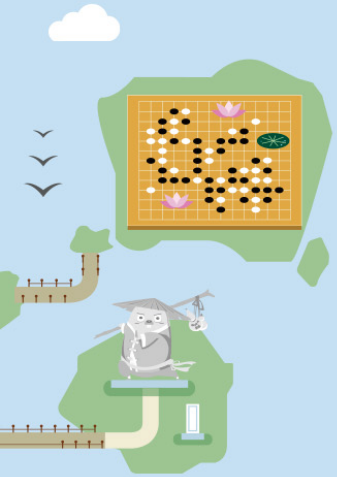


Harbor开源项目
容器镜像远程复制的实现

Henry Zhang (张海宁)
Chief Architect
VMWare China

GopherChina 2017

自我介绍

- VMware中国研发首席架构师
- Harbor开源企业级容器Registry项目创始人
- Cloud Foundry中国社区最早技术布道师之一
- 多年全栈工程师
- 《区块链技术指南》、《软件定义存储》作者之一

亨利笔记

《区块链技术指南》

《软件定义存储》

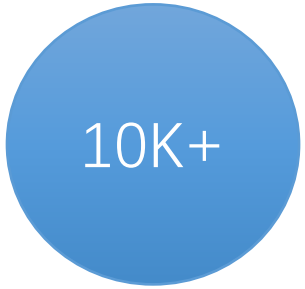
Introducing Project Harbor

- An open source enterprise-class registry server. (launched Mar 2016)
- Initiated by VMware China
- Apache 2 license
- https://github.com/vmware/harbor/

Project Harbor and Golang

- Harbor uses and grows with Go language from Day 1
  - Go v1.3-1.7
  - Beego: v1.3-1.6
- A member project of Golang Foundation

