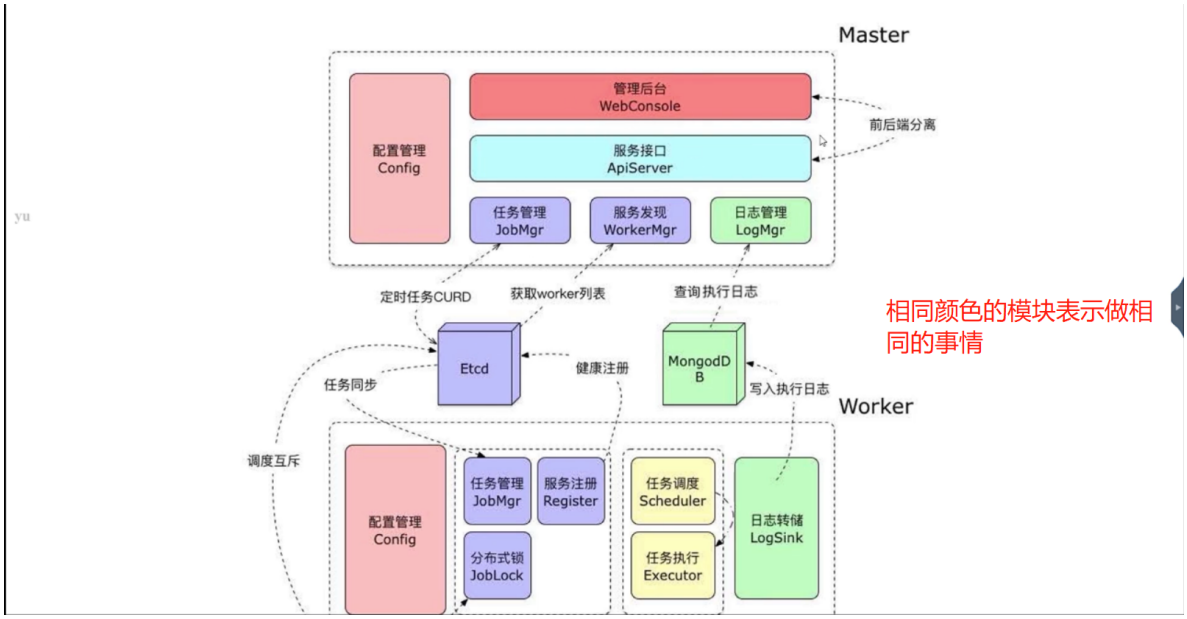
传统方案--crontab

- 缺点
  - 配置任务时,需要SSh登录脚本服务器进行操作
  - 服务器宕机,任务将终止调度,需要人工迁移
  - 排查问题低效,无法方便的查看任务状态与错误输出

分布式任务调度

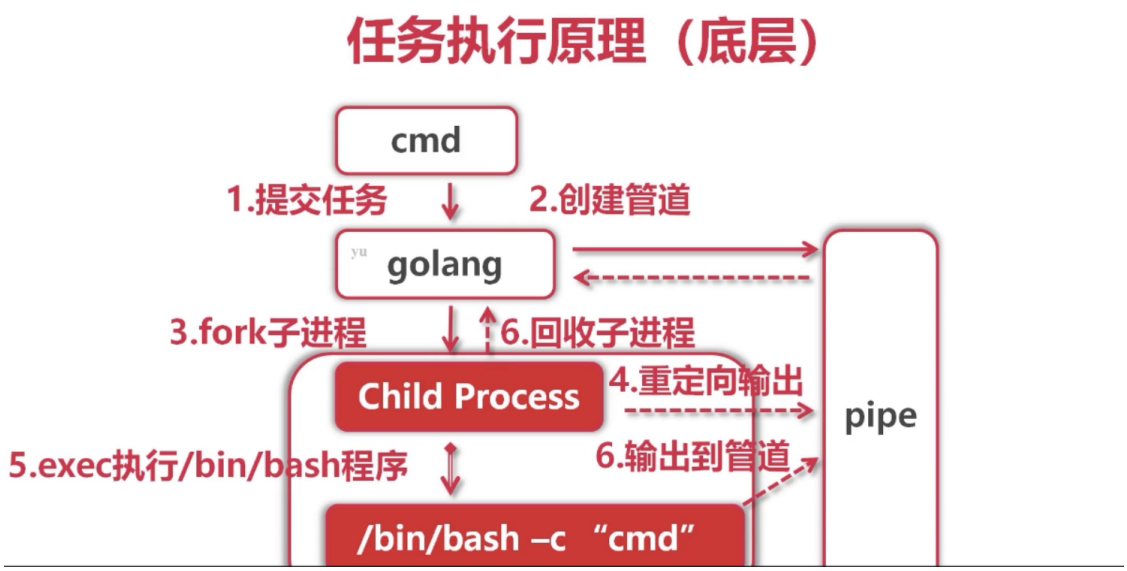
- 优点
  - 可视化Web后台,方便进行任务管理
  - 分布式架构、集群化调度,不存在单点故障
  - 追踪任务执行状态,采集任务输出,可视化log查看

