GopherChina2018

Go与虚拟化容器runV/Kata

王旭 (@gnawux)

HYPER.SH

关于王旭 & Hyper

- 王旭:
  - Hyper.sh Cofounder & CTO;
  - Kata Containers arch committee
- Hyper.sh 的一些开源项目(Go)
  - Hyperd + runV + hyperstart
  - Hypernetes & frakti
- Hyper.sh 主要参与/贡献的一些开源项目
  - Kata Containers
  - Kubernetes

- 语言背景
  - Golang: hyper.sh 各个项目
  - Scala/akka: 某后端服务
  - C/C++: 某云存储服务
  - Java: Hadoop HDFS
  - Python, Ruby, Shell, etc.

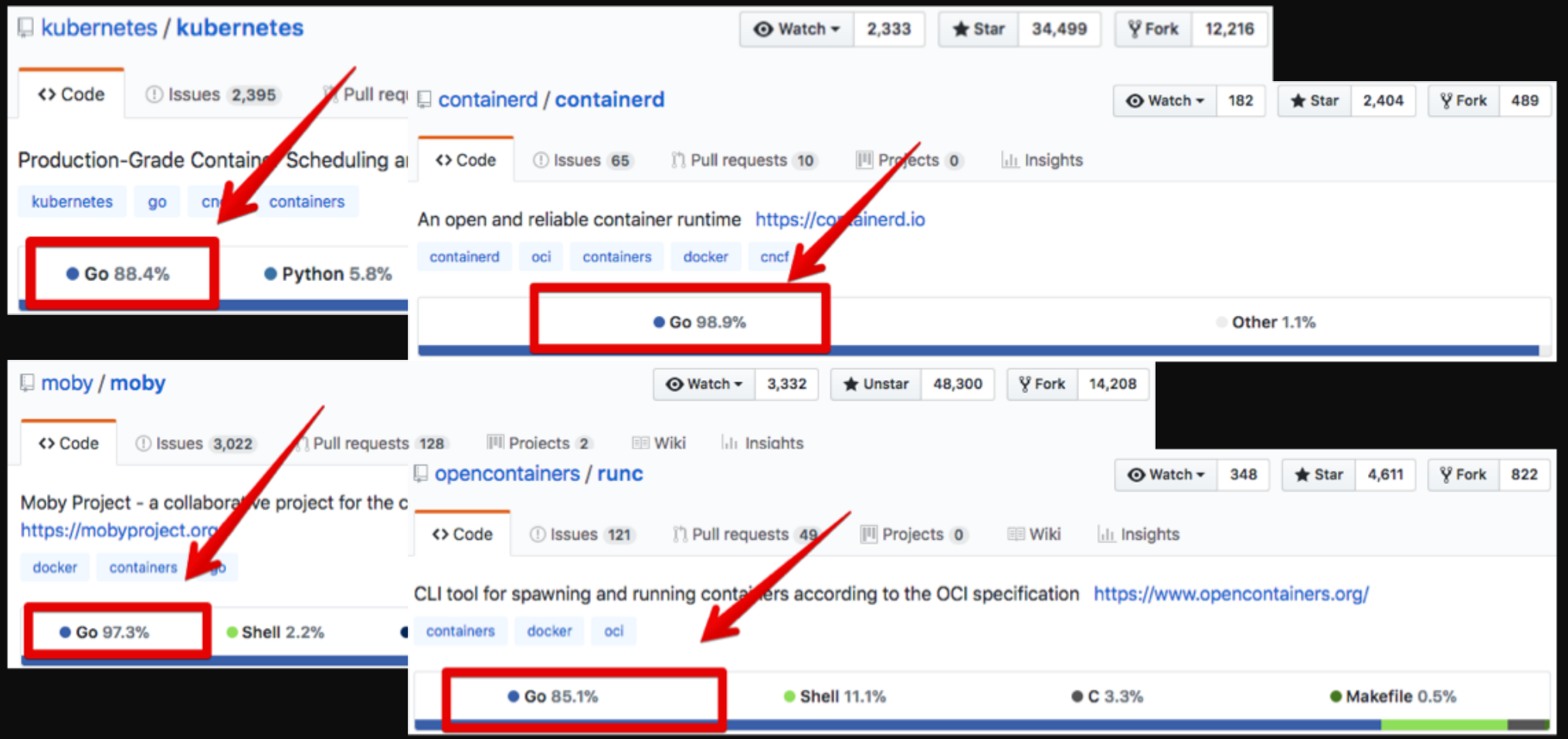
GopherChina2018

什么是虚拟化容器
为什么选择用 Go 开发

Secure as VM, Fast as Container

- runV/kata 是兼容 OCI 的容器执行引擎
  - runV 由 Hyper.sh 开发，始于2015年，并有 Huawei, ARM, IBM 等公司参与
  - 2017年12月，Hyper 和 Intel 宣布，runV 与 Clear Containers 合并成为 Kata Containers 项目，并由 OpenStack Foundation 管理
- runV / kata 使用 hypervisor 提供容器间的隔离，并可以以百毫秒级别的时间启动容器
  - Secure as VM, Fast as Container
- Hyper.sh 的团队分布在国内多个城市，基于 github 协同工作