Harbor Users and Partners



Downloads

Stars

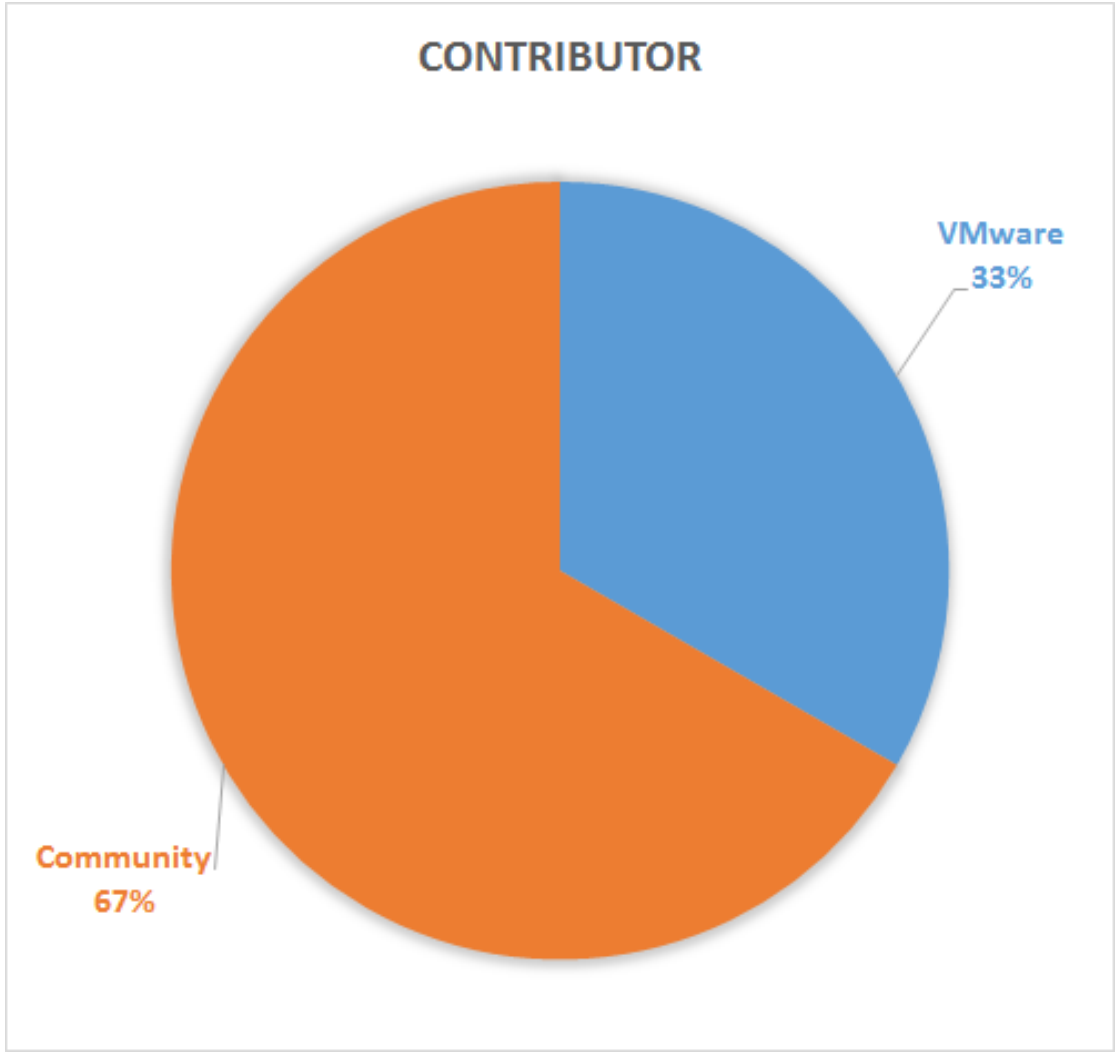
Users

Forks

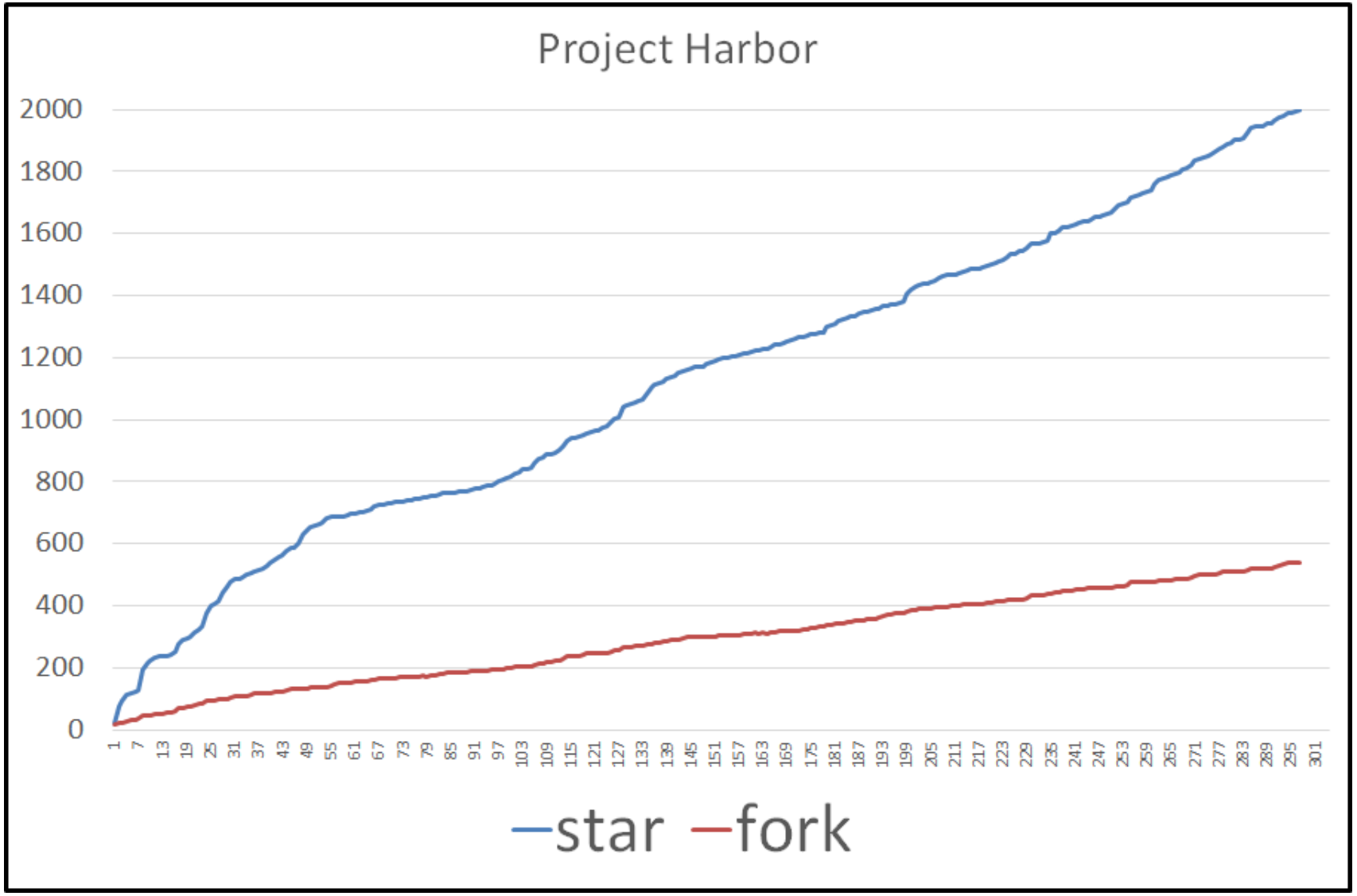
Contributors

Partners

Harbor Contributors Worldwide



Harbor Adoption



Key Users and Partners

HARBOR™

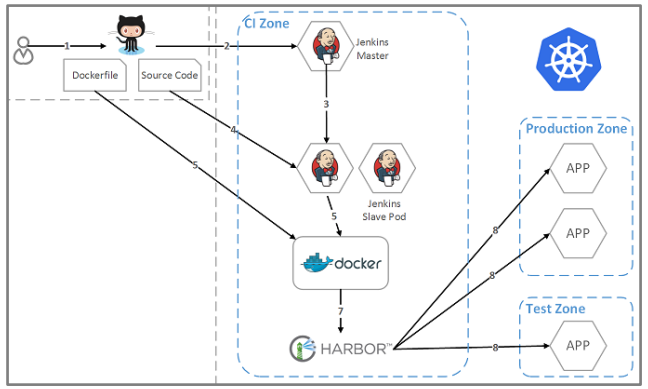
- **Users**

EMC² IBM Hewlett Packard Enterprise 中国移动 China Mobile China unicom中国联通 iSoftStone 软通动力 AsiaInfo 亚信

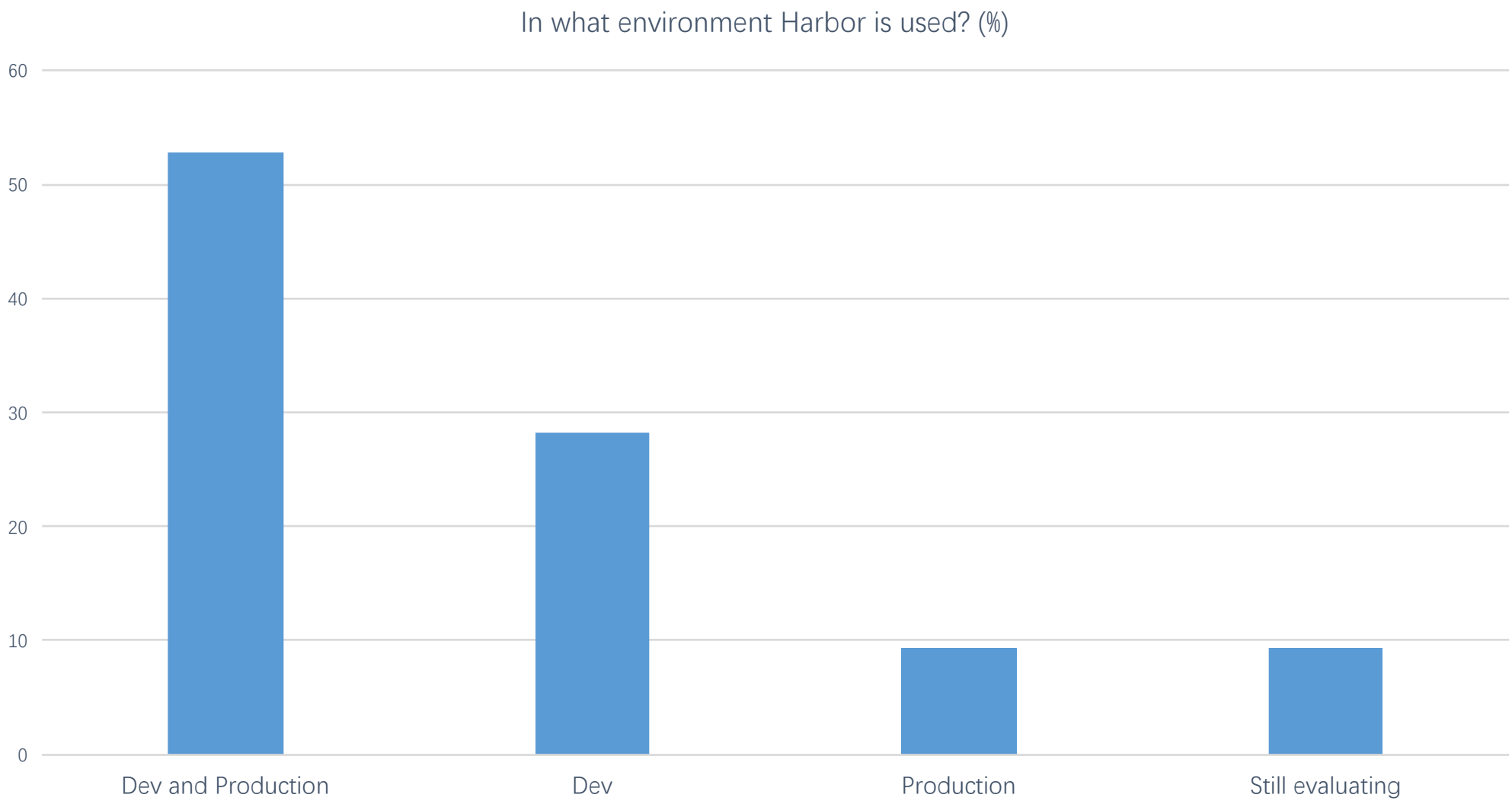
JD.京东.COM Tencent 麻袋理财 钱多多 D.com.cn 途牛 tuniu.com SLAMTEC 点融网 Dianrong.com

