گزارش کارآموزی
شرکت ایده گزین روماک (اسنپ)

الهه داستان

تابستان ۱۴۰۰

فصل ۱

گزارش‌های هفتگی

۱.۱ هفته اول - ۲۲ام تیر ماه

هفته‌ی اول کارآموزی بیشتر صرف شناخت ساختار شرکت و گرفتن دسترسی‌ها و خواندن کدهای قبلی می‌شود.

ساختار شرکت اسنپ از ساختار اسپاتیفای[۱] الهام گرفته شده و حالت شطرنجی دارد. بدین صورت که هر ونچر (منظور از ونچر بیزینس‌های مختلف می‌باشد، به طور مثال اسنپ کب[۲] و اسنپ باکس[۳] و اسنپ دکتر[۴] سه ونچر متفاوت اند) یک مدیر فنی ارشد دارد سپس هر ونچر به چندین چپتر[۵] تقسیم می‌شود برای مثال در ونچر اسنپ کب که من در آن مشغول هستم دو چپتر برای بک‌اند داریم که نام یک چپتر، آلفا[۶] است که عمده سرویس‌هایشان با زبان PHP توسعه یافته است و روی کد قدیمی و اولیه‌ی اسنپ کار می‌کنند. چپتر دیگر براوو[۷] است که با زبان گولنگ و روی میکروسرویس‌های جدید شرکت که از کد قدیمی جدا شده‌اند، کار می‌کند.

زبان گولنگ کامپایل شده، سرعت خوبی داشته و یادگیری آن ساده است. همگی این دلایل باعث شده‌اند که سرویس‌های جدید شرکت با این زبان توسعه پیدا کنند.

هر چپتر یک مدیر دارد که وظیفه‌ی سامان‌دهی به آن چپتر هم از نظر مدیریت افراد و هم فنی را دارد. هر چپتر به ورتیکال‌های مختلف تفسیم می‌شود که هر ورتیکال مسئول توسعه‌ی میکروسرویس‌های مشخصی است. هر ورتیکال نیز مدیر خود را دارد، در چپتر ما ورتیکال‌های مختلفی مانند شرد سرویسز، دیسپچیگ و ... وجود دارد.

[1] Spotify
[2] Snapp Cab
[3] Snapp Box
[4] Snapp Doctor
[5] Chapter
[6] Alpha
[7] Bravo

ورتیکالی که من در آن مشغول هستم یوزر نام دارد و تنها ورتیکالی است که به داده‌های خام کاربران دسترسی دارد و وظیفه ی ارتباط سرویس‌های داخلی با سرویس‌های خارجی دارد مثلا ساخت اکانت و لاگین کاربران اسنپ در دست این ورتیکال است یا ثبت نام دیجیتال راننده‌ها توسط این ورتیکال صورت میگیرد که اهمیت آن در دوران کرونا دو چندان شده است. همچنین پنهان کردن شماره های کاربران هنگام تماس با راننده و چندین سرویس دیگر برعهده این ورتیکال است.

در شرکت با توجه به نیاز ورتیکال‌ها هر ورتیکال می‌تواند تعداد مختلفی نیرو در کنار نیروهای بک‌اند داشته باشد. در ورتیکال یوزر به علت حساسیت و تنوع سرویس‌ها در کنار نیروهای بک‌اند، نیروهای فرانت‌اند، اندروید، تست و دواپس[8] همگی حضور دارند. با حضور نیروهای مدیریت محصول و اسکرام مستر[9] بزرگ‌ترین ورتیکال شرکت را تشکیل می‌دهد.

در شرکت یک نربان پیشرفت وجود دارد که هنگام ورود هر کس سطح او به او گفته می‌شود و هر ۶ ماه با توجه به عملکرد فرد که توسط مدیر ورتیکال، مدیر چپتر و نظر سایر هم‌تیمی‌های او سنجیده می‌شود می‌تواند سطح وی ارتقا پیدا کند.