Go: 容器领域的事实标准



优秀的系统编程能力

例: 使用 Unix Domain Socket 的 socket control message

```
46      scm := syscall.UnixRights(fd)
47      glog.V(1).Info("send net to qemu at %d", fd)
48      commands = append(commands, &QmpCommand{
49          Execute: "getfd",
50          Arguments: map[string]interface{}{
51              "fdname": "fd" + guest.Device,
52          },
53          Scm: scm,
54      }, &QmpCommand{
55          Execute: "netdev_add",
56          Arguments: map[string]interface{}{
57              "type": "tap", "id": guest.Device, "fd": "fd" + guest.Device,
58          },
59      })
```

对C语言开发者友好

- 语法语义：比较接近
- 功能性：都提供操作系统底层功能
- 高级语言特性：
  - 相对C语言，Go提供的高级特性比较少
  - 在语言层面提供了 channel, goroutine 这类非常实用功能
- 标准库
  - Go语言的标准库提供了实用的功能和工业级的效率
- 互操作
  - Go语言可以方便地调用已有的 C 库
  - Go语言可以方便地使用gdb

GopherChina2018

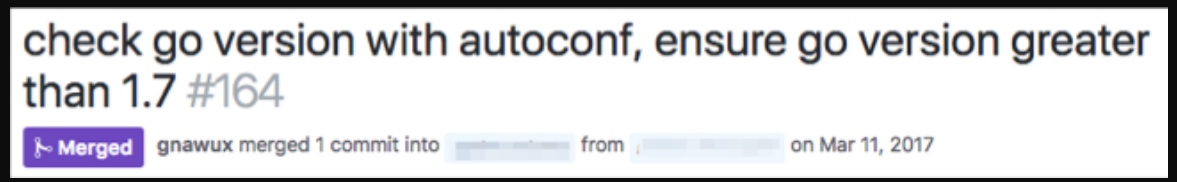
runV开发中的一些Go语言实践

说明与免责声明

- 只谈实践，不论好坏
  - 不争论 Go 语言是不是一种好语言
  - 不争论在讨论的是不是一个好特性
- 以下实践从整体设计到具体细节
- 欢迎指出谬误

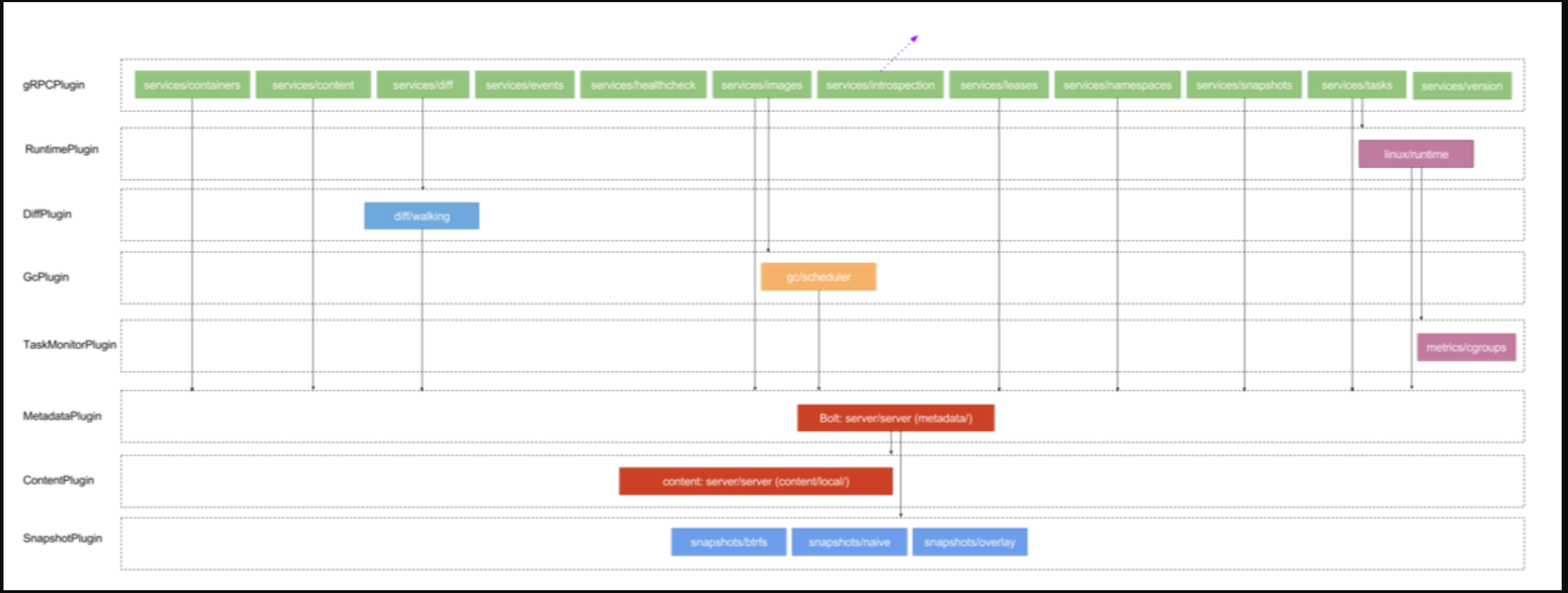
总是使用最新的Go编译器

- 案例: 2017年3月
- 现象: 系统部分状态不一致
- 历时一周的调查与解决
  - 数据库操作发现原子性问题，加锁解决
    - 但引出数据库性能异常问题
  - 通过数据库慢查询发现有索引问题，添加索引后问题仍然没有完全解决
    - CPU占用过高是根本原因
  - 通过 Perf 发现系统中性能瓶颈，之后进行修复
    - 修复后发现CPU占用显著低于问题发生前……
  - 最终，发现是第一次修复之后，使用了 go 1.6.3 编译，换到 1.7 之后，本来无需更多优化
- 结果，加入 configure 配置，拒绝低版本编译器



适度使用接口

例: containerd



注意资源的释放

例: 某数据库异常

cell monitor: close db session after start/stop hook finish