- **Partners**

数人云 shurenyun.com DaoCloud RANCHER Caicloud 才云科技

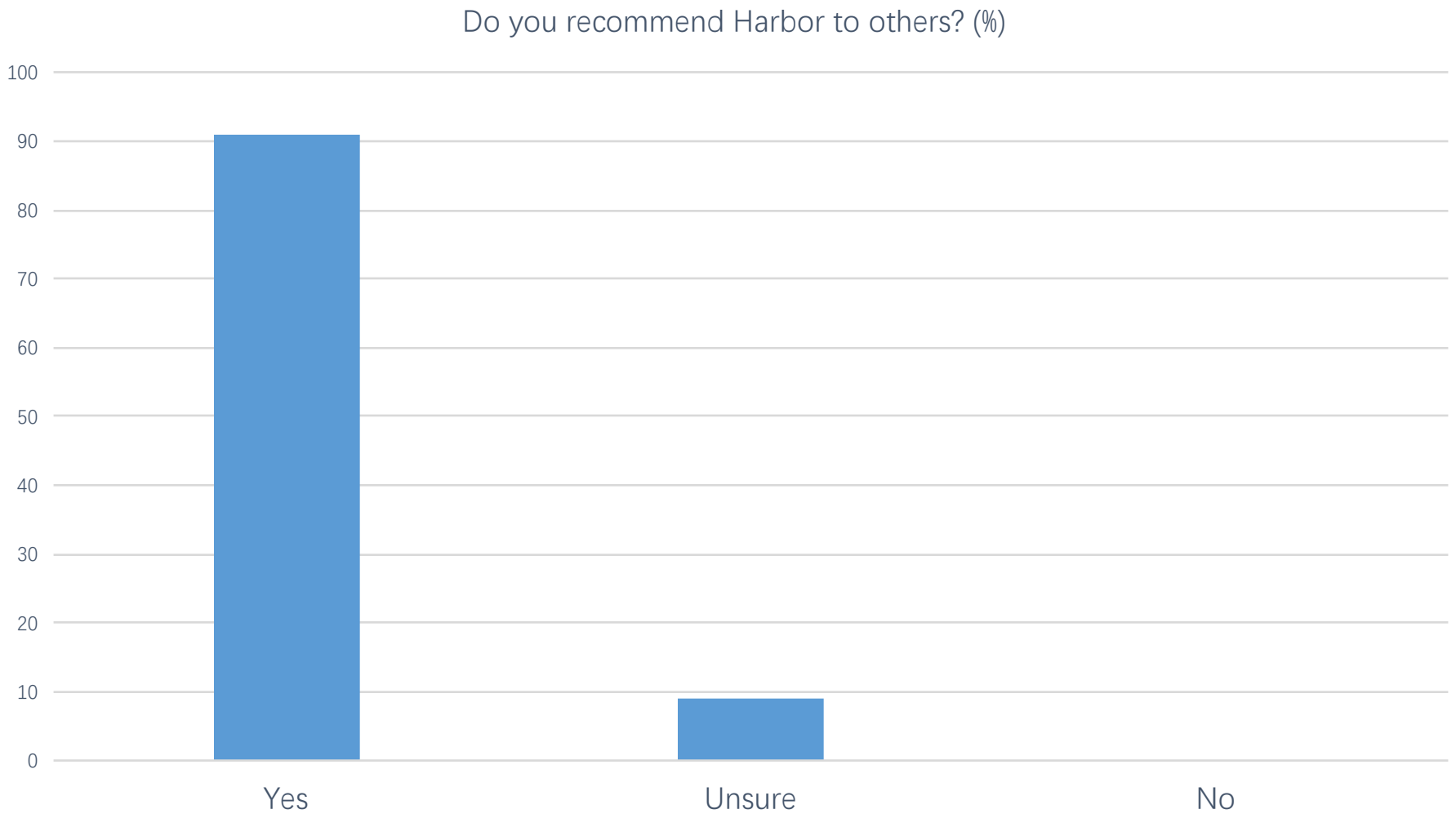


Harbor used in Production and Dev



Survey based on Chinese user community, 53 responses

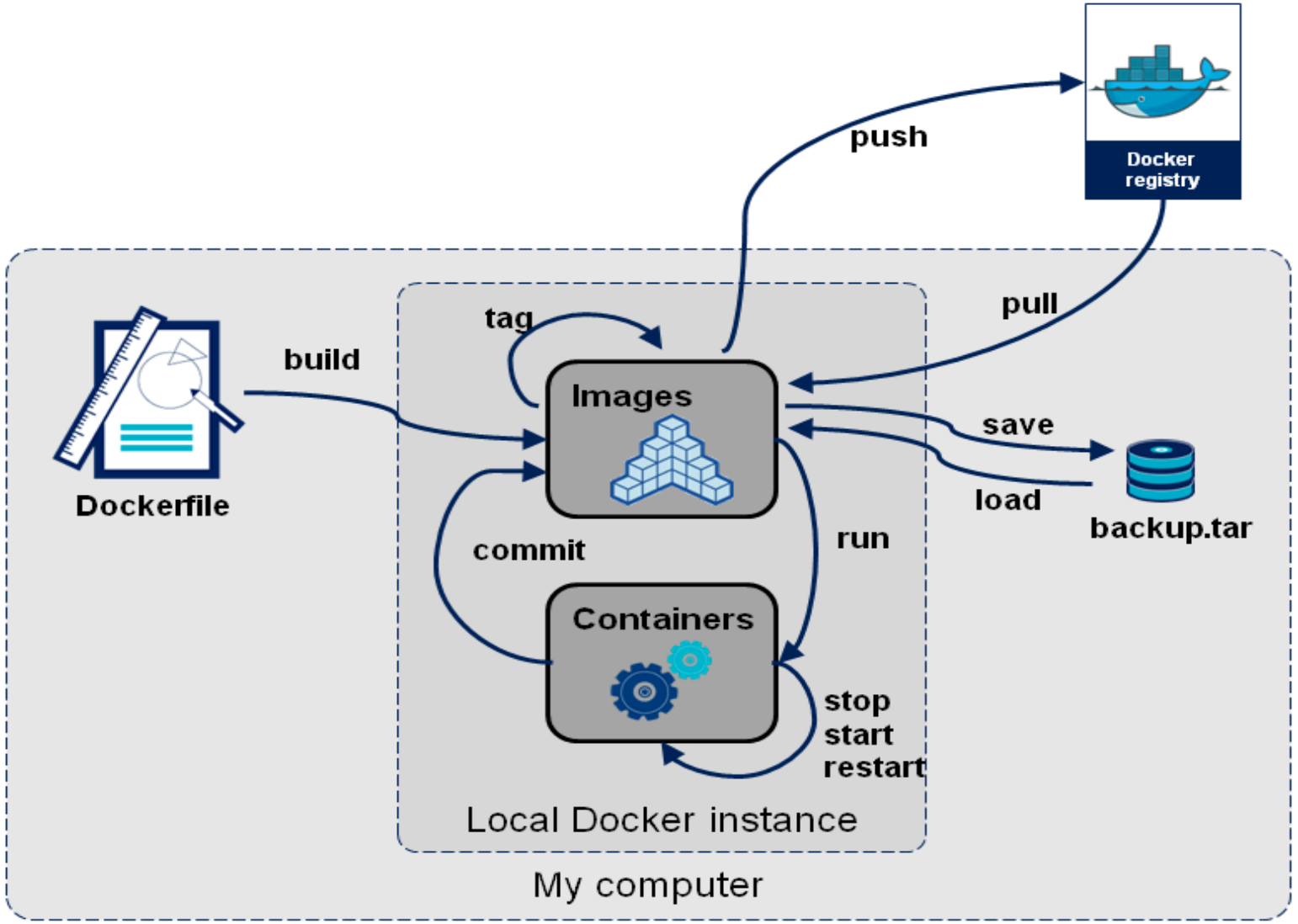
Do you recommend Harbor?



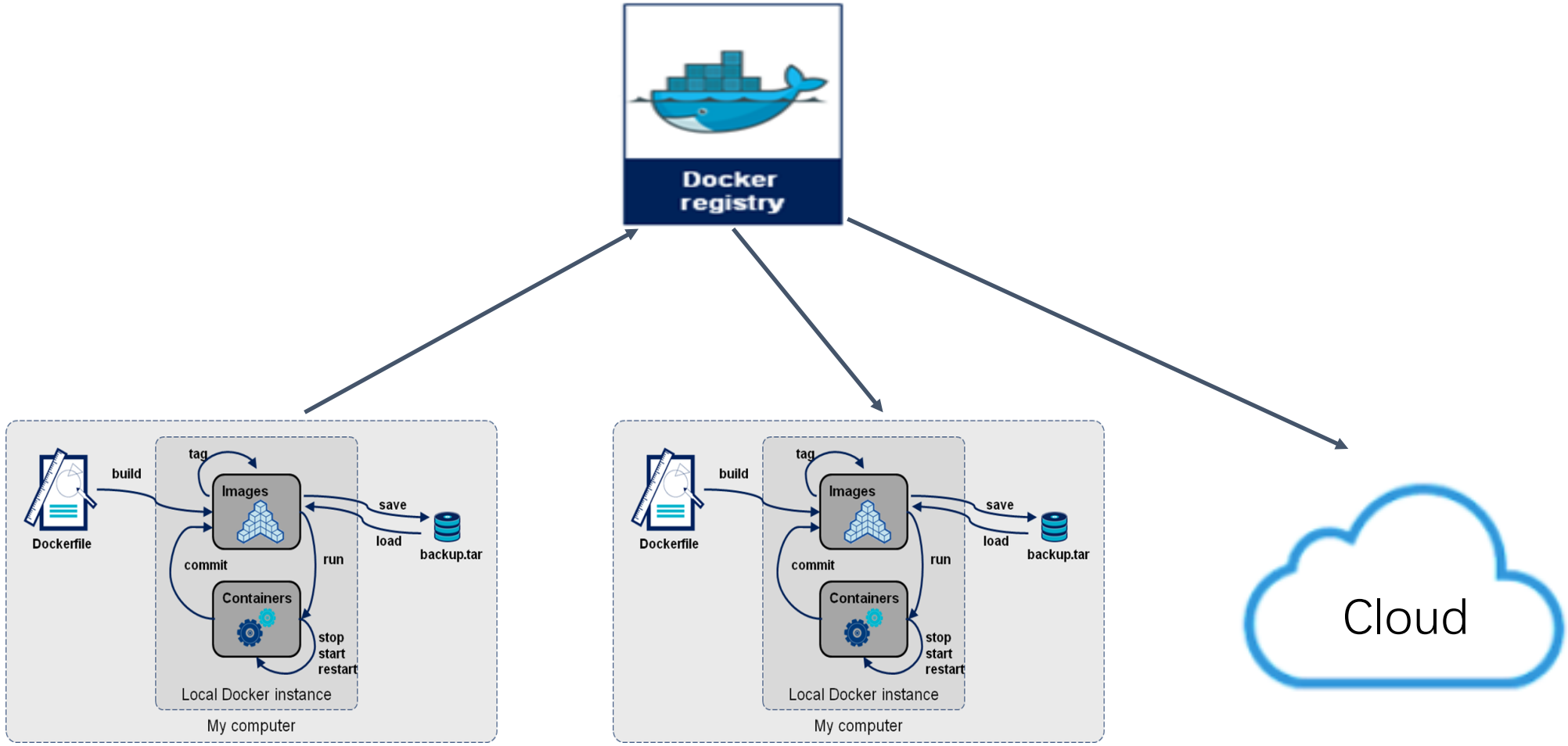
Survey based on Chinese user community, 53 responses