در شرکت از متدلوژی اسکرام استفاده می‌شود. هر فصل در سال یک کوآرتر محسوب می‌شود که در ابتدای آن شرکت اهداف کلی را تعیین می‌کند، که این اهداف با نظارت مستقیم مدیر فنی ارشد ونچر مشخص می‌شود. این اهداف در اختیار ورتیکال‌ها قرار می‌گیرد و هر ورتیکال دو هفته فرصت دارد تا اهداف خود را مشخص کند به طوری که در راستای اهداف شرکت قرار بگیرد. در ابتدای هر کوآرتر جلسات زیادی برای طراحی فنی پروژه‌ها، تخمین زمانی هر یک و ...می‌شود. هر کوآرتر به ۶ اسپیرینت[10] دو هفته‌ای تقسیم می‌شود و در ابتدای هر اسپیرینت نیز جلساتی برای انتخاب تسک‌ها و تخمین زمان آن ها تشکیل می‌شود. در انتهای هر اسپیرینت نموداری توسط اسکرام مستر کشیده می‌شود که نشان دهنده‌ی عملکرد اعضا و عملکرد کلی تیم است و با توجه به آن ظرفیت تیم برای اسپرینت بعدی تخمین زده می‌شود. همچنین در انتهای هر کوآرتر اسکرام مستر با تک تک اعضا جلساتی خواهد داشت تا از دغدغه‌ها و مشکلات آن‌ها مطلع شود. یک جلسه رترو[11] نیز تشکیل خواهد شد تا هر کس نقد ها و پیشنهادات خود را برای کوآرتر بعد بیان کند و همچنین هر کس باید به صورت ناشناس درباره ی هم تیمی هایش نظر بدهد.

پس از اینکه یک عضو جدید به شرکت اضافه می‌شود یک فرد از خود تیم به عنوان بادی به او معرفی می شود که وظیفه دارد او را با تیم و سرویس‌ها آشنا کند و همچنین دسترسی‌های او را برایش فراهم کند. هر برنامه‌نویس پس از اضافه شدن به تیم باید یک وی پی ان از شرکت بگیرد که تنها به وسیله‌ی آن می‌تواند به سرویس‌ها دسترسی داشته باشد. همچنین باید دسترسی به ایمیل سازمانی، گیت‌لب[12] که مخزن اصلی نگهداری کدها می‌باشد، جیرا[13] که برای مدیریت تسک‌ها می‌باشد، کانفلوئنس[14] که برای مدیریت مستندات می‌باشد و اسنپ کلاد[15] که سرویس ابری شرکت مبتنی بر Openshift می‌باشد را بگیرد. همچنین برای آشنا

[8] DevOps
[9] Scrum
[10] Sprint
[11] Retro
[12] Gitlab
[13] Jira
[14] Confluence
[15] SnappCloud

شدن فرد جدید با تیم در هفته‌ی اول سعی می‌شود جلسات غیرکاری تشکیل شود که در آن اعضا بازی‌های گروهی انجام می‌دهند و با یکدیگر آشنا می‌شوند. همچنین فرد جدید در هفته‌ی اول باید ساختار کدهای شرکت را مطالعه کند و همچنین مستندات سرویس‌های ورتیکالش را مطالعه کند تا با معماری سرویس‌های ورتیکال و شرکت آشنا شود و بداند هر کدام از سوریس‌های تیم با چه سرویس‌هایی از سایر ورتیکال‌ها در ارتباط است.

۲.۱ هفته دوم - ۲۹ تیر ماه

پس از اینکه با سرویس‌های شرکت به طور کلی آشنا شدم حال باید روی یک سرویس شروع به توسعه می‌کردم. سرویسی که برای شروع انتخاب شد سرویس ستار بود که عمل پنهان کردن شماره‌های کاربران و رانندگان را انجام می‌دهد تا امنیت بیشتری برای سفر فراهم کند. به علت قرارداد عدم افشای اطلاعات[۱۶] که با شرکت امضا شده است و حساسیت کار این ورتیکال از توضیح معماری سرویس یا کد به هر شکل و حتی ذکر نام شرکت‌های طرف قرارداد کاملا معذورم و به اجبار به توضیحات زیر بسنده می‌کنم.

