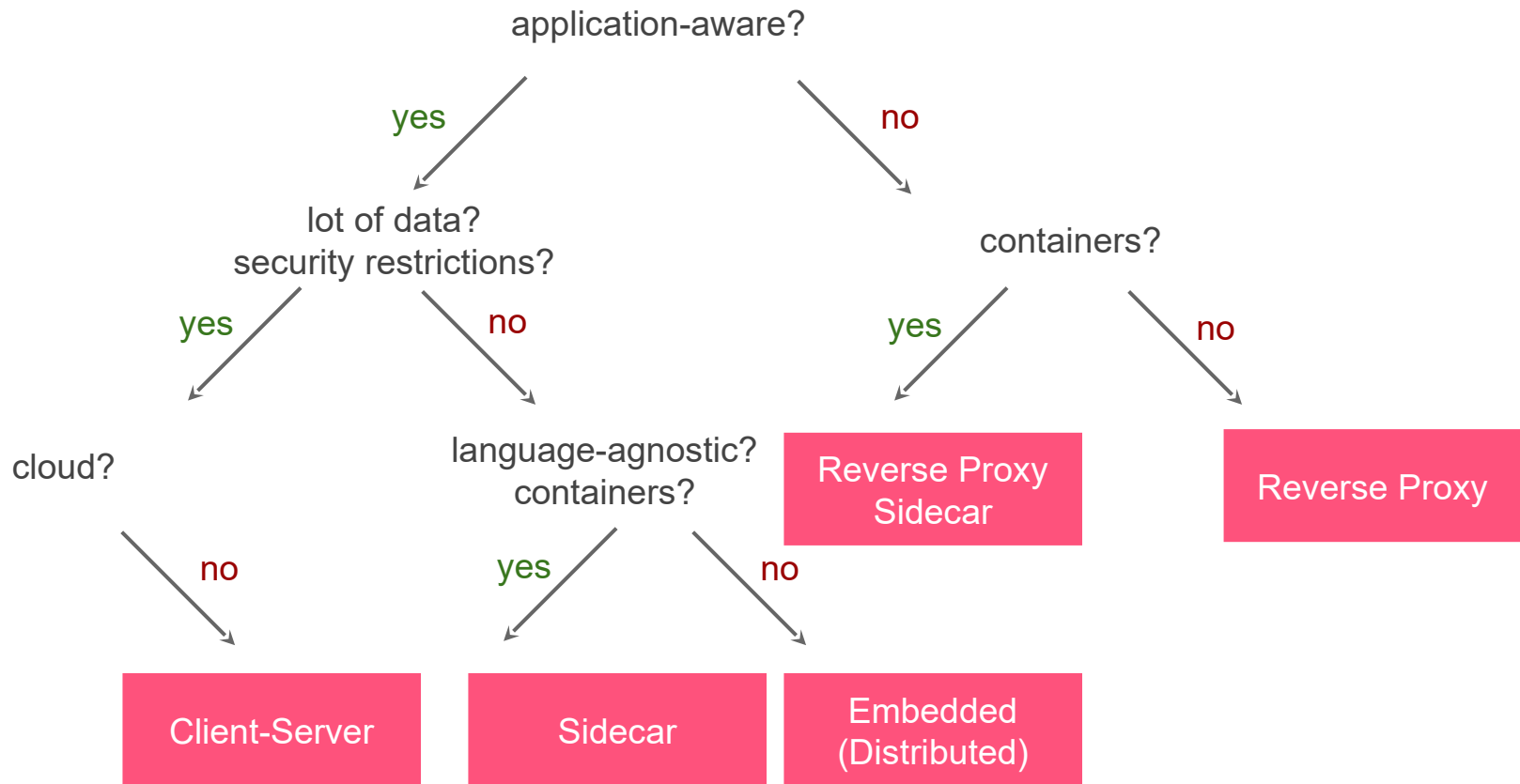


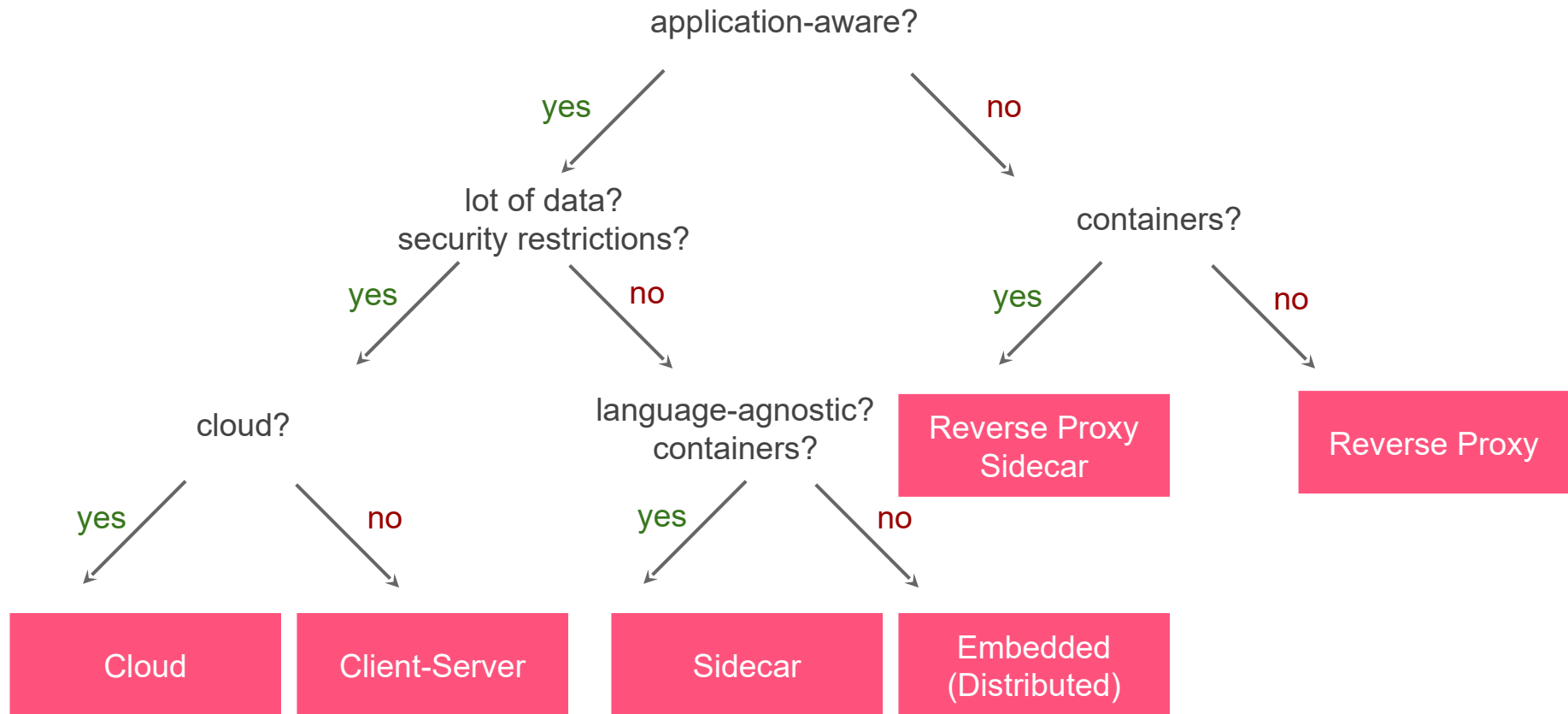












Resources

- 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



@RafałLeszko

rafalleszko.com