Docker Container Lifecycle: Build-Ship-Run

HARBOR™



Build-Ship-Run through Registry



- Registry is a key component of devops

Harbor : Enterprise-Class Private Registry

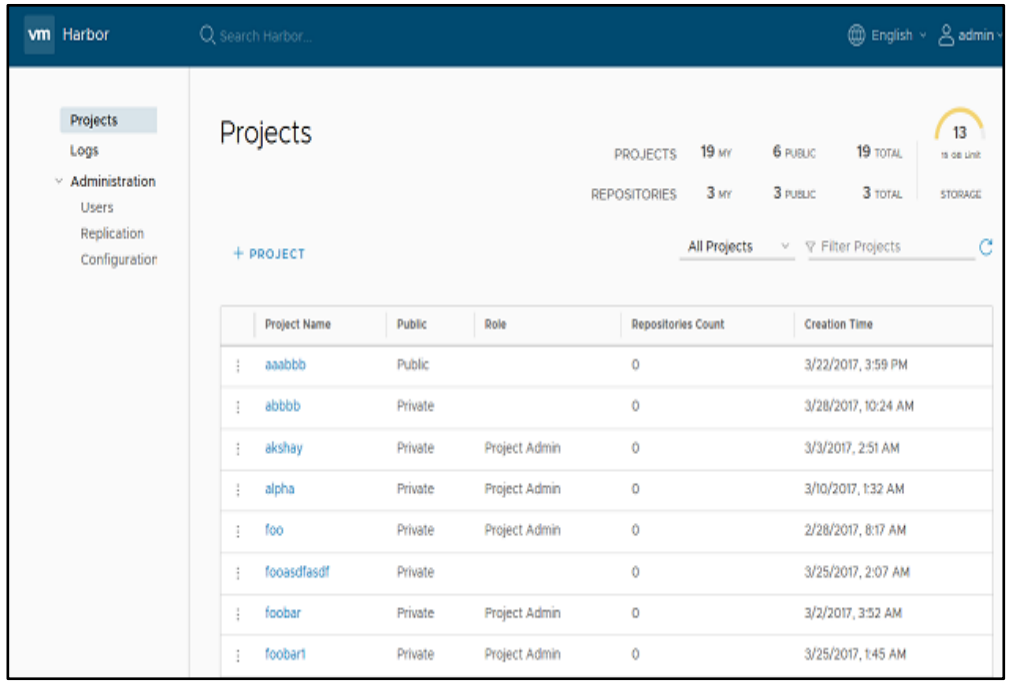
Why does one need a private registry?

- Efficiency
  - LAN vs WAN
- Security
  - Intellectual property stays in organization
  - Access Control

Enterprise Oriented Features

HARBOR™

- User management & access control
  - RBAC: admin, developer, guest
  - AD/LDAP integration
- Policy based image replication
- Web UI (中文 and English)
- Audit and logs
- Restful API for integration
- Lightweight and easy deployment



Project Harbor - Microservices Architecture

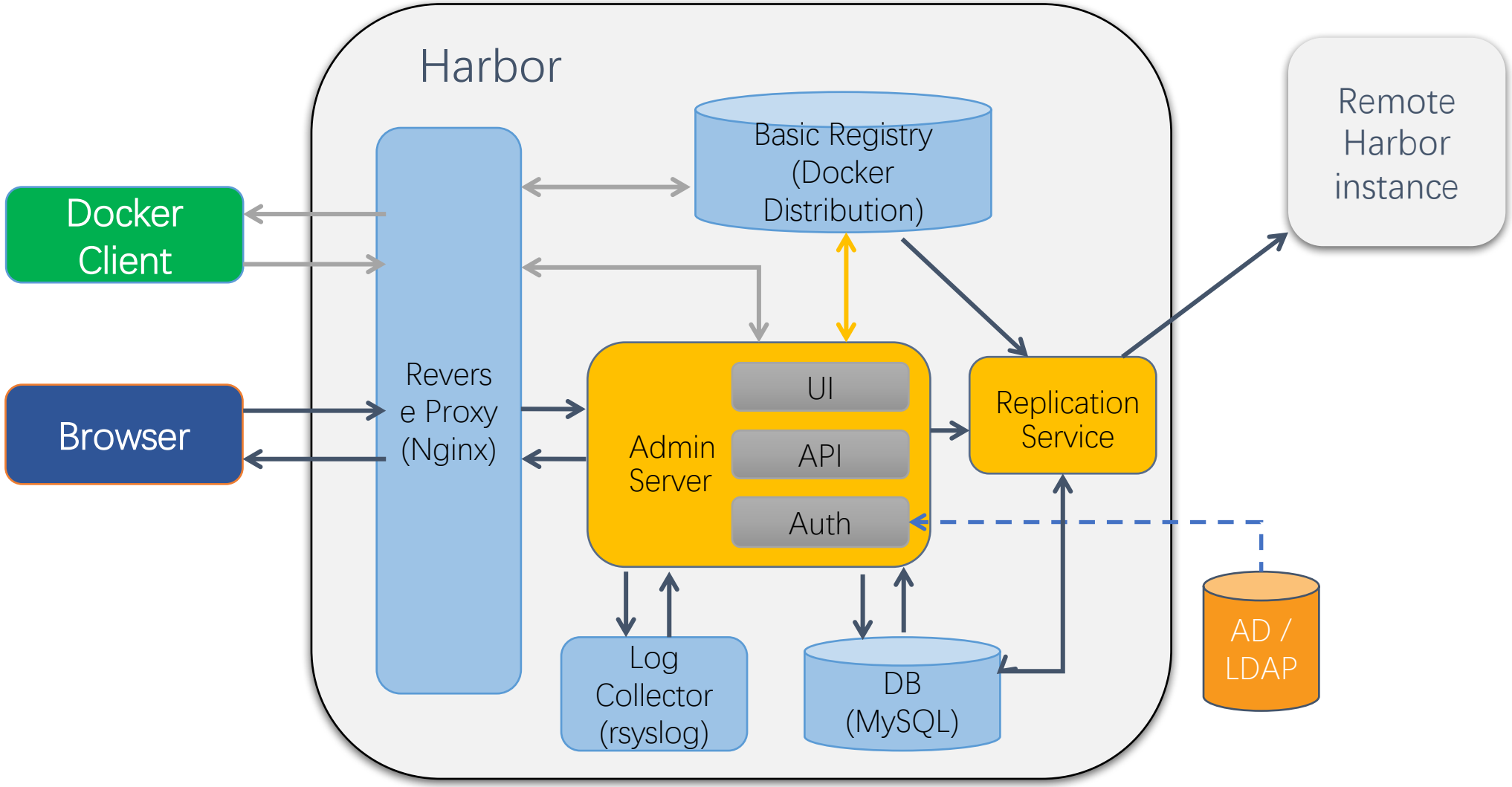


Image Replication between Registry Instances

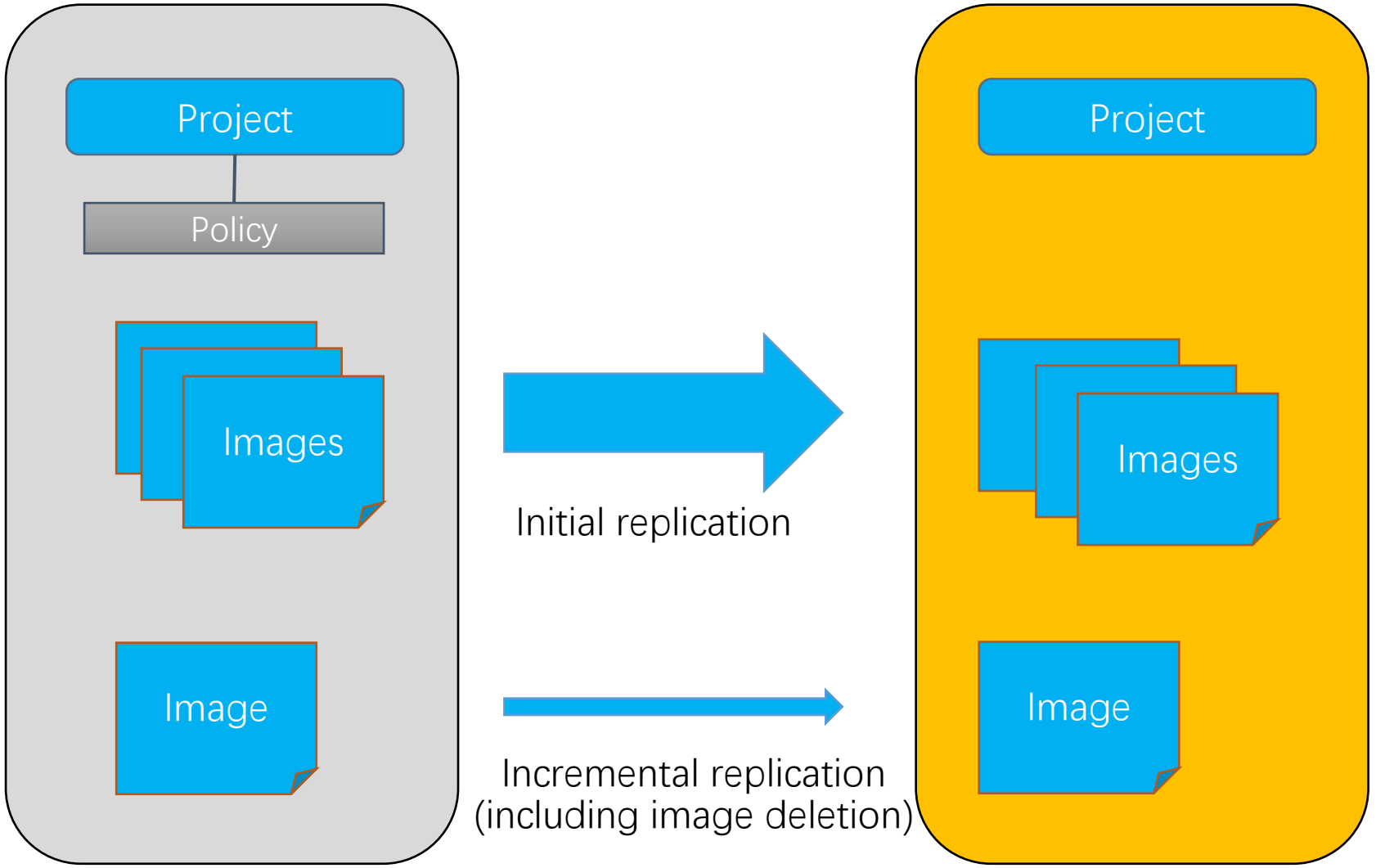


Image Replication Use Case(1)