سرویس پنهان کردن شماره‌های تماس در شرکت یکی از پایدارترین سرویس‌های شرکت است که برای انجام کار خود به اطلاعات راننده‌ها و مسافران نیاز دارد به همین جهت با پایگاه‌های داده‌ای متعددی سر و کار دارد. همچنین برای سرعت عمل بالا از Redis به عنوان حافظه نهان استفاده شده است. برای پنهان کردن شماره‌ها شرکت با فراهم کنندگان مختلفی قرارداد دارد که ما را در این امر برقراری تماس را برعهده دارند. سرویس‌های ما نیاز به فراخوانی این فراهم‌کنندگان دارند. همچنین ستار خود توسط سرویس‌های دیگری از ورتیکال‌های دیگر نیز فراخوانی می‌شود و از این رو سرویس ستار کیت توسعه سرویس (SDK) نیز دارد. به علت وابستگی بالای این سرویس به پایگاه‌های داده‌ای مختلف و علم به اینکه پایگاه‌های داده‌ای برای زیرساخت‌های ابری مناسب نیستند، تا اکنون این سرویس روی ماشین‌های مجازی بوده که یکی از تسک‌های مهم آن بردن این سرویس بر روی ابر است. همچنین این سرویس تا کنون برای حالت اسنپ برای دیگری فعال نبوده که یکی از تسک‌های مهم اضافه کردن این ویژگی به آن است که طبیعتا باعث می‌شود نیاز داشته باشیم با سرویس در شرکت به نام مورفیوس که وظیفه‌ی ارسال پیامک را دارد جهت اطلاع‌رسانی شماره‌ی پنهان شده به کاربر نیز در ارتباط داشته باشیم. اولین مشکلی که در در این سرویس وجود داشت بحث نشت حافظه آن بود. زمانی که در سیستم مانیتورینگ به عملکرد این سرویس در یک بازه‌ی زمانی نسبتا بزرگ نگاه می‌کردیم قابل مشاهده بود که حافظه مصرفی این سرویس با شیب کمی همواره در حال افزایش است. البته این حل این مشکل اولویت شرکت نبوده چرا که اولا اگر این سرویس از کار بیافتد نهایتا شماره‌ها پنهان نخواهند شد که این امر در عملکرد کلی شرکت خللی وارد نمی‌کند و از طرفی چندین نسخه از این کد بالا آورده شده است که هر زمان ممووری مصرفی هر کدام از حد مشخصی عبور کرد آن نسخه با توجه به تنظیمات صورت گرفته به طور خودکار راه‌اندازی مجدد خواهد شد. تسک اول من جستجو در کد و تحقیق جهت پیدا کردن مشکل نشتی حافظه و حل آن بود. در جهت جستجو برای حل این مشکل اقدامات زیر انجام شد.

کد این سرویس نسبتا زیاد است و خواندن خط به خط آن احتمالا زمان بسیار زیادی می‌خواست بنابراین روش بهتر این بود که یک نسخه از کد را داخل محیط تستی شرکت بالا بیاورم و سپس آن را مانیتور کنم. اگر بدون اینکه کد زیر بار باشد سایز هیپ[۱۷] مرتبا افزایش