架构图



go执行shell命令

- **1、执行程序：**/usr/bin/python start.py
- **2、调用命令：** cat nginx.log | grep "2022"
  - bash模式
    - 交互模式：ls -l
    - 非交互模式：/bin/bash -c "ls -l" ------ 我们使用这个
- 任务执行原理（底层）了解即可



实际我们在golang代码中调用Linux命令

1、普通调用

```
package main

import (
    "fmt"
    "os/exec"
)

var (
    output []byte
    err    error
)

func main() {
    // 要执行的命令
    cmd := exec.Command("bash.exe", "-c", "echo 111")

    // CombinedOutput-->捕获异常跟命令输出的内容
    if output, err = cmd.CombinedOutput(); err != nil {
        fmt.Println("error is :", err)
        return
    }

    // 打印输出结果
    fmt.Println(string(output))
}
```

2、结合协程调用，可控制中断调用

```
package main

import (
    "context"
    "fmt"
    "os/exec"
    "time"
```

```
)

// 接收子协程的数据,协程之间用chan通信
type result struct {
    output []byte
    err    error
}

func main() {
    // 执行一个cmd，让他在一个携程里面执行2s，
    // 1s的时候  杀死cmd
    var (
        ctx        context.Context
        cancelFunc context.CancelFunc
        cmd        *exec.Cmd
        resultChan chan *result
        res        *result
    )

    // 创建一个结果队列
    resultChan = make(chan *result, 1000)

    /*
        1. WithCancel()函数接受一个 Context 并返回其子Context和取消函数cancel

        2. 新创建协程中传入子Context做参数，且需监控子Context的Done通道，若收到消息，则退出

        3. 需要新协程结束时，在外面调用 cancel 函数，即会往子Context的Done通道发送消息

        4. 注意：当 父Context的 Done() 关闭的时候，子 ctx 的 Done() 也会被关闭
    */
    ctx, cancelFunc = context.WithCancel(context.TODO())

    // 起一个协程
    go func() {
        var (
            output []byte
            err    error
        )
        // 生成命令
        cmd = exec.CommandContext(ctx, "C:\\Windows\\System32\\bash.exe", "-c", "sleep 3;echo hello;")

        // 执行命令cmd.CombinedOutput(),且捕获输出
        output, err = cmd.CombinedOutput()

        // 用chan跟主携程通信,把任务输出结果传给main协程
        resultChan <- &result{
            err:    err,
            output: output,
        }
    }()

    // Sleep 1s
    time.Sleep(time.Second * 1)

    // 取消上下文,取消子进程,子进程就会被干掉
```

```
    cancelFunc()

    // 从子协程中取出数据
    res = <-resultChan

    // 打印子协程中取出数据
    fmt.Println(res.err)
    fmt.Println(string(res.output))

}
```