- Image distribution for large cluster
- Load balancing

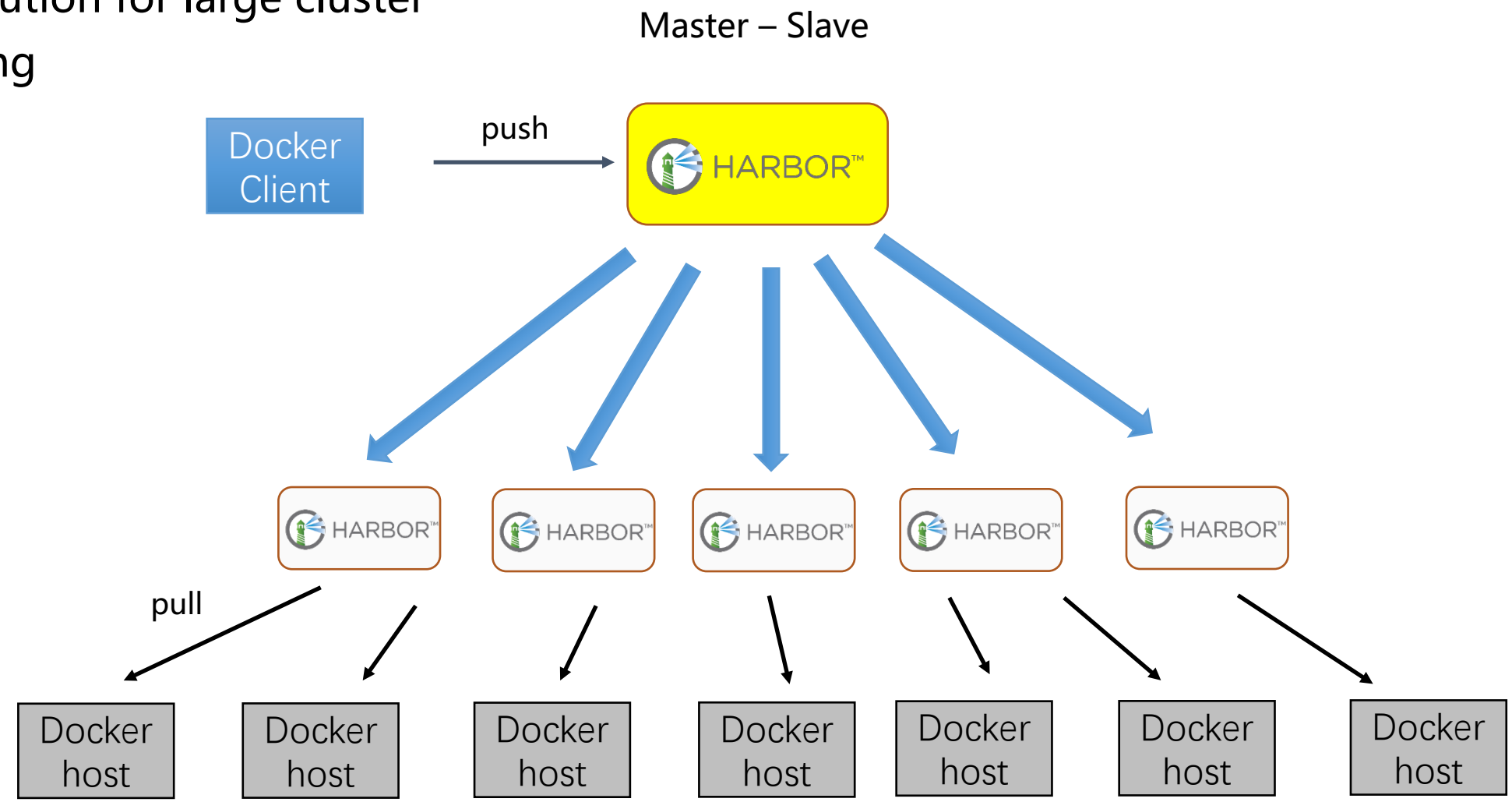
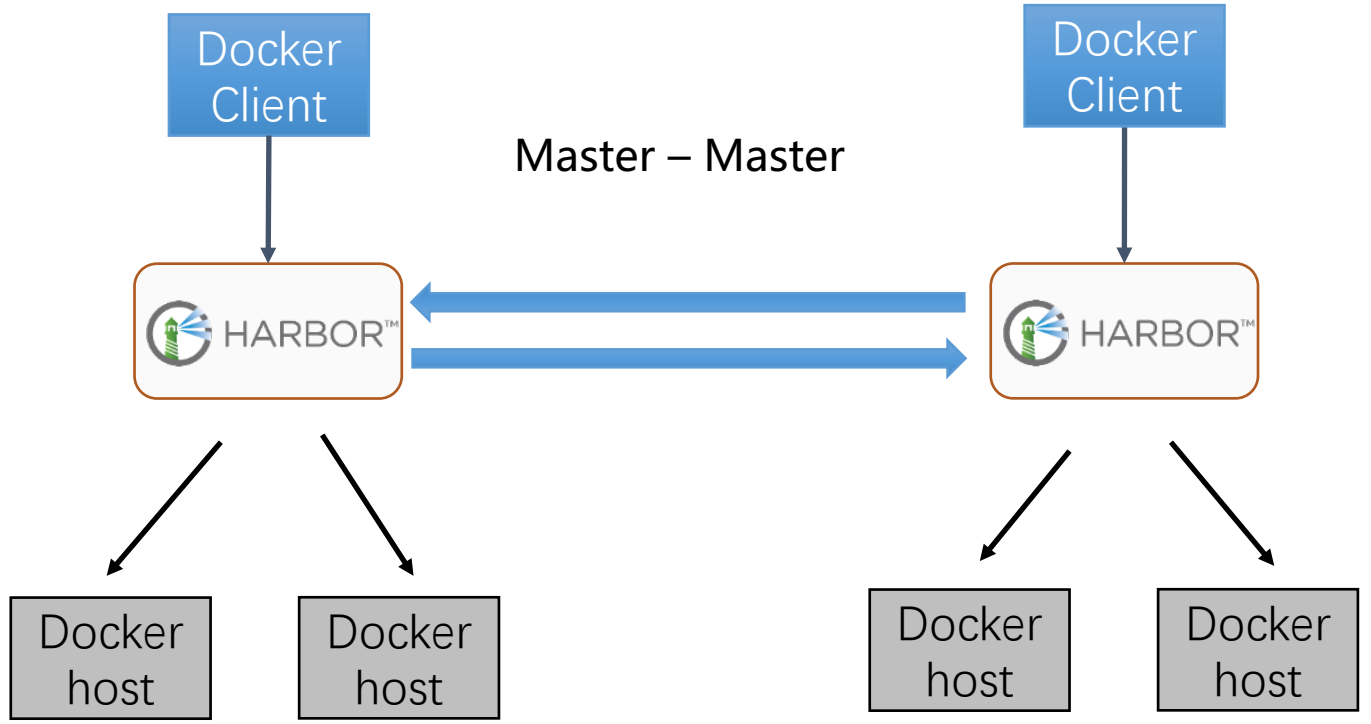
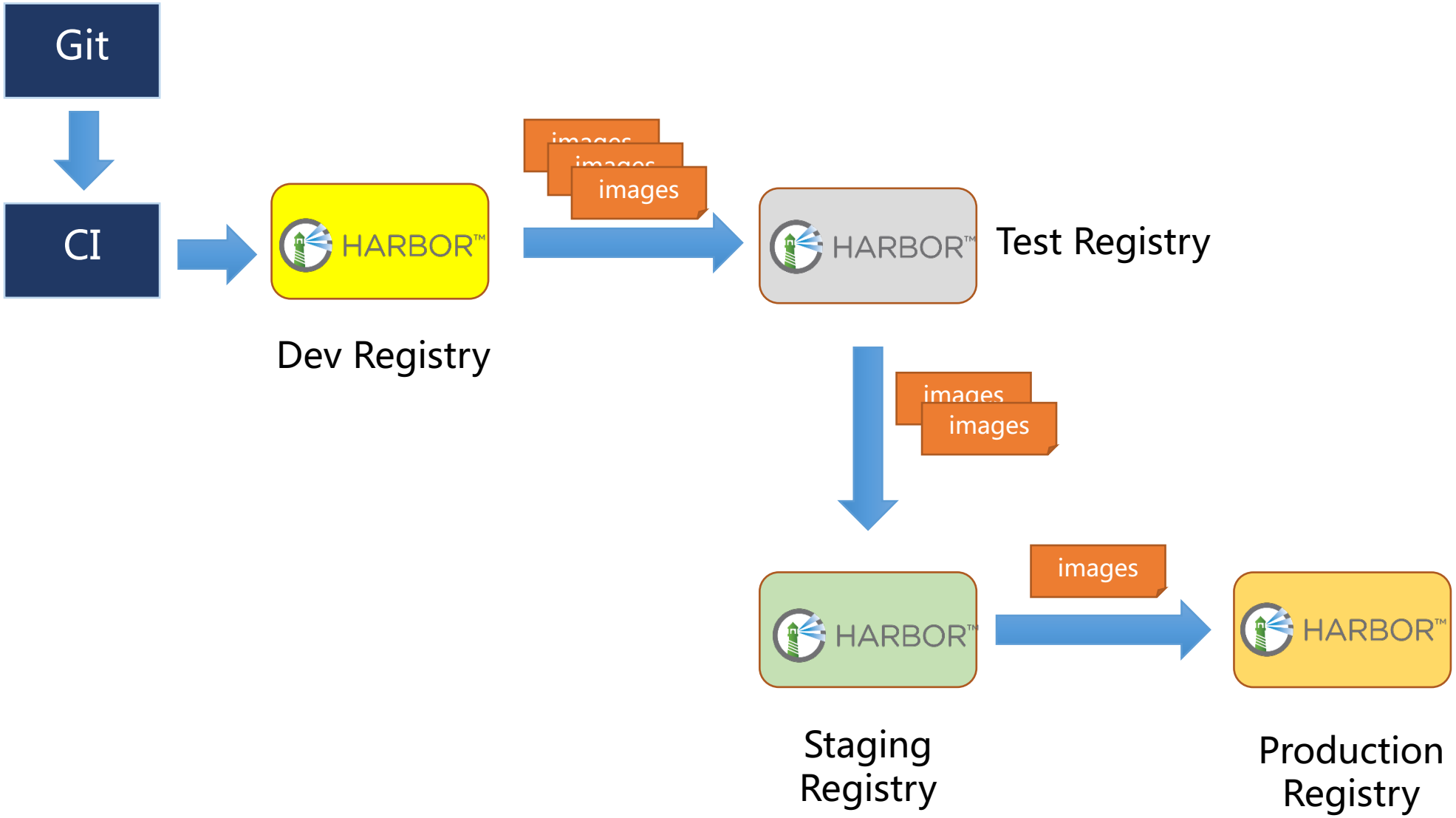