[16] NDA
[17] Heap

پیدا کند به احتمال زیاد باید یک گوروتین[۱۸] داخل کد وجود داشته باشد که در پس زمینه همواره در حال اجرا است و مشکلی در آن وجود دارد. البته سایز هیپ در حالتی که ریکویستی زده نمیشد افزایش پیدا نمیکرد پس مجبور شدم آن را زیر باز ببرم این کد چندین API داشت و هر یک را لود تست و به سایز مموری رجوع می‌کردم. هر API که زیر باز قرار گرفتن آن باعث افزایش سایز مموری شود سر نخ خوبی برای پیدا کردن مشکل است. برای لود تست کردن از کتابخانه‌ی bombardier[۱۹] و تکه کد زیر استفاده کردم.

```
#!/bin/bash

for i in {1..5} ; do
  for j in {1..2000};do
    curl -v -X POST -d '{"enable": true}' \
        -H 'Content-Type: application/json' \
        https://my-service.io/api/method
    sleep .5
  done
  sleep 5m
  curl -L https://my-service.io/debug/pprof/heap > heap.$i.pprof
done
```

بعد از پیدا کردن API مورد نظر شروع به خواندن کد مربوط به آن API کردم. نکته‌ی اول که باید مد نظر قرار دهیم این است که یک کانکشن باز یا هر مورد دیگری نمی تواند باعث مموری لیک شود اگر کد دچار مموری لیک شده یعنی داریم تکه کد مخربی را داخل یک حلقه برای مدت زمان زیادی اجرا می‌کنی.م هنگام خواندن کد باید به نکات مهمی داخل گولنگ توجه کنیم از جمله

۱. رفرنس‌ها
۲. کانتکست‌ها
۳. کانکشن‌ها
۴. تیکرها

به کدی که در ادامه می‌آید دقت کنید. یک نمونه از دیتابیس ساخته شده و به تابع داده شده است. سپس داخل تابع یک استراکت ساخته شده که نمونه دیتابیس به عنوان یک فیلد از آن استراکت قرار گرفته است. ممکن است انتظار داشته باشیم بعد از برگشتن از تابع Garbage Collector استراکت ساخته شده را از بین ببرد اما این اتفاق نمی افتد. چرا که این استراکت به نمونه دیتابیس رفرنس دارد که نمیتواند از بین برده شود. این اتفاق زمانی می افتد که با اشاره‌گرها کار می‌کنیم.

```
package main

import (
```

[18] Goroutine
[19] https://github.com/codesenberg/bombardier

```
        "database/sql"
        "log"
)

type Struct struct {
        db *sql.DB
        // some other fields
}

func main() {
    db, err := sql.Open("driverName", "dataSourceName")
    if err != nil {
        log.Fatalf("Cannot open database dataSourceName: %s", err)
    }

    DoSomething(db)
}

func DoSomething(db *sql.DB) {
    s := Struct{db: db}
    // do something
}
```

فراموش نکنید که کانتکست ها را کنسل کنید.

```
package main

import "context"

func main() {
    ctx, cancelCtx := context.WithCancel(context.Background())
    defer cancelCtx()
}
```

هر سوکت که باز می‌شود باید حتما بسته شود. البته اکثرا از کتابخانه‌ها برای باز کردن سوکت استفاده می‌کنیم. این کتابخانه ها خود موارد مختلفی را هندل می‌کنند تا بعد از اتمام کار ما سوکت بسته شود. اما به هر حال باید به آن توجه کنیم چراکه حالاتی وجود دارد که سوکت باز می‌ماند به تکه کد زیر دقت کنید.

```
package main
```

```go
import (
    "database/sql"
    "log"
)

type Storage struct {
    db *sql.DB
    // some other fields
}

func (s *Storage) fetchAll() {
    // *Rows should be closed
    rows, err := s.db.Query("SELECT * FROM somewhere")
    if err != nil {
            log.Fatal(err)
    }

    defer func() {
        err := rows.Close()
        if err != nil {
                log.Fatal(err)
        }
    }()

    if err := rows.Err(); err != nil {
        log.Fatal(err)
    }

    // If Next is called and returns false and there are no
    // further result sets, the Rows are closed automatically
    // but an error may occur inside the block
    for rows.Next() {
        // If this error or any other one occurs, this loop doesn't continue
        // so if I hadn't called close in a defer function we would have an
        // open connection forever
        err := rows.Scan("&dest")
        if err != nil {
            return
        }
    }
}

func main() {
    // usually we make a single db instance and use
    // it during the whole project life,
```

```
    // so it's rare to call Close function on it
    db, err := sql.Open("driverName", "dataSourceName")
    if err != nil {
        log.Fatalf("Cannot open database dataSourceName: %s", err)
    }

    s := Storage{db: db}
    s.fetchAll()

    // some other code
}
```