Signed-off-by:

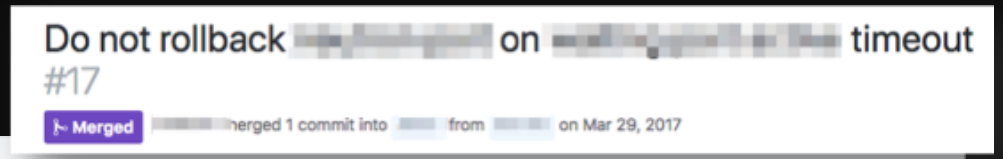
committed Verified commit a16

2 pkg/tracker/cell_monitor.go View

```
@@ -304,6 +304,7 @@ func (m *CellMonitor) setStarted(cont *ContainerStateInfo, timestamp *time.Time)
304 304                 if m.stopHook != nil {
305 305                         go func() {
306 306                                 s := m.stor.Copy()
    307 +                               defer s.Close()
307 308                                 m.stopHook.OnContainerStart(s, m.cell, cont.ID, cont.Tenant)
308 309                         }()
309 310                 }
@@ -342,6 +343,7 @@ func (m *CellMonitor) setStopped(cont *ContainerStateInfo, timestamp *time.Time,
342 343                 if m.stopHook != nil {
343 344                         go func() {
344 345                                 s := m.stor.Copy()
    346 +                               defer s.Close()
345 347                                 m.stopHook.OnContainerStopped(s, m.cell, cont.ID, cont.Tenant, exitCode)
346 348                         }()
347 349                 }
```

显式地忽略错误

- 在这种情况下




```
if err = foo(); err != nil {
    glog.Warningf("some bad thing occured, but not a fatal error. We don't want to break here: %v", err)
    //<HERE>
}
```

- 如果不打算异常退出，应该直接设置 err = nil，因为我们常常可能已经在 defer 里这样做了

```
defer func() {
    if err != nil {
        rollback()
    }
} ()
```