Image Replication Use Case(2)

- Remote image synchronization
  - Geographically distributed teams
  - On prem to public cloud
- Back up



Shipping (Publishing) Images via Replication



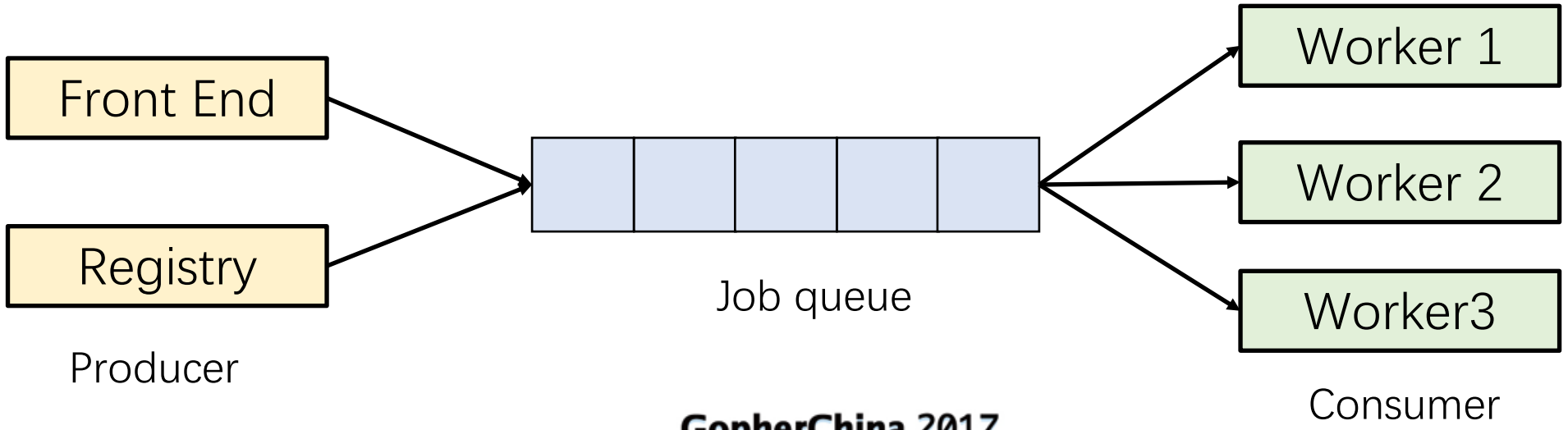
HARBOR™

Requirements of Image Replication

- Asynchronous replication (background job)
- Little impact to registry service (throttle)
- Reliable and auto retry failed operations (recovery)
- Manual intervention (admin interaction)

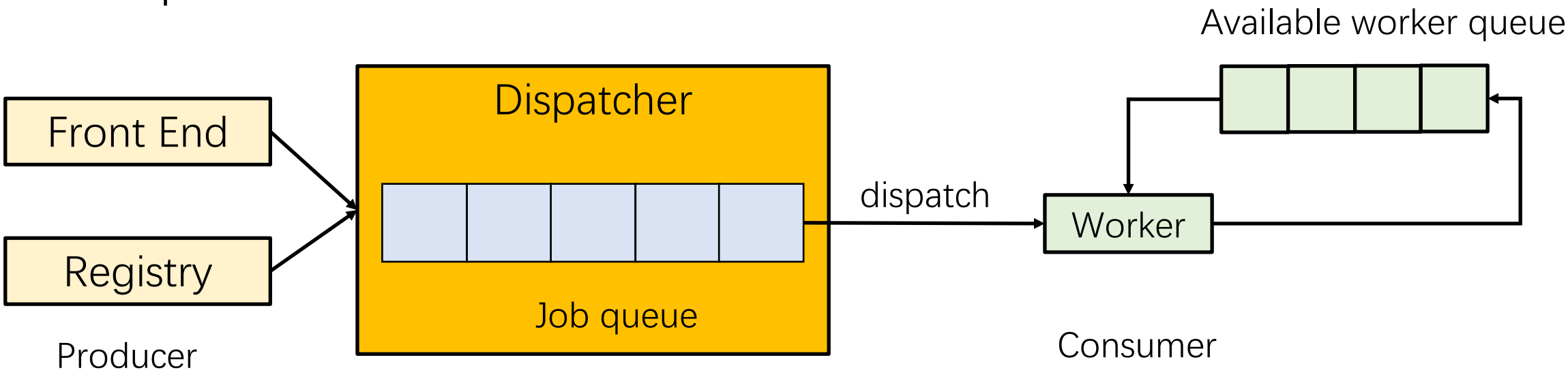
Producer and Consumer Pattern

- Front end (UI) or registry generates replication jobs (producer)
- Backend workers handle replication (consumer)
- Potential issues
  - Producers need to sleep or wait when buffer is full
  - Sleep or wait is not suitable for front end / registry



Modified Producer and Consumer Pattern

- Non blocking for producers
- Dispatcher queues jobs
- Dispatcher distributes jobs to available workers
- Workers added back to available worker queue after jobs are completed



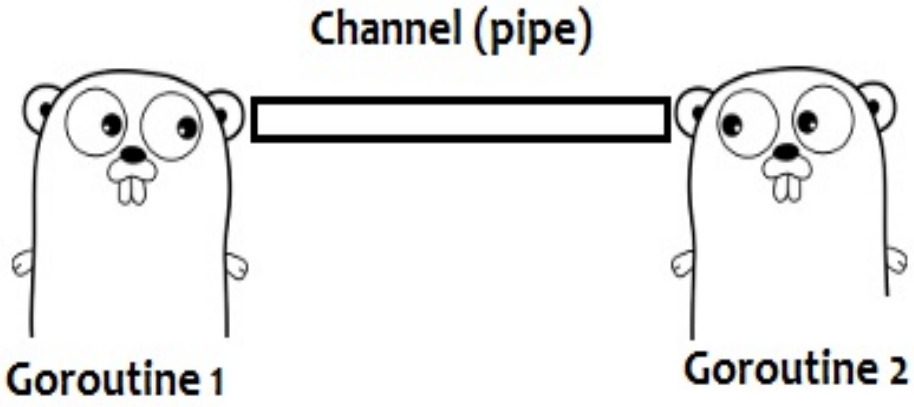
HARBOR™

Goroutine as Lightweight Thread

- Simple syntax
  - `go f(x,y,z)`
- Concurrency (asynchronousness)
- Shared the same address space
- Non blocking for main flow
- Ideal for background replication

Channel for Communication between Threads

- Syntax
  - No buffering: `make(chan Type)`
  - With buffering: `make(chan Type, capacity)`
  - Send: `ch <- v`
  - Receive: `v:= <- ch`
- Used to block or unblock threads
  - Dispatcher thread (producer)
  - Worker thread (consumer)
- Also used for stopping a job