دقت کنید که تیکرها بعد از شروع، متوقف شوند.

```
package main

import "time"

func main() {
    ticker := time.NewTicker(time.Second)
    defer ticker.Stop()
}
```

مشکل کد که باعث نشتی حافظه می‌شد به دلیل ساختن تیکرها داخل حلقه و متوقف نکردن آن‌ها بود. تکه کدی مانند زیر

```
package main

import "time"

func main() {
    go func() {
        for time.Now(); true; <-time.NewTicker(time.Second).C {
            // Do something
        }
    }()

    // Do something
}
```

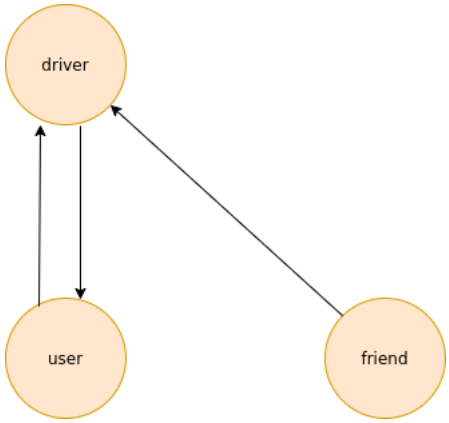
برای آنکه مطمئن شوم نشتی حافظه به دلیل تکه کد بالا به وجود آمده است با تکه کد زیر آن را تست کردم:

```
package main

import (
    "fmt"
    "runtime"
    "time"
)

func main() {
    // Below is an example of using our PrintMemUsage() function
    // Print our starting memory usage (should be around 0mb)
    PrintMemUsage()

    ticker := time.NewTicker(time.Millisecond)
    defer ticker.Stop()
    for time.Now(); true; <-ticker.C {
        a := 2
        a *= 2
        if a == 1024 {
            a = 2
            // Force GC to clear up, should see a memory drop
            runtime.GC()
        }
        PrintMemUsage()
    }
}

// PrintMemUsage outputs the current,
// total and OS memory being used.
// As well as the number
// of garage collection cycles completed.
func PrintMemUsage() {
    var m runtime.MemStats
    runtime.ReadMemStats(&m)

    // For info on each,
    // see: https://golang.org/pkg/runtime/#MemStats
    fmt.Printf("Alloc = %v MiB", bToMb(m.Alloc))
    fmt.Printf("\tTotalAlloc = %v MiB", bToMb(m.TotalAlloc))
    fmt.Printf("\tSys = %v MiB", bToMb(m.Sys))
    fmt.Printf("\tNumGC = %v\n", m.NumGC)
}
```

برنچی از مستر گرفتهو این تکه کد که باعث نشتی حافظه می‌شد را تغییر دادم و مرج ریکویست را قرار دادم. پس از مرور برنچ توسط مدیر ورتیکال این برنچ با مستر مرج شد.