遍历 slice 时区分用值还是索引

```go
package main

import (
        "fmt"
)

func main() {
        arr := []struct{v int}{{1},{2},{3}}

        for _, e := range arr {
                e.v++
        }
        fmt.Printf("%v\n", arr)

        for idx := range arr {
                arr[idx].v++
        }
        fmt.Printf("%v\n", arr)
}
```

```
[{1} {2} {3}]
[{2} {3} {4}]

Program exited.
```

https://play.golang.org/p/sSpjc-HV85

小心 append slice

```go
package main

import (
        "fmt"
)

func main() {
        s0 := []int{1, 2, 3, 4}
        s1 := s0[:2]
        s2 := append(s1, 5)
        s3 := append(s1, 6, 7)
        s4 := append(s1, 8, 9, 0)
        fmt.Println(s0)
        fmt.Println(s1)
        fmt.Println(s2)
        fmt.Println(s3)
        fmt.Println(s4)
}
```

```
[1 2 6 7]
[1 2]
[1 2 6]
[1 2 6 7]
[1 2 8 9 0]

Program exited.
```

https://play.golang.org/p/Kt4tKD3Jc6

操作空 channel 会死锁而非 panic

```go
package main

import "fmt"

func main() {
        var c chan string
        fmt.Println(<-c) // deadlocks
}
```

https://play.golang.org/p/xSva-xjoVHN

```go
package main

func main() {
        var c chan string
        c <- "let's get started"
}
```

https://play.golang.org/p/t5z1Jun9xp5

这两个情况都会死锁，而非 pinic，所以，如果一个 channel 可能为空，那么，即使有 panic handler，也必须要在访问前检查 channel 是否为空

GopherChina2018

对 Go 未来的一些期待

泛型？

```
apps/v1/types.go:type StatefulSetCondition struct {
apps/v1/types.go:type DeploymentCondition struct {
apps/v1/types.go:type DaemonSetCondition struct {
apps/v1/types.go:type ReplicaSetCondition struct {
apps/v1beta2/types.go:type StatefulSetCondition struct {
apps/v1beta2/types.go:type DeploymentCondition struct {
apps/v1beta2/types.go:type DaemonSetCondition struct
apps/v1beta2/types.go:type ReplicaSetCondition struct {
apps/v1beta1/types.go:type StatefulSetCondition struct {
apps/v1beta1/types.go:type DeploymentCondition struct {
autoscaling/v1/types.go:type HorizontalPodAutoscalerCondition
autoscaling/v2beta1/types.go:type HorizontalPodAutoscalerCondition
batch/v1/types.go:type JobCondition struct {
certificates/v1beta1/types.go:type CertificateSigningRequestCondition
core/v1/types.go:type PersistentVolumeClaimCondition struct {
core/v1/types.go:type PodCondition struct {
core/v1/types.go:type ReplicationControllerCondition struct {
core/v1/types.go:type NodeCondition struct {
core/v1/types.go:type ComponentCondition struct {
extensions/v1beta1/types.go:type DeploymentCondition struct {
extensions/v1beta1/types.go:type DaemonSetCondition struct {
extensions/v1beta1/types.go:type ReplicaSetCondition struct {
➜ api git:(release-1.9) grep -r "type .*Condition struct" *|wc -l
      22
```

```
// StatefulSetCondition describes the state of
type StatefulSetCondition struct {
    // Type of statefulset condition.
    Type StatefulSetConditionType `json:"type"
    // Status of the condition, one of True, F
    Status v1.ConditionStatus `json:"status" p
    // Last time the condition transitioned fr
    // +optional
    LastTransitionTime metav1.Time `json:"last
    // The reason for the condition's last tra
    // +optional
    Reason string `json:"reason,omitempty" pro
    // A human readable message indicating det
    // +optional
    Message string `json:"message,omitempty" p
}
```

```
// DeploymentCondition describes the state of a de
type DeploymentCondition struct {
    // Type of deployment condition.
    Type DeploymentConditionType `json:"type" prot
    // Status of the condition, one of True, False
    Status v1.ConditionStatus `json:"status" proto
    // The last time this condition was updated.
    LastUpdateTime metav1.Time `json:"lastUpdateTi
    // Last time the condition transitioned from c
    LastTransitionTime metav1.Time `json:"lastTran
    // The reason for the condition's last transit
    Reason string `json:"reason,omitempty" protobu
    // A human readable message indicating details
    Message string `json:"message,omitempty" proto
}
```

更好用的 go dep

- Go 是静态编译语言（尽管已经有了 plugin）
- Go 项目以源码形式引用项目
- Go 1.5 开始，规范了 vendor 的使用方式
- 很多工具可以用来管理 vendor
  - Godep, govendor, glide, dep…
- 然而当依赖多了的时候，还是……
- 官方 dep 工具已经开始越来越广泛的被应用了

GopherChina2018

Thank You

Any Questions?