Worker Pool

- Predefine a pool of available workers (default:3, not to overwhelm frontend tasks)
- A list of workers and a channel for dispatching job

```
25  type workerPool struct {
26          workerChan chan *Worker
27          workerList []*Worker
28  }
29
30  // WorkerPool is a set of workers each worker is associate to a statemachine for handling jobs.
31  // it consists of a channel for free workers and a list to all workers
32  var WorkerPool *workerPool
```

harbor/src/jobservice/job/workerpool.go

Worker

- A channel to receive replication job
- Another channel to receive special instruction, such as quitting

```
47  // Worker consists of a channel for job from which worker gets the next job to handle, and a pointer to a statemachine,
48  // the actual work to handle the job is done via state machine.
49  type Worker struct {
50          ID      int
51          RepJobs chan int64
52          SM      *SM
53          quit    chan bool
54  }
```

harbor/src/jobservice/job/workerpool.go

Workers Wait for Replication Job

HARBOR™

- Channel w.RepJobs blocked until a job is dispatched

```
56  // Start is a loop worker gets id from its channel and handle it.
57  func (w *Worker) Start() {
58          go func() {
59                  for {
60                          WorkerPool.workerChan <- w
61                          select {
62                          case jobID := <-w.RepJobs:
63                                  log.Debugf("worker: %d, will handle job: %d", w.ID, jobID)
64                                  w.handleRepJob(jobID)
65                          case q := <-w.quit:
66                                  if q {
67                                          log.Debugf("worker: %d, will stop.", w.ID)
68                                          return
69                                  }
70                          }
71                  }
72          }()
73  }
```

Dispatcher

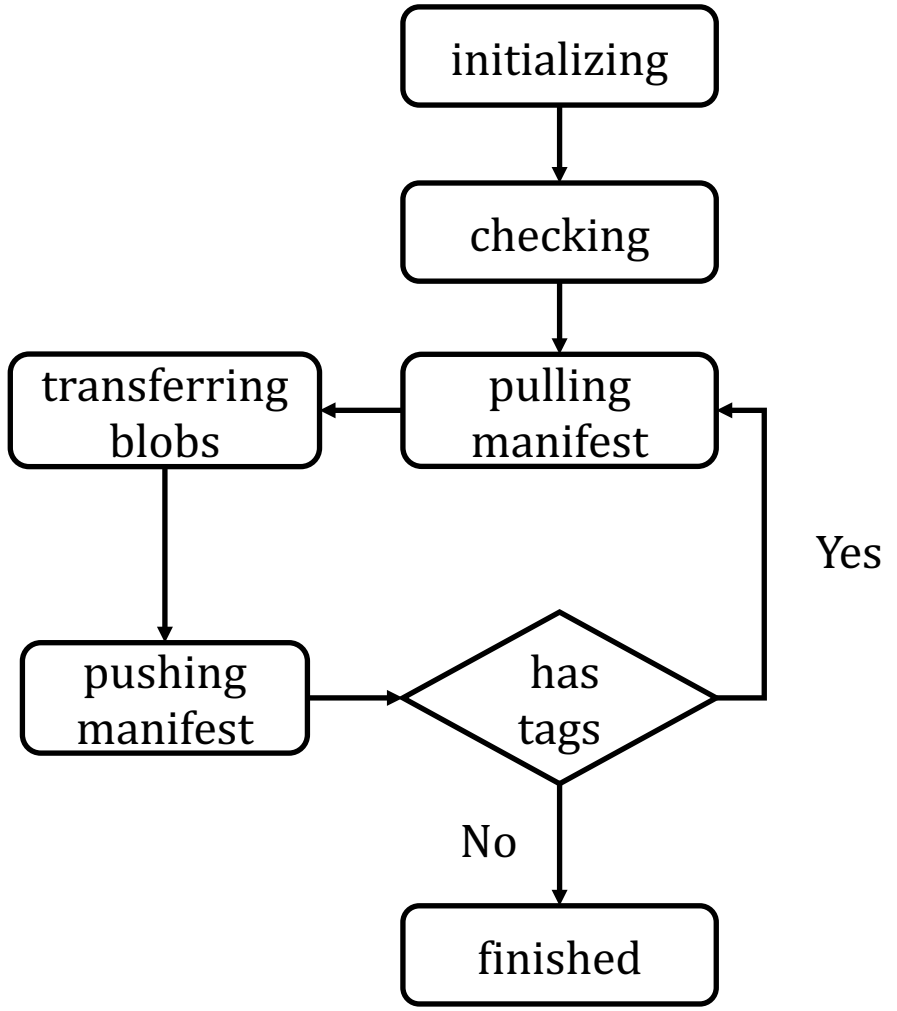
HARBOR™

- Receives job and distributes to available worker
- Channel WorkerPool.workerChan is blocked if no worker is available

```
func Dispatch() {
        for {
                job := <-jobQueue
                go func(jobID int64) {
                        log.Debugf("Trying to dispatch job: %d", jobID)
                        worker := <-WorkerPool.workerChan
                        worker.RepJobs <- jobID
                }(job)
        }
}
```

Replication Job

- Replicating an image itself seems not THAT hard
- However ….



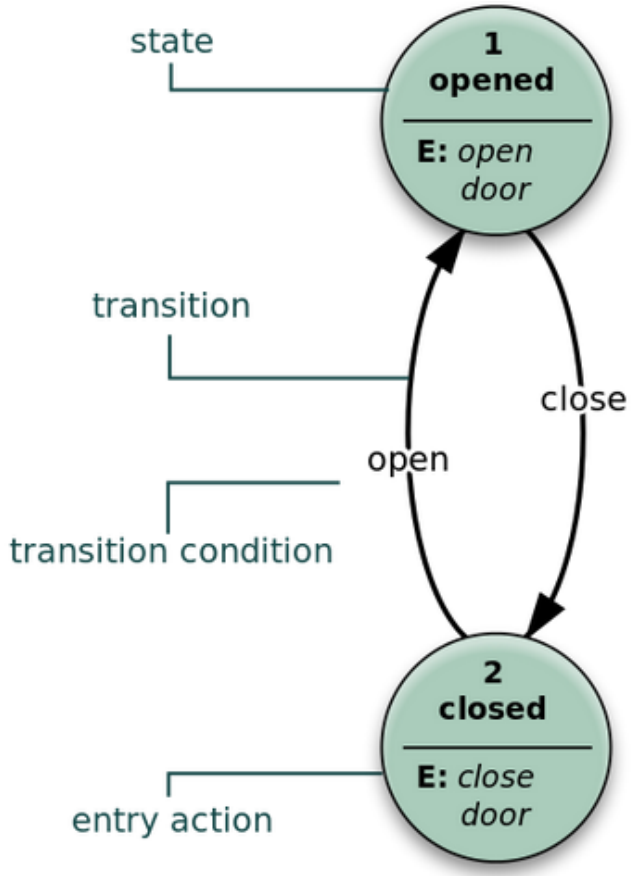
The Complexity of Replication Job

- The complexity adds up in these aspects:
  - Monitoring (logging)
  - Error handling
  - Arbitrary exit
  - Graceful retry
  - Auto recovery
- Really messy in control flow