۳.۱ هفته سوم - ۱۵ام مرداد ماه

در هفته‌ی سوم فرصت برای توسعه‌ی بیشتر روی سرویس ستار به وجود آمد. در هنگام گرفتن سفر در اکانت می‌توانید مشخص کنید که این سفر را برای خوتان می‌خواهید یا برای دوستتان. اگر سرویس پنهان کردن شماره را فعال کرده باشید و اسنپ را برای خودتان بخواهید هیچ مشکلی نخواهید داشت اما اگر برای فرد دیگری سفر بگیرید شماره‌ها پنهان نخواهند شد. تسک این هفته اضافه کردن این قسمت به کد بود. با توجه به اینکه کد بسیار ماژولار بود و هر عملکرد یکتایی تابع و استراکت خود را داشت اضافه کردن این فیچر کار بدون دردسری بود. در ابتدا تنها دو نفر داخل مساله بودند حال با اضافه کردن نفر سوم گراف ذهنی ما باید تغییر پیدا کند.



در گراف بالا نود دوست اضافه شده است که باید به کد نیز اضافه میشد علاوه بر آن زمانی که سفر برای دوست گرفته می‌شود و سرویس ستار فعال است باید از طریق پیامک شماره‌ی پنهان شده‌ی راننده به فرد سوم مورد نظر اطلاع‌رسانی شود. به همین دلیل باید با سرویس مورفیوس که وظیفه‌ی پیام رسانی را دارد نیز ارتبط می‌داشتیم. همچنین همانطور که بیان شد سرویس ستار توسط سرویس‌های دیگری داخل شرکت صدا زده میشد که از SDK استفاده می‌کردند به همن جهت SDK این سرویس نیز باید تغییر داده میشد و به تیم‌هایی که این سرویس را کال می کردند اطلاع رسانی میشد.

پس از اضافه کردن فیچر، ریویو توسط مدیر مستقیم ورتیکال و رفت و آمد برای انجام تغییرات درخواست مدیر ورتیکال، نوبت به تست آن می‌شود چون هیچ فیچر جدید یا تغییر زیاد در کد بدون امضای نیرو کیفیت سرویس نمی‌تواند منتشر شود. برای تست کردن باید ابتدا کد را داخل محیط تستی دیپلوی می‌کردم و در ارتباط با کیفیت سرویس می‌بودم تا اگر باگی در کد وجود داشت با مانیتور کردن لاگ‌ها هنگام تست متوجه اشکال میشدم. پس از تست کیفیت سرویس برنچ در برنچ مستر مرج می‌شود حال برای بالا بردن این فیچر باید کد جدید روی سرورها دیپلوی شود. دیپلوی حتما باید در نیمه شب انجام شود تا اگر به هر دلیل

اگر خطایی به وجود آمد، ضرر کمتری به تجارت شرکت وارد شود. تنها مدیر مستقیم ورتیکال اجازه‌ی دیپلوی و دسترسی SSH به سرورها پروداکشن دارد علاوه بر او باید فرد دیگری که علاوه بر توسعه دهنده‌ی فیچر است حضور داشته باشند و در زمان دیپلوی به دقت سرویس را مانیتور کند و اگر تغییر غیر عادی در ترافیک سرویس‌ها دید باید فیچر سریعا رول‌بک شود. لازم به ذکر است برخی از فیچرها که بسیار حساس باشند علاوه بر تایید مدیرمستقیم ورتیکال به تایید مدیر چپتر نیز نیاز دارند.

گروه‌های رسمی ای در پیام‌رسان سازمانی تشکیل می‌شوند که یکی از آن‌ها مربوط دیپلوی سرویس‌ها است. هر تیم قبل از ریلیز باید در آن گروه اعلام کند که در حال دیپلوی است و نام سرویس را ذکر کند تا اگر مشکلی به وجود آمد تمامی افراد از جمله دواپس بدانند که ریلیزی صورت گرفته و تمامی احتمالات را در نظر بگیرند.