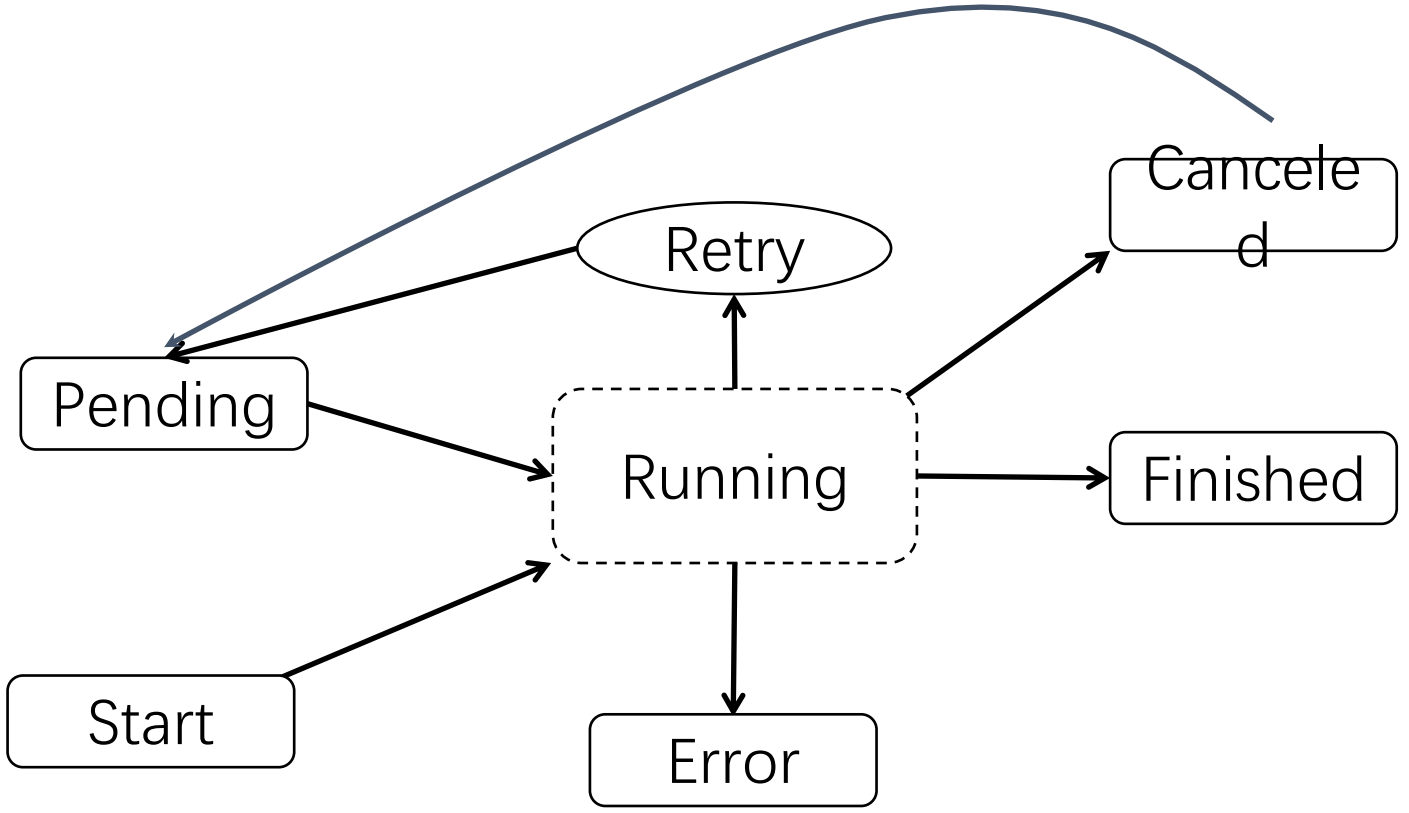
HARBOR™

State Machine Comes in to Rescue

- Simple is beautiful!
- A divide-and-conquer mindset to simplify logic
- Sort out a limited numbers of states
- Define conditions of transition
- Focus on handling logic of each state
- Separate concerns like errors and retries



Replication Job in State Diagram



State Machine

- Each worker has a state machine to execute core logic

```
func (sm *SM) Start(s string) {
        n, err := sm.EnterState(s)
        for len(n) > 0 && err == nil {
                if d := sm.getDesiredState(); len(d) > 0 {
                        n = d
                        sm.setDesiredState("")
                        continue
                }
                if n == models.JobContinue && len(sm.Transitions[sm.CurrentState]) == 1 {
                        for n = range sm.Transitions[sm.CurrentState] {
                                break
                        }
                        continue
                }
                ...
                n, err = sm.EnterState(n)
        }
        if err != nil {
                sm.EnterState(models.JobError)
        }
}
```

harbor/src/jobservice/job/statemachine.go

Configure State Machine

- Building state diagram by adding states and transition handlers

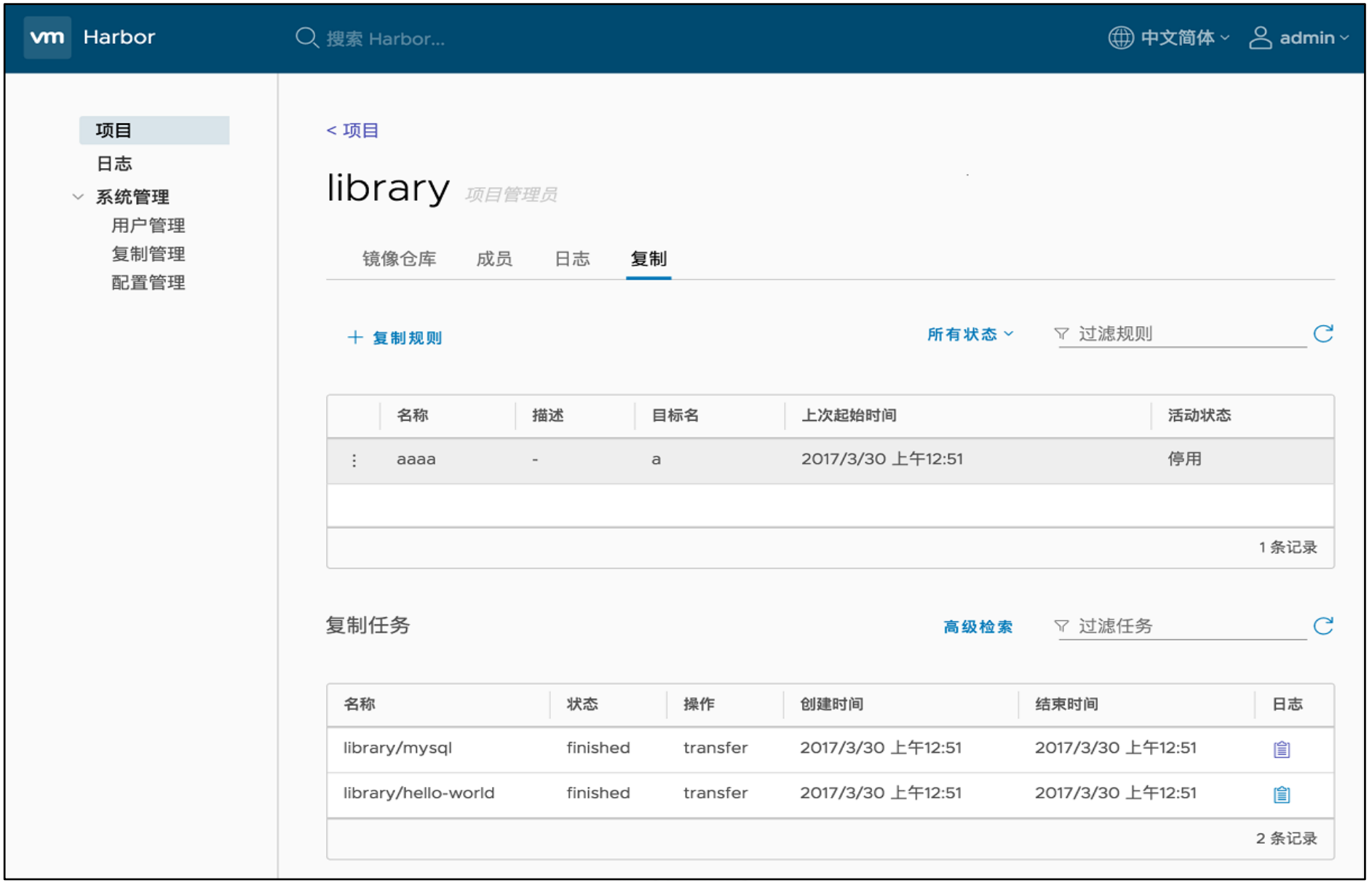
```
func addingTransferTransition(sm *SM) {
        base := replication.InitBaseHandler(sm.Parms.Repository, sm.Parms.LocalRegURL,
                config.JobserviceSecret(),sm.Parms.TargetURL, sm.Parms.TargetUsername,
                sm.Parms.TargetPassword,sm.Parms.Insecure, sm.Parms.Tags, sm.Logger)

        sm.AddTransition(models.JobRunning, replication.StateInitialize,
                        &replication.Initializer{BaseHandler: base})
        sm.AddTransition(replication.StateInitialize, replication.StateCheck,
                        &replication.Checker{BaseHandler: base})
        sm.AddTransition(replication.StateCheck, replication.StatePullManifest,
                        &replication.ManifestPuller{BaseHandler: base})
        sm.AddTransition(replication.StatePullManifest, replication.StateTransferBlob,
                        &replication.BlobTransfer{BaseHandler: base})
        sm.AddTransition(replication.StatePullManifest, models.JobFinished,
                        &StatusUpdater{sm.JobID, models.JobFinished})
        sm.AddTransition(replication.StateTransferBlob, replication.StatePushManifest,
                        &replication.ManifestPusher{BaseHandler: base})
        sm.AddTransition(replication.StatePushManifest, replication.StatePullManifest,
                        &replication.ManifestPuller{BaseHandler: base})
}
```

harbor/src/jobservice/job/statemachine.go

Demo

HARBOR™


Harbor
搜索 Harbor...
中文简体
admin
项目
日志
系统管理
用户管理
复制管理
配置管理
< 项目
library
项目管理员
镜像仓库
成员
日志
复制
+ 复制规则
所有状态
过滤规则
名称
描述
目标名
上次起始时间
活动状态
aaaa
-
a
2017/3/30 上午12:51
停用
1 条记录
复制任务
高级检索
过滤任务
名称
状态
操作
创建时间
结束时间
日志
library/mysql
finished
transfer
2017/3/30 上午12:51
2017/3/30 上午12:51
library/hello-world
finished
transfer
2017/3/30 上午12:51
2017/3/30 上午12:51
2 条记录

HARBOR™

Results

- Small code base
- Straightforward logic
- Reliable operations
- Monitoring and logging
- Container image replication is very welcome by users

HARBOR™

Summary

- Goroutine is great for concurrency programming.
- Channel used for coordination between goroutines.
- State machine pattern simplifies the implementation of control flow.

- Try it, love it and contribute to it!

https://github.com/vmware/harbor