۴.۱ هفته چهارم - ۱۲ مرداد

در این هفته بحث بردن سرویس ستار روی ابر مطرح شد. همانطور که در گزارش هفته‌های پیش بیان کردم این سرویس به علت وابستگی به چندین پایگاه داده و عدم کارکردن مناسب پایگاه‌های داده با ابر، این سرویس روی ماشین‌های مجازی در حال سرویس‌دهی بود، اما با توجه به سیاست‌های شرکت همه‌ی تیم‌ها سعی می‌کنند تا تمامی سرویس‌های خود را روی ابر بالا بیاورند تا نگه‌داری آن‌ها به عهده‌ی تیم ابر اسنپ باشد و تیم‌ها خود درگیری کمتری برای نگه‌داری آن‌ها بعد از دیپلوی داشته باشند. به همین جهت در این هفته من باید کار با Kubernete و ابر را یاد می‌گرفتم. روزهایی در هفته صرف یادگرفتن چگونگی کارکرد Kubernetes و معماری پشت آن شد تا درک عمیق تری از اتفاقاتی که در پشت آن می‌افتد پیدا کنم. همچنین باید روی سیستمی که شرکت در اختیار من گذاشته بود Docker نصب می‌کردم که به علت تحریم‌ها برای نصب و استفاده از این سرویس باید پشت چندین سرویس VPN قرار می‌گرفتم که دردسرهای زیادی داشت و وقت زیادی صرف آن شد. پس از آن باید فایل‌های YAMLای می‌نوشتم که به وسیله‌ی آن ها سرویس روی ابر دیپلوی می‌شود که در واقع به آن‌ها manifestهای سرویس گفته می‌شود. البته استفاده از Helm Chart به جای دستی نوشتن فایل‌های manifest کار توسعه و نگهداری را بسیار راحت‌تر می کرد اما ورتیکال یوزر هنوز به آن روی نیاورده و باید فایل‌های manifest دستی نوشته شوند.

Kubernetes دارای manifestهای متنوعی است که می‌بایست با توجه به نوع سرویس از آن‌ها استفاده شود. این manifestها در جهت توصیف pod که کوچکترین عنصر در محیط Kubernetes بوده و خود از چند container تشکیل شده است تا عناصر بزرگی مانند Deployment و ...استفاده می‌شوند.

در سرویس ستار ما از Deployment استفاده کردیم چرا که نمونه‌های مختلف ستار به یکدیگر وابسته نبوده و می‌توانند مستقل فعالیت کنند. به این ترتیب مدیریت آن‌ها کامل به Kubernetes سپرده می‌شود تا در صورت خرابی یا افزایش لود و ...تعداد podها را افزایش یا کاهش دهد. برای آشنایی بیشتر شایان ذکر است که Deployment در واقع مدیریت podها را بر عهده می‌گیرد و آن‌ها افزایش یا کاهش می‌دهد. در ضمن سلامت آن‌ها را نیز مطابق با آنچه در manifest بیان شده است مانیتور می‌کند.

برای دسترسی به سرویس‌ها می‌بایست پورت‌های آن‌ها در قالب یک manifest تعریف شود. این Service manifest نام دارد. برای سرویس ستار نیز یک Service تعریف شده و پورت‌های آن تعریف گشت.

هر پروژه نیازمنده تنظیمات است و این تنظیمات در قالب mainfestهای ConfigMap و Secret تعریف می‌شوند. در نظر داشته باشید که تنظیمات شما چه طریق فایل و چه طریق متغیرهای محیطی تعریف شده باشند این manifestها برای شما مورد نیاز خواهند بود.

دست آخر ستار می‌بایست توسط سایر سرویس‌های شرکت که ممکن است روی ابر باشند یا نباشند مورد فراخوانی قرار می‌گیرد. از این روی برای آن یک manifest به نام Route تعریف می‌شود. که اجازه تعریف URL برای دسترسی به سرویس را می‌دهد. این URLها خود می‌توانند به صورت Public و Private باشند.

در نهایت من این فایل‌ها را حاضر کردم و بنابر استاندارد تیم فایل اسکریپتی برای نصب همزمان همه این manifestها روی ابر نوشتم. این کار روی ابر تستی تیم تست شده و در نهایت آماده برای اجرای روی پروداکشن شد.